

Amortized Analysis of Exponential Time- and Parameterized Algorithms: Measure & Conquer and Reference Search Trees

Daniel Binkele-Raible

June 10, 2010

Dissertation

zur Erlangung des akademischen Grades eines Doktors der
Naturwissenschaften

Dem Fachbereich IV der Universität Trier vorgelegt

Zusammenfassung

Diese Arbeit diskutiert die algorithmische Handhabbarkeit schwieriger kombinatorischer Probleme. Grundsätzlich betrachten wir \mathcal{NP} -schwere Probleme. Für diese Art von Problemen ist es unmöglich Polynomialzeit-Algorithmen zu finden (modulo $\mathcal{P} \neq \mathcal{NP}$). Mehrere algorithmische Ansätze existieren bereits, um diesem Dilemma zu entkommen. Darunter befinden sich (randomisierte) Approximations-Algorithmen und Heuristiken. Obwohl diese in annehmbarer Zeit eine Lösung liefern, ist diese im allgemeinen nicht optimal. Falls wir Optimalität voraussetzen dann gibt es nur zwei Methoden die dies gewährleisten: Exponentialzeit-Algorithmen und parameterisierte Algorithmen. Der erste Ansatz versucht Algorithmen zu finden, die intelligenter handeln als ein trivialer Algorithmus, der einfach alle Lösungskandidaten aufzählt. Typischerweise benötigt solch ein naiver Aufzählungsansatz eine Laufzeit von $\mathcal{O}^*(2^n)$. Deshalb ist die prinzipielle Aufgabe Algorithmen zu entwerfen, die eine Laufzeit der Form $\mathcal{O}^*(c^n)$, wobei $c < 2$ gilt, garantieren.

Der zweite Ansatz betrachtet einen weiteren Parameter k neben der Größe der Eingabe n . Dieser Parameter soll weitere Information über das Problem zu Verfügung stellen und überdies eine typische Charakteristik beschreiben. Die Standard-Parameterisierung sieht k als eine obere (beziehungsweise untere) Schranke für die Größe der Lösung im Fall eines Minimierungsproblems (beziehungsweise Maximierungsproblems). Eine parameterisierter Algorithmus sollte dann in der Lage sein das Problem in einer Laufzeit $f(k) \cdot n^\beta$ zu lösen, wobei β eine Konstante und f unabhängig von n ist. Prinzipiell versucht diese Methode die kombinatorische Komplexität bezüglich des Parameters k zu messen, falls dies überhaupt möglich ist. Grundannahme hierbei ist, dass k relativ klein ist verglichen mit der kompletten Eingabegröße.

In beiden Gebieten ist der Entwurf von Verzweigungs-Algorithmen eine Standard-Technik. Diese Algorithmen lösen das Problem indem sie auf eine ausgeklügelte Art und Weise den Lösungsraum traversieren. Schrittweise wählen sie ein Objekt aus der Eingabe aus und schaffen zwei neue Teilprobleme, eines in dem das Objekt der zukünftigen Lösung zugesprochen wird, und ein weiteres in dem es aus dieser Lösung ausgeschlossen wird. In beiden Fällen kann es sein, dass durch die Fixierung dieses einen Objekts weitere ebenfalls fixiert werden. Ist dies der Fall dann ist die Anzahl der besuchten möglichen Lösungen kleiner als der gesamte Lösungsraum. Diese besuchten Lösungen können als Suchbaum betrachtet werden. Um die Laufzeit solcher Algorithmen zu bestimmen benötigt man eine Methode die scharfe obere Schranken bezüglich der Suchbaumgröße bereitstellt. Zu diesem Zweck wurde im Bereich der Exponentialzeit-Algorithmen eine mächtige Methode entwickelt, die sich *Measure&Conquer* nennt. Sie wurde bereits erfolgreich auf viele Probleme angewandt, insbesondere auf Probleme wo andere Versuche fehlschlügen die triviale Laufzeitschranke zu unterbieten.

Im Gegensatz dazu ist *Measure&Conquer* im Bereich der parameterisierten Algorithmen kaum bekannt. Diese Arbeit wird verschiedene Beispiele präsentieren wo diese Methode in diesem Bereich angewandt werden kann. Darüber hinaus werden Exponentialzeit-Algorithmen für harte Probleme vorgestellt, die *Measure&Conquer* anwenden. Ein weitere Aspekt ist, dass eine Formalisierung (und Generalisierung) des Begriffes Suchbaum

gegeben wird. Es wird gezeigt, dass für bestimmte Probleme diese Formalisierung sehr nützlich ist.

In Kapitel 2 wird in die Technik des *Measure&Conquer* eingeführt. Wir identifizieren die Unterschiede zwischen der klassischen Methode der Suchbaumgrößenabschätzung und *Measure&Conquer*. Zu diesem Zweck behandeln wir mehrere Fallbeispiele.

Kapitel 3 umreißt das Problem MAX-2-SAT. Dieses Problem verlangt nach einer Wahrheitswertzuweisung der Variablen, welche die grösstmögliche Anzahl an Klauseln einer bool'schen Formel in KNF erfüllt. Ebenfalls enthält jede Klausel höchstens zwei Literale. Es wird ein Algorithmus präsentiert, der eine Laufzeit von $\mathcal{O}^*(2^{\frac{K}{6.2158}})$ für MAX-2-SAT garantiert, wobei K die Anzahl der Klauseln darstellt. Diese Laufzeit wurde erreicht indem man heuristische Prioritäten bezüglich der Variablen, auf der verzweigt wird, anwendet. Die Implementierung dieser heuristischen Prioritäten ist recht einfach, obwohl sie signifikante Auswirkungen auf die Laufzeiten haben. Die Laufzeitanalyse verfolgt den *Measure&Conquer*-Ansatz.

In Kapitel 4 wird das \mathcal{NP} -schwere Problem einen Spannbaum mit einer grösstmöglichen Anzahl interner Knoten betrachtet. Dieses Problem ist eine Generalisierung des bekannt HAMILTONIAN PATH Problems. Wir geben Algorithmen basierend auf Dynamischem Programmieren für den allgemeinen und gradbeschränkten Fall an, die eine Laufzeit der Form $\mathcal{O}^*(c^n)$, $c \leq 3$, haben. Das Hauptresultat jedoch ist ein Verzweigungsalgorithmus für Graphen mit Maximalgrad drei. Dieser benötigt nur polynomiellen Platz und eine Laufzeit von $\mathcal{O}^*(1.8669^n)$, wobei n die Anzahl der Knoten im Graphen ist. Wir zeigen ebenfalls eine Laufzeit von $\mathcal{O}(2.1364^k n^{\mathcal{O}(1)})$, wenn es das Ziel ist einen Spannbaum mit mindestens k internen Knoten zu finden. Beide Laufzeiten werden mit der *Measure&Conquer*-Methode gezeigt, wobei der zweite Fall eine neuartige Anwendungsweise dieser Methode zur Analyse von parameterisierten Algorithmen darstellt.

Kapitel 5 beschreibt einen Algorithmus für das Problem einen gerichteten Spannbaum mit möglichst vielen Blättern zu finden. Diese Art von Spannbaum ist ebenfalls bekannt als out-branching. Der Algorithmus verfolgt das Branch-and-Reduce-Paradigma und erreicht eine Laufzeit von $\mathcal{O}^*(1.9044^n)$. Da die Eingabe auf gerichtete Graphen erweitert wird und wir die Laufzeit primär bezüglich der Anzahl der Knoten abschätzen, unterscheidet sich dieser Algorithmus und seine Laufzeitanalyse von demjenigen aus Kapitel 6 erheblich.

Kapitel 6 umfasst das Problem einen Spannbaum mit möglichst vielen Blättern zu finden. Dieses ist \mathcal{NP} -schwer. Wir stellen einen Algorithmus vor der einen Spannbaum mit mindestens k Blättern in höchstens $\mathcal{O}^*(3.4581^k)$ Schritten findet. Diese Laufzeitabschätzung, vor einem parameterisierten Hintergrund, wird mittels *Measure&Conquer* bewältigt. Analysieren wir denselben Algorithmus bezüglich der Anzahl der Knoten n erreichen wir eine Laufzeit von $\mathcal{O}^*(1.8961^n)$.

Kapitel 7 behandelt das Problem eine maximale irredundante Menge zu finden. Obwohl es eine klare Verwandtschaft zum bekannten DOMINATING SET Problem gibt, ist es bisher nicht gelungen die triviale Schranke mittels Aufzählung, nämlich $\mathcal{O}^*(2^n)$, zu

unterbieten. Wir erreichen dieses Ziel indem wir einen parameterisierten Algorithmus mit Laufzeit $\mathcal{O}^*(3.069^k)$ entwerfen und hierauf aufbauend ein WIN/WIN-Konzept verwenden. Wir nutzen abermals *Measure&Conquer* um die Laufzeitschranke für diesen Algorithmus zu bestimmen. Es hat ebenfalls den Anschein, dass dieser Umweg über einen parameterisierten Algorithmus wichtig ist, da eine direkte Adaption die 2^n -Schranke nicht unterschreitet.

In Kapitel 8 wird ein Exponentialzeit-Algorithmus für das POWER DOMINATING SET Problem entwickelt. Das POWER DOMINATING SET Problem ist eine Erweiterung des bekannten Dominierungsproblem auf Graphen, in einer Weise, dass wir es mit einer zweiten Propagationsregel anreichern: Zu einem gegebenen Graphen $G(V, E)$ ist eine Menge $P \subseteq V$ eine power-dominierende Menge falls jeder Knoten nach der erschöpfenden Anwendung der folgenden zwei Regeln observiert ist. Erstens ist ein Knoten v observiert falls $v \in P$ oder er einen Nachbarn in P hat. Zweitens wird ein unobservierter Knoten u observiert, falls er einen observierten Nachbarn hat, dessen Nachbarschaft bis auf u bereits observiert ist. Es wird gezeigt, dass POWER DOMINATING SET auf kubischen Graphen \mathcal{NP} -schwer bleibt. Es wird ein Algorithmus entwickelt, der dieses Problem mit einer Laufzeit von $\mathcal{O}^*(1.7548^n)$ löst. Hierfür nutzen wir die formalisierte Version des Begriffes Suchbaum, was wir Referenzsuchbaum nennen. Diese Suchstruktur beinhaltet globale, nicht-lokale Zeiger.

Kapitel 9 untersucht das Problem MAXIMUM ACYCLIC SUBGRAPH. Einen grösstmöglichen azyklischen Teilgraphen zu finden gehört zu den Problemen, die sich aus der Sichtweise der parameterisierten Komplexität schwer bewältigen lassen. Wir beschränken uns hier auf spezielle Graphklassen. Genauer gesagt konzipieren zwei effizient Algorithmen (einer ist ein Exponentialzeit-Algorithmus, der andere ist parameterisiert) für sogenannte $(1, \ell)$ -Graphen, eine Graphklasse die kubische Graphen enthält. Die Laufzeiten betragen $\mathcal{O}^*(1.2133^m)$ beziehungsweise $\mathcal{O}^*(1.2471^k)$. Beide Laufzeiten werden mittels der *Measure&Conquer*-Methode gezeigt. Ebenfalls spielt der hier eingeführte Begriffe eines Referenzsuchbaumes eine wichtige Rolle um diese Laufzeiten zu erreichen. Für kubische Graphen erhält man leicht besser Laufzeiten, nämlich $\mathcal{O}^*(1.1798^m)$ und $\mathcal{O}^*(1.201^k)$. Eine Konsequenz hieraus ist, dass CUBIC DIRECTED FEEDBACK VERTEX SET in $\mathcal{O}^*(1.282^n)$ Schritten lösbar ist.

Abstract

This work addresses the algorithmic tractability of hard combinatorial problems. Basically, we are considering \mathcal{NP} -hard problems. For those problems we can not find a polynomial time algorithm (modulo $\mathcal{P} \neq \mathcal{NP}$). Several algorithmic approaches already exist which deal with this dilemma. Among them we find (randomized) approximation algorithms and heuristics. Even though in practice they often work in reasonable time they usually do not return an optimal solution. If we constrain optimality then there are only two methods which suffice for this purpose: exponential time algorithms and parameterized algorithms. In the first approach we seek to design algorithms consuming exponentially many steps who are more clever than some trivial algorithm (who simply enumerates all solution candidates). Typically, the naive enumerative approach yields an algorithm with run time $\mathcal{O}^*(2^n)$. So, the general task is to construct algorithms obeying a run time of the form $\mathcal{O}^*(c^n)$ where $c < 2$.

The second approach considers an additional parameter k besides the input size n . This parameter should provide more information about the problem and cover a typical characteristic. The standard parameterization is to see k as an upper (lower, resp.) bound on the solution size in case of a minimization (maximization, resp.) problem. Then a parameterized algorithm should solve the problem in time $f(k) \cdot n^\beta$ where β is a constant and f is independent of n . In principle this method aims to restrict the combinatorial difficulty of the problem to the parameter k (if possible). The basic hypothesis is that k is small with respect to the overall input size.

In both fields a frequent standard technique is the design of branching algorithms. These algorithms solve the problem by traversing the solution space in a clever way. They frequently select an entity of the input and create two new subproblems, one where this entity is considered as part of the future solution and another one where it is excluded from it. Then in both cases by fixing this entity possibly other entities will be fixed. If so then the traversed number of possible solution is smaller than the whole solution space. The visited solutions can be arranged like a search tree. To estimate the run time of such algorithms there is need for a method to obtain tight upper bounds on the size of the search trees. In the field of exponential time algorithms a powerful technique called *Measure & Conquer* has been developed for this purpose. It has been applied successfully to many problems, especially to problems where other algorithmic attacks could not break the trivial run time upper bound.

On the other hand in the field of parameterized algorithms *Measure & Conquer* is almost not known. This piece of work will present examples where this technique can be used in this field. It also will point out what differences have to be made in order to successfully apply the technique. Further, exponential time algorithms for hard problems where *Measure & Conquer* is applied are presented. Another aspect is that a formalization (and generalization) of the notion of a search tree is given. It is shown that for certain problems such a formalization is extremely useful.

In Chapter 2 we introduce the technique of *Measure & Conquer*. We identify the differences between the classical method of search tree estimation and *Measure & Conquer*.

We provide several case studies for this purpose.

Chapter 3 outlines the problem MAX-2-SAT. In MAXSAT we ask for an assignment of the variables which satisfies the maximum number of clauses for a boolean formula in CNF. We present an algorithm yielding a run time upper bound of $\mathcal{O}^*(2^{\frac{K}{6.2158}})$ for MAX-2-SAT (each clause contains at most 2 literals), where K is the number of clauses. The run time has been achieved by using heuristic priorities on the choice of the variable on which we branch. The implementation of these heuristic priorities is rather simple, though they have a significant effect on the run time. The analysis follows the *Measure & Conquer* paradigm.

In Chapter 4 we consider the \mathcal{NP} -hard problem of finding a spanning tree with a maximum number of internal vertices. This problem is a generalization of the famous HAMILTONIAN PATH problem. Our dynamic-programming algorithms for general and degree-bounded graphs have running times of the form $\mathcal{O}^*(c^n)$ ($c \leq 3$). The main result, however, is a branching algorithm for graphs with maximum degree three. It only needs polynomial space and has a running time of $\mathcal{O}(1.8669^n)$ when analyzed with respect to the number of vertices. We also show that its running time is $2.1364^k n^{\mathcal{O}(1)}$ when the goal is to find a spanning tree with at least k internal vertices. Both running time bounds are obtained via a *Measure & Conquer* analysis, the latter one being a novel use of this kind of analysis for parameterized algorithms.

Chapter 5 presents an algorithm for the problem of finding a directed spanning tree with the maximum number of leaves. This kind of spanning tree is also known as out-branching. The algorithm follows the branch-and-reduce-paradigm and achieves a run time of $\mathcal{O}^*(1.9044^n)$. As the input is generalized to directed graphs and its running time is analyzed primarily with respect to the number of vertices, the structure of the algorithm and its run time estimation differs notably from the one in chapter 6.

Chapter 6 covers the problem of finding a spanning tree in an undirected graph with a maximum number of leaves. It is known to be \mathcal{NP} -hard. We present an algorithm which finds a spanning tree with at least k leaves in time $\mathcal{O}^*(3.4581^k)$. The estimation of the run time is done by using *Measure & Conquer* in a parameterized setting. By analyzing the same algorithm with respect to the number of vertices n we show a run time upper bound of $\mathcal{O}^*(1.8961^n)$.

Chapter 7 addresses the problem of finding a maximum irredundant set. Although this problem is closely related with the famous DOMINATING SET problem up to now the trivial enumeration bound of $\mathcal{O}^*(2^n)$ has not been broken yet. Here this goal is achieved by designing a parameterized algorithm with run time $\mathcal{O}^*(3.069^k)$ and by further on using this algorithm in a WIN/WIN-approach. Once more we extensively use *Measure & Conquer* to derive the run time for this parameterized algorithm. Also it seems that this detour via a parameterized algorithm is indeed crucial to break the 2^n -Barrier, as a direct interpretation of this algorithm fails to break the trivial upper bound.

In Chapter 8 an exponential time algorithm for the POWER DOMINATING SET problem is developed. The POWER DOMINATING SET problem is an extension of the well-known

domination problem on graphs in a way that we enrich it by a second propagation rule: Given a graph $G(V, E)$ a set $P \subseteq V$ is a power dominating set if every vertex is observed after we have applied the next two rules exhaustively. First, a vertex v is observed if $v \in P$ or it has a neighbor in P . Secondly, if an observed vertex has exactly one unobserved neighbor u , then also u will be observed as well. We show that POWER DOMINATING SET remains \mathcal{NP} -hard on cubic graphs. We designed an algorithm solving this problem in time $\mathcal{O}^*(1.7548^n)$ on general graphs. To achieve this we use the formalized version of the notion of search trees called reference search trees providing non-local pointers.

Chapter 9 considers MAXIMUM ACYCLIC SUBGRAPH. Finding a maximum acyclic subgraph is on the list of problems that seem to be hard to tackle from a parameterized perspective. We restrict our attention to special graph classes. More precisely, we develop two quite efficient algorithms (one is exact exponential-time, the other parameterized) for so-called $(1, \ell)$ -graphs, a class containing cubic graphs. The run times are $\mathcal{O}^*(1.2133^m)$ and $\mathcal{O}^*(1.2471^k)$, respectively, determined both by the usage of the *Measure & Conquer* technique. Also the introduced notion of reference search trees plays an important key role in achieving this run time. We derive slightly better run times for cubic graphs, namely $\mathcal{O}^*(1.1798^m)$ and $\mathcal{O}^*(1.201^k)$. As a consequence CUBIC DIRECTED FEEDBACK VERTEX SET can be solved in $\mathcal{O}^*(1.282^n)$ steps.

Preface

The dissertation at hand emerged during my employment at the group of Prof. Dr. Henning Fernau at the University of Trier. I have been working there as a research assistant from May 2006 until October 2009. During this time several publications evolved. Not all of them contribute to this dissertation. As I have focused on search trees in this dissertation only those publications who can be classified into this field have been considered. In the following list they are marked with a \diamond -symbol. Each item in the list states the authors, the title and where it was published.

- The paper ‘The Complexity of Probabilistic Lobbying’ (G. Erdélyi, H. Fernau, J. Goldsmith, N. Mattei, D. Raible and J. Rothe [39]) appeared in the proceedings of *Algorithmic Decision Theory (ADT09)*.
- The paper ‘Kernel(s) for Problems with No Kernel: On Out-Trees with Many Leaves’ (H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh and Y. Villanger [48]) appeared in the proceedings of the *Symposium on the Theoretical Aspects in Computer Science (STACS09)*.
- The paper ‘Exact Exponential-Time Algorithms for Finding Bicliques in a Graph’ (H. Fernau, S. Gaspers, D. Kratsch, M. Liedloff and D. Raible [49]) appeared in the proceedings of the *Cologne-Twente Workshop (CTW09)*.
- \diamond The paper ‘Exact and parameterized Algorithms for Max Internal Spanning Tree’ (H. Fernau, S. Gaspers and D. Raible [50]) appeared in the proceedings of the *International Workshop on Graph-Theoretic Concepts in Computer Science (WG09)*.
- The paper ‘An exact algorithm for the Maximum Leaf Spanning Tree problem’ (H. Fernau, A. Langer, M. Liedloff, J. Kneis, D. Kratsch, D. Raible and P. Rossmanith [51]) appeared in the proceedings of the *International Workshop on Parameterized and Exact Computation (IWPEC09)*.
- The paper ‘Alliances in graphs: a complexity-theoretic study’ (H. Fernau and D. Raible [53]) appeared in the proceedings vol.II of the *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM07)*.
- \diamond The paper ‘Exact Algorithms for Maximum Acyclic Subgraph on a Superclass of Cubic Graphs’ (H. Fernau and D. Raible [54]) appeared in the proceedings of the *Workshop on Algorithms and Computation (WALCOM08)*.
- The paper ‘A Parameterized Perspective on Packing Paths of Length Two’ (H. Fernau and D. Raible [55]) appeared in the proceedings of *Combinatorial Optimization and Applications (COCOA08)* and as a journal version [56] in the *Journal on Combinatorial Optimization*.
- The paper ‘Packing Paths: Recycling Saves Time’ (H. Fernau and D. Raible [57]) appeared in the proceedings of the *Cologne-Twente Workshop (CTW09)*.

- ◇ The paper ‘Searching Trees: an Essay’ (H. Fernau and D. Raible [58]) appeared in the proceedings of *Theory and Applications of Models of Computation (TAMC09)*.
- ◇ The paper ‘A New Upper Bound for MAX-2-SAT: A Graph-Theoretic Approach’ (D. Raible and H. Fernau [133]) appeared in the proceedings of *Mathematical Foundations of Computer Science (MFCS08)*.
- ◇ The paper ‘Power Domination in $O^*(1.7548^n)$ Using Reference Search Trees’ (D. Raible and H. Fernau [134]) appeared in the proceedings of the *International Symposium on Algorithms and Computation (ISAAC08)*
- ◇ The paper ‘A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem’ (D. Raible and H. Fernau [136]) appeared in the proceedings of the *Computer Science Symposium in Russia (CSR10)*
- ◇ The paper ‘A Parameterized Route to Exact Puzzles: Breaking the 2-Barrier for Irredundance’ (Daniel Binkele-Raible, Ljiljana Brankovic, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff and Peter Rossmanith [8]) appeared in the Proceedings of the *International Conference on Algorithms and Complexity (CIAC10)*

Finally, I would like to thank Henning Fernau for his valuable and friendly advice. I have benefited a lot from our common work. I thank all my colleagues at the University of Trier for the nice and productive atmosphere. Also I thank all my co-authors for sharing their ideas with me. A very special dedication goes out to Maria Teresa Binkele.

How to read this thesis?

The first two chapters are introductory chapters. Chapter one gives an overview of known techniques of how to cope with \mathcal{NP} -hard problems. A focus is placed on exponential-time- and parameterized algorithms. Here the reader also finds the basic definitions which are used. It provides also an introduction to run time estimation of branch&reduce algorithms as well as the important definition of reference search trees. Chapter two provides a gentle introduction to branch&reduce algorithms, *Measure&Conquer* and its parameterized pendant. Chapter two is the basis for all sections which are subsumed in the subsequent part I. In part I (*Parameterized Measure&Conquer*) is applied to a variety of problems. Thus, it is recommended to first read chapter two before one proceeds to part I. Part II contains two sections where reference search trees play an important role. Therefore they cannot be read independently from section 1.4 in chapter one. Moreover, the run time analysis also uses *Measure&Conquer*.

Contents

1. Introduction & Basic Definitions	15
1.1. Hard Combinatorial Optimization Problems	15
1.1.1. Approximation Algorithms	15
1.1.2. Heuristics	16
1.1.3. Randomized Algorithms	16
1.1.4. Moderate Exponential Time Algorithms	17
1.1.5. Parameterized Algorithms	18
1.2. Notation & Terminology	20
1.2.1. Graph-Theoretic Notions	20
1.2.2. Notions from Satisfiability	21
1.3. Estimating running times	22
1.4. Reference Search Trees	24
2. Measure&Conquer	29
2.1. Case Study: A Search Tree Algorithm for Vertex Cover	29
2.2. Case Study: A Search Tree Algorithm for Dominating Set	32
2.3. Case Study: Parameterized Measure & Conquer for Nonblocker	36
2.3.1. A Simple Approach	36
2.3.2. An Approach with a fine-grained Measure	39
2.4. Case Study: Connected Vertex Cover	42
2.5. Case Study: Edge Dominating Set	48
2.6. Obtaining The Weights	53
2.6.1. Local Search	53
2.6.2. Convex Programming	54
1. <i>Measure&Conquer</i> applied to Exponential-Time- and Parameterized Algorithms	57
3. A New Upper Bound for Max-2-SAT	61
3.1. Introduction	61
3.1.1. Our Problem	61
3.1.2. Results So Far	61
3.1.3. Our Results	62
3.1.4. Problem Statement	62
3.2. Reduction Rules & Basic Observations	62

3.3.	The Algorithm	65
3.3.1.	Heuristic Priorities	66
3.3.2.	Key Ideas	67
3.4.	The Analysis	67
3.4.1.	G_{var} has High Maximum Degree	68
3.4.2.	G_{var} has Maximum Degree Four	68
3.4.3.	The 4- 5- and 6-regular case	72
3.4.4.	The Cubic Case	78
3.5.	Combining Two Approaches	81
3.5.1.	General Exposition	81
3.5.2.	5-regular Branches in the Combined Approach	82
3.5.3.	Analysis of the 6- 4- and 3-phase in the Combined Approach	84
3.6.	Conclusion	87
4.	Exact and Parameterized Algorithms for MAX INTERNAL SPANNING TREE	89
4.1.	Introduction	89
4.2.	The Problem on General Graphs	91
4.3.	Subcubic Maximum Internal Spanning Tree	93
4.3.1.	Observations	93
4.3.2.	Reduction Rules	94
4.3.3.	Triangles	96
4.3.4.	The Algorithm	98
4.3.5.	An Exact Analysis of the Algorithm	98
4.3.6.	A Parameterized Analysis of the Algorithm	104
4.4.	Conclusion & Future Research	110
5.	A Faster Exact Algorithm for Directed Maximum Leaf Spanning Tree	111
5.1.	Introduction	111
5.1.1.	Known Results.	111
5.1.2.	Our Achievements.	112
5.1.3.	Preliminaries, Terminology & Notation	112
5.1.4.	Basic Idea of the Algorithm	113
5.2.	The Polynomial Part	113
5.2.1.	Halting Rules	113
5.2.2.	Reduction rules	114
5.3.	The Exponential Part	115
5.3.1.	Branching rules	115
5.3.2.	Correctness of the algorithm	115
5.3.3.	Analysis of the Run Time	118
5.4.	Conclusions	124
5.4.1.	An Approach Using Exponential Space	124
5.4.2.	Résumé	124
6.	Parameterized Measure&Conquer for k-Leaf Spanning Tree	127

6.1.	Introduction.	127
6.1.1.	Our Contributions.	128
6.1.2.	Terminology.	128
6.1.3.	Overall Strategy.	128
6.2.	Reduction Rules & Observations.	129
6.2.1.	Reduction Rules.	129
6.2.2.	Observations.	131
6.3.	The Algorithm.	134
6.3.1.	Correctness.	134
6.3.2.	Run Time Analysis.	139
6.4.	An Exact Exponential Time Analysis	141
6.5.	Conclusions.	142
7.	Breaking the 2^n-Barrier for irredundance	145
7.1.	Introduction	145
7.2.	Preliminaries and a Linear Kernel	146
7.2.1.	A Linear Kernel	147
7.2.2.	Basic Facts	148
7.3.	Measure & Conquer Tailored To The Problem	149
7.3.1.	Reduction Rules	149
7.3.2.	The Algorithm	152
7.4.	Conclusions	160
II.	Applications of Reference Search Trees	163
8.	An Exact Exponential Time Algorithm for POWER DOMINATING SET	167
8.1.	Introduction	167
8.1.1.	Discussion of Related Results	168
8.1.2.	New Results	168
8.1.3.	Terminology and Notation	169
8.2.	\mathcal{NP} -hardness of Planar Cubic Power Dominating Set	169
8.3.	An Exact Algorithm for Power Dominating Set	171
8.3.1.	Annotated Power Dominating Set	171
8.3.2.	Algorithm	172
8.4.	Conclusion and Further Perspectives	180
9.	Exact Algorithms for Maximum Acyclic Subgraph on a Superclass of Cubic Graphs	183
9.1.	Introduction and Definitions	183
9.1.1.	Our problems	183
9.1.2.	Motivation.	183
9.1.3.	Discussion of related results.	184
9.1.4.	Our contributions.	185

Contents

9.1.5. Fixing terminology	185
9.2. The Algorithm	185
9.2.1. Preprocessing	185
9.2.2. Reduction Rules	187
9.2.3. The Concrete Algorithm	189
9.3. The Analysis	189
9.3.1. Analyzing the Reduction Rules	191
9.3.2. Analyzing the Algorithm	195
9.4. Reparameterization	205
9.5. Conclusions	205
9.6. Recursions and run times	207
9.6.1. The maximum degree three case	207
10. Conclusion	209
11. Appendix	213
11.1. Code For Local Search	213
11.1.1. Meshsearch.m	214
11.1.2. Mshlp	215
11.2. AMPL Code for Convex Programming	216

Chapter 1.

Introduction & Basic Definitions

1.1. Hard Combinatorial Optimization Problems

Throughout this work the main task is to design efficient algorithms for combinatorial optimization problems. Generally, efficiency will be measured with respect to the input size n of the particular instance of the problem. An algorithm which consumes $\mathcal{O}(n^2)$ time is widely considered as efficient. Generally, algorithms whose run time can be upper bounded by $\mathcal{O}(n^c)$ (c is a constant) are accepted to be efficient. They are said to have a polynomial run time. Regardlessly, there are numerous optimization problems arising from applications which resist to be efficient in the above sense. For these problems only algorithms whose run times are upper bounded by expressions of the form $\mathcal{O}(c^n)$ are known. For these kind of problems we try to design algorithms which are efficient in a broader sense.

In this section we assume the reader is conscious of basic concepts of classical complexity theory (i.e. theory of \mathcal{NP} -hardness) as described in C. H. Papadimitriou [127]. In 1971 S. Cook [29] showed \mathcal{NP} -hardness of the satisfiability problem (SAT). He reduced NON-DETERMINISTIC TURING ACCEPTANCE to SAT via a polynomial time reduction. Since then \mathcal{NP} -hard problems basically are seen as practical infeasible. One year later in 1972 R.M. Karp [97] presented 21 further problems which he proved to be \mathcal{NP} -hard. Among them were prominent problems like FEEDBACK ARC SET, SET COVER, VERTEX COVER, STEINER TREE and 3-SAT. Afterwards many interesting problems, many of them evolving from practical applications, were proven to be \mathcal{NP} -hard, see M. R. Garey and D. S. Johnson [77] for an overview until 1979.

There evolved many follow-up lines of research which promise to be a way out of this dilemma. See the book of J. Hromkovič [94] for a gentle and far more detailed introduction to the following fields.

1.1.1. Approximation Algorithms

This approach tries to circumvent the inefficiency problem for \mathcal{NP} -hard problems by relaxing the constraint that solution must be optimal. Here we are satisfied if we are given some solution which differs from the optimal solution by some percentage in quality. For example if in case of an minimization problem we can guarantee that an approximate solution is at most greater in cost by a constant factor of, say, $c := 1.5$. On the one hand

we give up the aim of optimality but on the other we constrain the algorithm to have a polynomial run time. There are also examples where we can get as close as possible to an optimal solution. For any given $\epsilon > 0$ we can find an approximate solution differing by a factor of ϵ and still consuming only polynomial time. Nevertheless, in many cases we only can achieve a run time of $\mathcal{O}(n^{\frac{1}{\epsilon}})$, such that the exponent rapidly grows in ϵ . Regrettably, this approach fails for certain problems in the sense that any polynomial time algorithm cannot provide a constant factor approximation. For example, for SET COVER a factor of only $\log n$ is possible (assuming $\mathcal{P} \neq \mathcal{NP}$). This deficiency is often not acceptable in practice.

For a detailed introduction into the field we refer to [3, 154].

1.1.2. Heuristics

A heuristic in a broad sense is a simple algorithm following a transparent and straightforward strategy. Usually, one cannot make any statements about the quality of the solution. Beyond that in many cases a proper upper bound on the run time can be given. The main idea is that the heuristic behaves well on typical instances of the optimization problem at hand. A very basic heuristic is local search. Starting from some initial solution we proceed on and on by picking some neighboring solution of higher quality. What exactly the neighborhood consists of is problem dependent. One further advantage lies in its simplicity which permits an easy implementation process.

Another advantage is that they follow a very general strategy which can be applied to a vast number of different optimization problems. Heuristics with this property are often referred to as meta-heuristics. Two well-known meta-heuristics are simulated annealing and genetic algorithms. Both strategies imitate some real optimization process.

Simulated annealing emulates a thermodynamic process. It can be viewed as a clever way of applying local search. To prevent the algorithm to get stuck in a local optimum it is allowed to move to a worse solution with some probability. This probability of deterioration of solution quality is dependent on the thermodynamic process.

Genetic Algorithms imitate the natural selection process of individuals in nature. Starting from a set of solutions at each stage a process of evolution is applied. Existing solutions are paired (or combined) resulting in new solutions. Also spontaneous mutations of a solutions (i.e. small changes) are possible at this point. After this part the solutions are evaluated using a fitness function. Only those solutions survive whose fitness is greater than a given threshold.

As a good starting point into the field of heuristics we propose [81, 117].

1.1.3. Randomized Algorithms

Randomized Algorithms can be seen as deterministic algorithms who get as an additional input a sequence s of random bits. The algorithm will make choices depending on s . Moreover, a randomized algorithm can be seen as a set of algorithms where one randomly is chosen for the given input (depending on the sequence s).

There are two kinds of randomized algorithms. So-called Monte-Carlo algorithms return

an answer which is dependent on the random bits in polynomial time. A Las-Vegas algorithm always returns the correct answer but its run time may vary. We can express it this way: In a Monte-Carlo algorithm the out-put is a random variable and in a Las-Vegas algorithm the run time is a random variable.

A randomized algorithm maybe more efficient and more practical than the best known deterministic counterparts. One could seek to de-randomized such an algorithm. But it is not known if this is possible without paying a great amount of resources. Moreover, a fact which is quite crucial in our case: Up to now no polynomial time randomized algorithm for a \mathcal{NP} -hard problem is known ([94]). See also [121, 118] for introductory books.

A way out of this maybe randomized approximation algorithms. Here the approximation ratio can be seen as a random variable. This seems to make sense if a problem resisted the approximation approach. Then it possibly can be made tractable by additionally using randomization. See the book [3] for a good starting point.

1.1.4. Moderate Exponential Time Algorithms

All algorithmic approaches up to now were seeking to establish tractability for \mathcal{NP} -hard problems by making compromises. Either we relax the requirement that a returned solution is optimal or we get a solution with some uncertainty. But if we insist on a certain optimal solution then an exponential time algorithm seems to be unavoidable. Once one has accepted that for \mathcal{NP} -hard problems there is no polynomial time algorithm modulo the assumption $\mathcal{P} \neq \mathcal{NP}$, one seeks to lower the constant c in the run time $\mathcal{O}^*(c^n)$ (which is the standard form for a vast number of problems). The standard vertex selection problem is to find a subset of vertices $V' \subset V$ for a given graph $G(V, E)$ smallest in cardinality and with a certain property. Enumerating all such subsets results in the trivial brute force algorithm with run time $\mathcal{O}^*(2^n)$. Quite early at the time where \mathcal{NP} -completeness theory was established researchers have experienced that for some problems this is not a run time lower bound:

- ☆ E. Horowitz and S. Sahni [93] designed an algorithm for BINARY KNAPSACK with a run time of $\mathcal{O}^*(2^{\frac{n}{2}}) \subseteq \mathcal{O}^*(1.414^n)$.
- ☆ M. Held and R. M. Karp [91] presented an algorithm solving TRAVELING SALES MAN in $\mathcal{O}(2^n)$ steps whereas the naive algorithm needs $n! \approx n^n/e^n$ steps.
- ☆ E.L. Lawler [109] was able to enumerate all maximal independent sets in $\mathcal{O}^*(1.4423^n)$. By this 3-COLORING could be solved within the same time bound. Additionally, he solved CHROMATIC NUMBER in $\mathcal{O}^*(2.4423^n)$.
- ☆ R.E. Tarjan and A.E. Trojanowski [148] stated a run time upper bound for MAXIMUM INDEPENDENT SET of $\mathcal{O}^*(1.2599^n)$.
- ☆ SATISFIABILITY was the subject of B. Monien and E. Speckenmeyer [120]. In case of 3-SAT a run time of $\mathcal{O}^*(1.6181^n)$ was shown.

Regardless, many important and famous problems resisted to be solvable faster than their trivial algorithm. Only recently progress was made which is due to new algorithmic techniques. Some problems own the characteristic of a certain non-locality like MAXIMUM LEAF SPANNING TREE and FEEDBACK VERTEX SET. For others like DOMINATING SET unexpectedly for a long time there had not been any improvement. These problems now admit run times of $\mathcal{O}^*(1.8966^n)$ (H. Fernau *et al.* [51]), $\mathcal{O}^*(1.7548^n)$ (F.V. Fomin *et al.* [64]) and $\mathcal{O}^*(1.5048)$ (J. M. M. van Rooij *et al.* [153], respectively). The authors used an approach for run time estimation which is called *Measure&Conquer*. We provide a case study in chapter 2 for the purpose of a gentle introduction to the field. Consider also the PhD Thesis of S. Gaspers [78] and M. Liedloff [112] (in french) for an overview of the topic.

Finally, let us mention a bit about the practicality of exact exponential algorithms. For the sake of practicality we hope for a small base term c . If we can achieve that $c = 1.2$ then $1.2^n < n^3$ for $n \leq 69$ and $1.2^n < n^4$ for $n \leq 101$. Usually, algorithms which are based on a branching procedure even behave more favorable in the average case. The stated run time refers to a worst case scenario. It is often very hard to even come up with an example instance where the run time bound is tight for the given algorithm. In almost all cases there is a notable gap between run time upper bound of the algorithm and the particular lower bound of it. Thus, if c is small enough then the algorithm contributes to the tractability of the problem in practice.

To further lower the constant c some authors took an approach called *Exponential Time Approximation*. Here again we relax the condition of optimality. But we gain by lowering the bases c . See M. Cygan and M. Pilipczuk [31] and Fürer *et al.* [75] for some discussions on BANDWIDTH concerning this topic. Further see Cygan *et al.* [30] dealing with WEIGHTED SET COVER and exponential time approximation.

1.1.5. Parameterized Algorithms

This approach like the previous one seeks to solve problems to optimality. But beyond that it also takes also the particular problem into account. In exponential time algorithmics all that matters is the input size n . In parameterized complexity analysis we also take into account a parameter k . This parameter should capture further information of the problem. This parameter can be, just to mention a few examples, the treewidth of the given graph, an upper (lower) bound on the solution size, the number of leaves in a spanning tree or the chromatic number of the given input graph. There are even occasions when graph parameters like the vertex cover number are subject to parameterization (J. Fiala *et al.* [60], M.R. Fellows *et al.* [46]).

Now one wishes to measure the complexity of the problem not only in the input size n but also with respect to the parameter k . Assume that the function in n and k which measures this complexity is only slowly growing in n , say $4^{k^2}n^2$. Then the problem is tractable for small values of k . So, the right choice of the parameter is crucial for this approach to succeed. We now provide some brief and formal introduction.

Definition 1.1.1. A parameterized problem is a subset $\mathcal{P} \subseteq \Sigma^* \times \Sigma^*$.

In this definition Σ is the alphabet for coding the problem. For convenience we only consider problems $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$ as only such problems are subject of this thesis.

Definition 1.1.2 (Fixed Parameter Tractability). A parameterized problem \mathcal{P} is fixed-parameter tractable if there is an algorithm that correctly decides for an input $(I, k) \in \Sigma^* \times \mathbb{N}$, whether $(I, k) \in \mathcal{P}$ in time $f(k) \cdot n^c$. Here n is the input size ($n = |I|$), k the parameter, c a fixed constant independent of k and n , and f is an arbitrary function independent of n .

By \mathcal{FPT} we denote the class of all fixed-parameter tractable problems. If for an instance (I, k) we have $(I, k) \in \mathcal{P}$ then it is a **YES**-instance and otherwise a **NO**-instance. An algorithm satisfying the requirements of Definition 1.1.2 is called a *parameterized algorithm*.

Equivalently, one can define the class of fixed-parameter tractable problems as follows: strive to find a polynomial-time transformation that, given an instance (I, k) , produces another instance (I', k') of the same problem, where $|I'|$ and k' are bounded by some functions $h(k)$ and $g(k)$.

Definition 1.1.3. A kernelization for a problem \mathcal{P} is a polynomial time algorithm with the following properties:

1. It transforms any instance (I, k) to an instance (I', k')
2. $k' \leq g(k)$
3. $|I'| \leq h(k)$
4. (I, k) is a **YES**-instance if and only if (I', k') is a **YES**-instance

Here h and g are arbitrary functions which are not dependent on n . The instance (I', k') will be called the *kernel*.

Kernelization informally can be viewed as a kind of pre-processing providing also an upper bound on the size of the resulting instance (with respect to k). We mention that through out this document g will always be the identity. As mentioned we have the following relation

Lemma 1.1.1 ([35]): A problem is in \mathcal{FPT} if and only if it admits a kernel.

Consequently, there are two methods of showing that a problem belongs to \mathcal{FPT} . It follows also that for every problem two lines of improvement open up. Firstly, try to improve on the function $f(k)$ in the run time upper bound $f(k) \cdot n^c$ of the parameterized algorithm which decides the problem. Improving means to find a function f which is only slowly increasing. Secondly, try to improve on h which gives an upper bound on the kernel.

Nevertheless, there is a not negligible number of problems who resisted any algorithmic

attack in order to show their \mathcal{FPT} membership. For this purpose a complexity hierarchy based on \mathcal{FPT} has been introduced:

$$\mathcal{FPT} \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[\text{Sat}] \subseteq W[P] \subseteq XP$$

To complete the hierarchy we also need a kind of reduction, the *parameterized reduction*. It is basically the standard many-one reduction with an additional constraint. Roughly speaking, the solution size of the instance created by the reduction must be upper bounded by some function only depending on k . There is a strong evidence that all these subset inclusions are strict. Thus, there should be problems in $W[1] \setminus \mathcal{FPT}$. It is commonly believed that SHORT TURING ACCEPTANCE (does a Turing Machine halt in k steps?) is not in \mathcal{FPT} . Actually, it is the defining problem for $W[1]$. So, whenever we can find a parameterized reduction from a $W[1]$ -hard problem to a unclassified problem we can assume that it is not in \mathcal{FPT} .

Valuable introductions are the books of R. G. Downey and M. R. Fellows [35], J. Flum and M. Grohe [62] and R. Niedermeier [125]. The latter one provides a more algorithmic access to the topic. The second book focuses more on complexity theoretical issues.

1.2. Notation & Terminology

1.2.1. Graph-Theoretic Notions

Undirected Graphs

An undirected graph is denoted as $G(V, E)$ where V is the set of *vertices* and $E \subseteq \{\{u, v\} \mid u, v \in V\}$ are the *edges*. Let $n := |V|$ and $m := |E|$. The (*open*) *neighborhood* of a vertex $v \in V$ in G is $N_G(v) := \{u \mid \{u, v\} \in E\}$ and its *degree* is $d_G(v) := |N_G(v)|$. The *closed neighborhood* of v is $N_G[v] := N_G(v) \cup \{v\}$ and for a set $V' \subseteq V$ we let $N_G(V') := (\bigcup_{u \in V'} N_G(u))$. We omit the subscripts of $N_G(\cdot)$, $d_G(\cdot)$, and $N_G[\cdot]$ when G is clear from the context. A *path* P of length ℓ is a sequence of vertices $p_1 \dots p_\ell$ such that $\{p_i, p_{i+1}\} \in E$ for $1 \leq i < \ell$. In a *simple path* we have that the vertices are pairwise different. A cycle of length ℓ is simple path of length ℓ such that $\{p_1, p_\ell\} \in E$ additionally holds. A *subcubic graph* has maximum degree at most three. For some $V' \subseteq V$ let $E_{V'}(v) := \{\{u, v\} \in E \mid u \in V'\}$ and $E(V') = \bigcup_{v \in V'} E_{V'}(v)$. For some $\tilde{E} \subseteq E$ we set $V(\tilde{E}) := \bigcup_{e \in \tilde{E}} e$. $G[V'] = G(V', E(V'))$ and $G[\tilde{E}] = G(V(\tilde{E}), \tilde{E})$ are the graphs induced on V' and \tilde{E} , respectively.

Directed Graphs

We consider directed multigraphs $G(V, A)$ in the course of our later algorithms, where V is the vertex set and A the arc set. From A to V we have two kinds of mappings: For $a = (u, v) \in A$, $init(a)$ denotes the vertex at the end of the arc a and $ter(a)$ at the tip, i.e., $init(a) = u$ and $ter(a) = v$. A *cycle* of length ℓ in a directed graph is a sequence of arcs a_1, \dots, a_ℓ such that $ter(a_i) = init(a_{i+1})$ for $1 \leq i < \ell$ and $ter(a_\ell) = init(a_1)$. An *undirected cycle* is an acyclic arc set in $G(V, A)$ which is a cycle in the underlying undirected

graph $G_u(V_u, E_u := \{\{u, v\} \mid (u, v) \in A\})$. A *directed path* of length ℓ in G is a set of pairwise different vertices v_1, \dots, v_ℓ such that $(v_i, v_{i+1}) \in A$ for $1 \leq i < \ell$. An *undirected path* is an arc set in $G(V, A)$ which is a path in G_u . However, mostly graphs considered are directed graphs without loops and multiple arcs. Let G be a directed graph with the set of vertices $V(G)$ and the set of arcs $A(G)$. Let $v, w \in V(G)$. If $(w, v) \in A(G)$, we say that w is an *entering* neighbor of v and v is a *leaving* neighbor of w . The *in-neighborhood* of a vertex $v \in V$ is $N_{V'}^-(v) = \{u \in V' \mid (u, v) \in A\}$ and, analogously, its *out-neighborhood* is $N_{V'}^+(v) := \{u \in V' \mid (v, u) \in A\}$. The *in- and out-degrees* of v are $d_{V'}^-(v) := |N_{V'}^-(v)|$ and $d_{V'}^+(v) := |N_{V'}^+(v)|$ and its *degree* is $d_{V'}(v) := d_{V'}^-(v) + d_{V'}^+(v)$. If $V' = V$ then we might suppress the subscript. For $V' \subseteq V$ we let $N^+(V) := \bigcup_{v \in V'} N^+(v)$ and $N^-(V')$ is defined analogously.

We distinguish between two kinds of arc-neighborhoods of a vertex v which are $A^-(v) := \{a \in A \mid \text{ter}(a) = v\}$ (the ingoing arcs) and $A^+(v) := \{a \in A \mid \text{init}(a) = v\}$ (the outgoing arcs). We have an in- and outdegree of a vertex, that is $d^-(v) := |A^-(v)|$ and $d^+(v) := |A^+(v)|$. We set $N_A(v) := A^-(v) \cup A^+(v)$. We also define a neighborhood for arcs a $AN(a) := \{N_A(u) \cup N_A(v) \setminus \{a\} \mid a = (u, v)\}$ and for $A' \subseteq A$ we set $AN(A') := \bigcup_{a' \in A'} AN(a')$. For $V' \subseteq V$ we set $A(V') := \{a \in A \mid \exists u, v \in V', \text{init}(a) = u, \text{ter}(a) = v\}$.

Given a graph $G = (V, A)$ and a graph $G' = (V', A')$, G' is a *subgraph* of G if $V' \subseteq V$ and $A' \subseteq A$. The subgraph of G induced by a vertex set $X \subseteq V$ is denoted by $G(X)$ and is defined by $G(X) = (X, A')$ where $A' = A(X)$. The *subgraph* of G induced by an arc set $Y \subseteq A$ is denoted by $G(Y)$ and is defined by $G(Y) = (V(Y), Y)$ where $V(Y) = \{u \in V \mid \exists (u, v) \in Y \vee \exists (v, u) \in Y\}$.

1.2.2. Notions from Satisfiability

Let $V(F)$ be the set of variables of a given Boolean formula F . For $v \in V(F)$ by \bar{v} we denote the negation of v . If v is set, then it will be assigned the values *true* or *false*. By the word *literal*, we refer to a variable or its negation. A *clause* is a disjunction of literals. We represent clauses by sets of literals (e.g., $\{x, \bar{y}\}$). Likewise, we could view a formula as a multi-set of clauses. If l is a literal and C is a clause, then we say that l *occurs* in C if $l \in C$. Likewise, we say that l occurs in a formula F if $l \in C$ for a clause $C \in F$. A variable x *occurs positively* (or *negatively*) in F if $x \in C$ for some $C \in F$ (or $\bar{x} \in C$ for some $C \in F$). The variable x *occurs only positively* in F if x occurs positively in F and \bar{x} does not occur in F . Accordingly, we may express that x *occurs only negatively* in F . If $(x \in C \vee \bar{x} \in C)$, we also say that x *appears* in C ; we also write $x \in V(C)$ for short. We consider formulas in *conjunctive normal form (CNF)*, that is a conjunction of clauses. We allow only 1- and 2-clauses, i.e., clauses with at most two literals.

The weight of v , written $\#_2(v)$, refers to the number of 2-clauses in which v appears, i.e., in which v or \bar{v} occurs. For a set $U \subseteq V(F)$ we define $\#_2(U) := \sum_{u \in U} \#_2(u)$. A set A of literals is called *assignment* if for every $v \in A$ it holds that $\bar{v} \notin A$. Loosely speaking if $l \in A$ for a literal l , then l receives the value *true*. We allow the formula to contain

truth-clauses of the form $\{\mathcal{T}\}$ that are always satisfied. Furthermore, we consider a MAX-2-SAT instance as multiset of clauses. A variable $x \in V(F)$ is a *neighbor* of v , written $x \in N(v)$, if they appear in a common 2-clause. Let $N[v] := N(v) \cup \{v\}$. The *variable graph* $G_{var}(V, E)$ is defined as follows: $V = V(F)$ and $E = \{\{u, v\} \mid u, v \in V(F), u \in N(v)\}$. Observe that G_{var} is an undirected multigraph and that it neglects clauses of size one. We can therefore employ graph-theoretic notions like $G_{var}[U]$ for $U \subseteq V$, referring to the multigraph induced by the vertex set U , or $N(x)$ or $N[x]$ referring to the open or closed neighborhood of x . Observe that $\#_2(v)$ denotes the degree of v in G_{var} , and that $\#_2(V)$ is just twice the number of edges in G_{var} . We will not distinguish between the words “variable” and “vertex”. Every variable in a formula corresponds to a vertex in G_{var} and vice versa. By writing $F[l]$, we mean the formula which emerges from F by setting the literal l to true the following way: First, substitute all clauses containing l by $\{\mathcal{T}\}$, then delete all occurrences of \bar{l} from any clause and finally delete all empty clauses from F . Notice that empty clauses cannot be satisfied. $F[\bar{l}]$ is defined analogously: we set l to false. Let us illustrate these definitions with a small example.

Let $F = \{x_1, \bar{x}_3\}, \{\bar{x}_1, x_2\}, \{\bar{x}_3, \bar{x}_2\}, \{x_2, \bar{x}_4\}, \{x_1, x_4\}, \{\bar{x}_1\}$ then
 $F[x_1] = \{\mathcal{T}\}, \{x_2\}, \{\bar{x}_3, \bar{x}_2\}, \{x_2, \bar{x}_4\}\{\mathcal{T}\}$ and
 $F[\bar{x}_1] = \{x_3\}, \{\mathcal{T}\}, \{\bar{x}_3, \bar{x}_2\}, \{x_2, \bar{x}_4\}, \{x_4\}\{\mathcal{T}\}$

The expression $F[l_1, \dots, l_k]$ where the l_i 's are literals is defined recursively:
 $F[l_1, \dots, l_k] = F[l_1][l_2, \dots, l_k]$.

1.3. Estimating running times

Run-time estimates of exponential-time algorithms (as being typical for search tree algorithms) have only relatively recently found renewed interest, obviously initiated by the steadily growing interest in parameterized complexity theory and parameterized (and exact exponential-time) algorithms. Yet, the basic knowledge in this area is not very new. This is possibly best exemplified by the textbook of Mehlhorn [116]. There, to our knowledge, the very first parameterized algorithm for the vertex cover problem was given, together with its analysis, much predating the advent of parameterized algorithmics. This algorithm is quite simple: if any edge $e = \{v_1, v_2\}$ remains in the graph, produce two branches in the search tree, one putting v_1 into the (partial) cover and the other one putting v_2 into the cover. In both cases, the parameter k upper bounding the cover size is decremented, and we consider $G[V \setminus \{v_i\}]$ instead of G in the recursive calls. Given a graph G together with an upper bound k on the cover size, such a search tree algorithm produces a binary search tree of height at most k ; so in the worst case, its size (the number of leaves) is at most 2^k . Namely, if $T(k)$ denotes the number of leaves in a search tree of height k , the described recursion implies $T(k) \leq 2T(k-1)$, which (together with the anchor $T(0) = 1$) yields $T(k) \leq 2^k$.

This reasoning generalizes when we obtain recurrences of the form:

$$T(k) \leq \alpha_1 T(k-1) + \alpha_2 T(k-2) + \dots + \alpha_\ell T(k-\ell) \quad (1.1)$$

Algorithm 1 Simple time analysis for search tree algorithms, called ST-simple

Input(s): a list $\alpha_1, \dots, \alpha_\ell$ of nonnegative integers, the coefficients of inequality (1.1)

Output(s): a tight estimate c^k upper bounding $\mathcal{O}^*(T(k))$

1: Consider inequality (1.1) as equation:

$$T(k) = \alpha_1 T(k-1) + \alpha_2 T(k-2) + \dots + \alpha_\ell T(k-\ell)$$

2: Replace $T(k-j)$ by x^{k-j} , where x is still an unknown to be determined.

3: Divide the equation by $x^{k-\ell}$ {This leaves a polynomial $p(x)$ of degree ℓ }.

4: Determine the largest positive real zero (i.e., root) c of $p(x)$.

5: return c^k .

for the size $T(k)$ of the search tree (which can be measured in terms of the number of leaves of the search tree, since that number basically determines the running time of a search tree based algorithm).

More specifically, α_i is a natural number that indicates that in α_i of the $\sum_j \alpha_j$ overall branches of the algorithm, the parameter value k got decreased by i . Notice that, whenever $\ell = 1$, it is quite easy to find an estimate for $T(k)$, namely α_1^k . A recipe for the more general case is contained in Alg. 1. Why does that algorithm work correctly? Please observe that in the simplest case (when $\ell = 1$), the algorithm does what could be expected. We only mention here that

$$p(x) = x^\ell - \alpha_1 x^{\ell-1} - \dots - \alpha_\ell x^0$$

is also sometimes called the *characteristic polynomial* of the recurrence given by Eq. 1.1, and the base c of the exponential function that Algorithm 1 returns is called the *branching number* of this recurrence. Due to the structure of the characteristic polynomial, c is the dominant positive real root.

Alternatively, such a recursion can be also written in the form

$$T(k) \leq T(k-a_1) + T(k-a_2) + \dots + T(k-a_r). \quad (1.2)$$

Then, (a_1, \dots, a_r) is also called the *branching vector* of the recurrence. A (a_1, \dots, a_r) -branch describes a recursive algorithm which generates recursively r problems where the complexity measure of the i -th problem has been reduced by an amount a_i , i.e., the algorithm induces a recursion of the form as in equation 1.2.

As detailed in [82, pp. 326ff.], a general solution of an equation

$$T(k) = \alpha_1 T(k-1) + \alpha_2 T(k-2) + \dots + \alpha_\ell T(k-\ell)$$

(with suitable initial conditions) takes the form

$$T(k) = f_1(k)\rho_1^k + \dots + f_\ell(k)\rho_\ell^k,$$

where the ρ_i are the distinct roots of the characteristic polynomial of that recurrence, and the f_i are polynomials (whose degree corresponds to the multiplicity of the roots

(minus one)). As regards asymptotics, we can conclude $T(k) \in \mathcal{O}^*(\rho_1^k)$, where ρ_1 is the dominant root.

The exact mathematical reasons can be found in the theory of polynomial roots, as detailed in [82, 61, 83, 106, 107]. It is of course also possible to check the validity of the approach by showing that $T(k) \leq \rho^k$ for the obtained solution ρ by a simple mathematical induction argument.

Due to case distinctions that will play a key role for designing refined search tree algorithms, the recurrences often take the form

$$T(k) \leq \max\{f_1(k), \dots, f_r(k)\},$$

where each of the $f_i(k)$ is of the form

$$f_i(k) = \alpha_{i,1}T(k-1) + \alpha_{i,2}T(k-2) + \dots + \alpha_{i,\ell}T(k-\ell).$$

Such a recurrence can be solved by r invocations of Alg. 1, each time solving $T(k) \leq f_i(k)$. This way, we get r upper bounds $T(k) \leq c_i^k$. Choosing $c = \max\{c_1, \dots, c_r\}$ is then a suitable upper bound.

Eq. (1.1) somehow suggests that the entities a_j that are subtracted from k in the terms $T(k - a_j)$ in Eq. (1.2) are natural numbers. However, this need not be the case, even in the case that the branching process itself suggests this, e.g., taking vertices into the cover to be constructed. How do such situations arise? Possibly, during the branching process we produce situations that appear to be more favorable than the current situation. Hence, we could argue that we take a certain credit on this future situation, this way balancing the current (bad) situation with the future (better) one. Interestingly, this approach immediately leads to another optimization problem: How to choose the mentioned credits to get a good estimate on the search tree size? We will describe this issue in more detail in a separate section. This sort of generalization is the backbone of the search tree analysis in exact exponential algorithms, where the aim is, say in graph algorithms, to develop non-trivial algorithms for hard combinatorial graph problems with run-times estimated in terms of n (number of vertices) or sometimes m (number of edges). One typical scenario where this approach works is a situation where the problem allows for nice branches as long as large-degree vertices are contained in the graph, as well as for nice branches if all vertices have small degree, assuming that branching recursively generates new instances with degrees smaller than before, see [67, 73, 134, 151]

1.4. Reference Search Trees

We will formalize a search scheme for combinatorial optimization problems. These problems can usually be modeled as follows. We are given a triple $(\mathcal{U}, \mathcal{S}, c)$ such that $\mathcal{U} = \{u_1, \dots, u_n\}$ is called the *universe*, $\mathcal{S} \subseteq \mathcal{P}(\mathcal{U}) := \{M \mid M \subseteq U\}$ is the *solution space* and $c : \mathcal{P}(\mathcal{U}) \rightarrow \mathbb{N}$ is the *value function*. Generally we are looking for a $S \in \mathcal{S}$ such that $c(S)$ is minimum or maximum. We then speak of a combinatorial minimization

(maximization, resp.) problem. The general search space is $\mathcal{P}(\mathcal{U})$.

The *set vector* (sv_Q) of a set $Q \in \mathcal{P}(\mathcal{U})$ is a 0/1-vector indexed by the elements of \mathcal{U} such that: $sv_Q[i] = 1 \iff u_i \in Q$. We write $sv_Q \in \mathcal{S}$ when we mean $Q \in \mathcal{S}$. A *solvec* is a 0/1/ \star -vector.

We define the following partial order \preceq on solvecs s_1, s_2 of length n :

$$\begin{aligned} s_1 \preceq s_2 \iff & \forall 1 \leq i \leq n : (s_1[i] = \star \Rightarrow s_2[i] = \star) \\ & \wedge (s_1[i] = 0) \Rightarrow (s_2[i] \in \{0, \star\}) \\ & \wedge (s_1[i] = 1) \Rightarrow (s_2[i] \in \{1, \star\}) \end{aligned}$$

If $s_1 \preceq s_2$ then s_2 can be transformed to s_1 by replacing an entry $s_2[i] = \star$ by $s_1[i]$. In the following we think of a solvec s_2 as a partial solution of a combinatorial problem. The fact $s_1 \preceq s_2$ implies that s_2 can be extended to s_1 , i.e. $\{i \mid s_2[i] = 1\} \subseteq \{i \mid s_1[i] = 1\}$, $\{i \mid s_2[i] = 0\} \subseteq \{i \mid s_1[i] = 0\}$ and $\{i \mid s_2[i] = \star\} \supseteq \{i \mid s_1[i] = \star\}$.

A *out-tree* is a directed tree $D(V, T)$ with root $r \in V$ such that all arcs are directed from the father-vertex to the child-vertex. For a vertex $u \in V$ the term ST_u refers to the sub-tree rooted at u .

Definition 1.4.1. A *reference search tree (rst)* for a combinatorial minimization (maximization, resp.) problem $(\mathcal{U}, \mathcal{S}, c)$ is a directed graph $D(V, T \cup R \cup L)$ together with an injective function $label : V \rightarrow \{(z_1, \dots, z_n) \mid z_i \in \{0, 1, \star\}\}$ with the following properties:

1. $D(V, T)$ is an out-tree.
2. $D(V, T \cup R \cup L)$ is acyclic.
3. Let $u, v \in V(D)$ then u is a descendant of v in $D(V, T)$ iff $label(u) \preceq label(v)$.
4. For any set vector sv_Q of a set $Q \in \mathcal{P}(\mathcal{U})$ with $Q \in \mathcal{S}$ and a vertex $v \in V(D)$ such that $sv_Q \preceq label(v)$ we have either one of the following properties:
 - a) There exists a vertex $x \in V(ST_v)$ such that there is an arc $(x, y) \in L$, $(y, x) \notin L$ and we have that there is a 0/1-vector h with $h \preceq label(y)$, $c(h) \leq c(sv_Q)$ ($c(h) \geq c(sv_Q)$, resp.), $h \in \mathcal{S}$ and $y \in V(ST_v)$.
 - b) There exists a vertex $x \in V(ST_v)$ such that there is an arc $(x, y) \in R$ and we have that there is a 0/1-vector h with $h \preceq label(y)$, $c(h) \leq c(sv_Q)$ ($c(h) \geq c(sv_Q)$, resp.) $h \in \mathcal{S}$ and $y \notin V(ST_v)$.

In item 4 of Definition 1.4.1 we suppose that the label of v represents a partial solution. Now if $sv_Q \preceq label(v)$ this means that $label(v)$ can be extended to the set vector sv_Q . It is possible that such a set vector does not have to be considered as a solution.

How can a rst be exploited algorithmically? It is important to see that in a rst all the information for finding an optimal solution is included. In a branching algorithm we often skip a solution s due to local exchange arguments. So, if $s \preceq label(u)$ for a sub-tree ST_u we skip s if we can find a solution in ST_u which is no worse. This fact is reflected by item 4a. Here a solution $sv_Q \preceq label(v)$ is skipped if a local reference

(x, y) exists, i.e., $y \in V(ST_v)$. Thus, we find the alternative solution in the same subtree attached to $v \in V(D)$. These kind of local references are implicitly made by any branching algorithm beating the 2^n -barrier. Observe that $sv_Q \preceq label(v)$, $h \preceq label(v)$ and $h \preceq label(y)$ where h is the alternative solution. Thus, the 0/1-entries in $label(v)$ are also contained in sv_Q and h .

Nevertheless, we try to extend this current ad-hoc notion of search trees: In a reference search tree we also have the possibility to make a reference to another subtree ST_f with $label(f) \not\preceq label(v)$ where such a solution could be found, see Figure 1.1 for an illustration. In ST_f it might also be the case that we have to follow a reference once more. So, the only obstacle seems to be that, if we follow reference after reference, we end up in a cycle. But this is prevented by item 2. of Definition 1.4.1. An algorithm building up an rst can eventually benefit by cutting of branches and introducing references instead. In the rest of this document we use the following convention: a *local reference* is an arc $(x, y) \in L$ and a *global reference* an arc $(x, y) \in R$. The word *reference* refers to both kinds.

The Nodes of the Reference Search Tree The nodes of the rst $V(D)$ represent choices made concerning the, say, blank objects in the input. These could be vertices in some graph for which we have not decided whether they are part of the future solution or not. So, each node of $V(D)$ represents exactly one choice for such an object. These choices can be due to branching or to applying reduction rules. Hence there is a 1-to-1 correspondence between $V(D)$ and the application of reduction rules and branchings. According to this we will speak of *full nodes* and *flat nodes*, i.e. full nodes have two children in $D(V, T)$, flat nodes only one. In particular, full nodes correspond to a branching operation and flat nodes to the application of a reduction rule. Observe that nodes where reference pointers start are flat. If we encounter an object v where a choice has been already made in the current node q of the search tree we can find a second node $d_v \in V(D)$ which represents the choice made on v , i.e., $label(d_v)$ contains a 0/1-entry at the position of v whereas for the father of d this is not true. That is we must have that $label(q) \preceq label(d_v)$. We can find d_v by simply going up the search tree starting from q . We sometimes indicate this relation by writing d_v^q , whereas we omit the superscript where it is clear from the context. By $p(d_v^q)$ we denote the parent of d_v^q in the reference search tree $D(V, T)$, i.e., the only entering neighbor of d_v^q .

Drawing the Reference Search Tree. The correctness proofs proceed to some extent in a graphical way. For this we draw the out-tree (the search tree without references) $D(V, T)$ in the plane with x - and y -coordinates. If u is a point in the plane then $pos_x(u)$ denotes its x - and $pos_y(u)$ its y -coordinate.

Definition 1.4.2. It is possible to draw an out-tree $D(V, T)$ satisfying three properties:

1. Firstly, if $v \in V(D)$ is a father of $u \in V(D)$ then $pos_y(v) > pos_y(u)$.
2. Secondly, let $v \in V(D)$ have two children $u_v, u_{\bar{v}}$, i.e., it is a full node. The child $u_{\bar{v}}$ corresponds to the branch where we decided that some object v is excluded, in

u_v we decided to include v in to the future solution. We want D to be drawn such that for all $z \in ST_{u_{\bar{v}}}$ we have $pos_x(v) > pos_x(z)$ and for all $z' \in ST_{u_v}$ we have $pos_x(z') > pos_x(v)$. Hence we may speak of $u_{\bar{v}}$ as the left and u_v as the right child of v . According to this we will refer to them as $l(v)$ and $r(v)$, respectively.

3. Thirdly, let $v \in V(D)$ be a flat node with its only child v_c . Then we require that $pos_x(v) = pos_x(v_c)$.

A drawing of an out-tree in the plane satisfying these properties will be called *proper*.

The idea behind a proper drawing of a reference tree is that if we are inserting global references, say, strictly pointing from the left to the right then the overall structure remains acyclic.

Lemma 1.4.1: Let $D(V, T \cup R \cup L)$ be a directed graph such that $D(V, T)$ is an out-tree. If all arcs in R are pointing from the left to right (right to left, resp.) with respect to x -coordinate and for all arcs $(x, z) \in L$ we have $pos_y(x) \geq pos_y(z)$ in the proper drawing of $D(V, T)$ then $D(V, T \cup R \cup L)$ is acyclic.

Proof. Assume, w.l.o.g, all arcs in R point from the left to the right. Firstly, suppose there is a cycle $C \subset T \cup R$ in $D(V, T \cup R \cup L)$. The existence of such a cycle C implies that there is at least one $r \in R$ which must point from the right to the left, a contradiction. Secondly, suppose there is a cycle $C \subset T \cup L$ in $D(V, T \cup R \cup L)$. If there is at least one $(a, b) \in L \cap C$ with $pos_y(a) > pos_y(b)$ the existence of such a cycle would imply an arc $(x, z) \in L \cap C$ with $pos_y(x) < pos_y(z)$, a contradiction. If for all $(a, b) \in L \cap C$ we have $pos_y(a) = pos_y(b)$ we must have a cycle of length two. Thus, there are arcs $(a, b), (b, a) \in L$ violating Definition 1.4.1.4a.

Thirdly, suppose there is a cycle $C \subset T \cup R \cup L$ in $D(V, T \cup R \cup L)$ such that $C \cap R \neq \emptyset$ and $C \cap L \neq \emptyset$. Let $C = (c_1, c_2)(c_2, c_3) \dots (c_{\ell-1}, c_{\ell})$ and, w.l.o.g, $(c_1, c_2) \in R$. C can be partitioned the following way: $C = S_1 S_2 \dots S_t$ such that

- a) for $i \equiv 1 \pmod{2}$ we have $S_i \subseteq R$
- b) for $i \equiv 0 \pmod{2}$ we have $S_i \subseteq T \cup L$.

Let $S_i = (s_1^i, s_2^i)(s_2^i, s_3^i) \dots (s_{|S_i|-1}^i, s_{|S_i|}^i)$. Consider $S_2 \subseteq T \cup L$ and let $(a, b) \in S_1$ and $(p, q) \in S_2$. Then we have that $pos_x(a) < pos_x(p)$ and $pos_x(a) < pos_x(q)$ due to item 2 in Definition 1.4.2. By an inductive argument it can be proven that for all $(p, q) \in S_t$ and for all $1 \leq j < t$ with $S_j \in R$ and $(a, b) \in S_j$ we have $pos_x(a) < pos_x(p)$ and $pos_x(a) < pos_x(q)$. Especially, we can deduce that $pos_x(c_1) < pos_x(s_{|S_t|}^t)$. On the other hand note that $s_{|S_t|}^t = c_{\ell}$ and due to C being a cycle $c_1 = c_{\ell}$. Thus, we also have $pos_x(c_1) = pos_x(s_{|S_t|}^t)$, a contradiction. \square

Thus, item 2 of Definition 1.4.1 is easily proven if we can show that some algorithm is inserting global references pointing in only one horizontal direction and local references pointing *downwards*, i.e., for all arcs $(x, z) \in L$ we have $pos_y(x) \geq pos_y(z)$ in the proper drawing. Also we must show that: $(a, b) \in L \Rightarrow (b, a) \notin L$. Local references point *strictly downwards* if for all arcs $(x, z) \in L$ we have $pos_y(x) > pos_y(z)$ in the proper drawing.

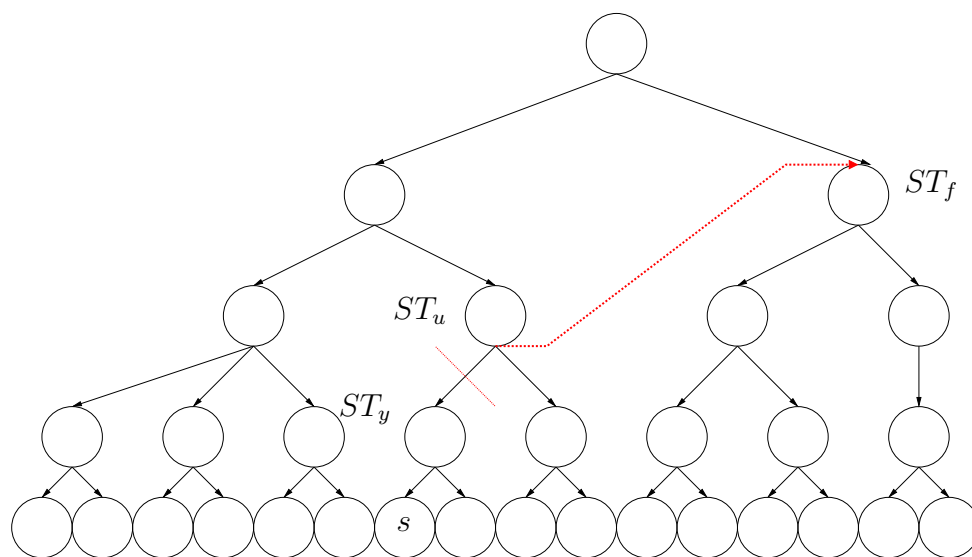


Figure 1.1.: The left child y of the subtree rooted at u (called ST_u) is deleted from the search space. Instead we add a reference (u, f) to the root of ST_f . Here we have to assure that every solution in the sub tree ST_y is no better than at least one solution in ST_f .

Chapter 2.

Measure&Conquer

The search tree method is one of the most applied techniques in the design of exponential-time algorithms. They play an important role in parameterized and non-parameterized exponential-time algorithms. It can be seen as an intelligent way of traversing the solution space of a combinatorial problem. The efficiency of such algorithms is in direct relation to the size of the traversed solution space. Generally, we try to argue that certain parts of the solution space can be neglected as we will find equivalent solutions in the considered parts. We now explain this concept by the following case study:

2.1. Case Study: A Search Tree Algorithm for Vertex Cover

We are considering the following problem

VERTEX COVER

Given: $G(V, E)$.

Task: Find $V' \subseteq V$ such that for all $e \in E$ we have $V' \cap e \neq \emptyset$ and $|V'|$ is minimum.

A very simple and intuitive algorithm is Algorithm 2, which has already been described in an intuitive manner in section 1.3:

Algorithm 2 $\text{SimpVC}(G(V, E), Sol)$

```
1: if  $\exists e = \{u, v\} \in E$  then
2:    $S_1 = \text{SimpVC}(G[V \setminus \{u\}], Sol \cup \{u\})$ .
3:    $S_2 = \text{SimpVC}(G[V \setminus \{v\}], Sol \cup \{v\})$ .
4:   if  $|S_1| < |S_2|$  then
5:     return  $S_1$ 
6:   else
7:     return  $S_2$ 
8:   end if
9: else
10:  return  $Sol$ 
11: end if
```

If we invoke the procedure in the manner $\text{SimpVC}(G(V, E), \emptyset)$ it will return a minimum vertex cover. This algorithm uses the fact that every edge $e = \{u, v\}$ must be

covered by u or v . This means that if $Sol \subseteq V$ is a solution to the problem then either $u \in Sol$ or $v \in Sol$. This fact is used in steps 5 and 6. We solve the problem by creating two new problems of smaller size as either u or v are deleted from the input graph. We call this a recursive algorithm. Thus, the maximum recursion depth is n and the number of created subproblems is 2^n . These subproblems can be viewed as the leaves of a tree, where the internal vertices correspond to the non-terminating calls of `SimpVC`. We call this search structure a *search tree*. We will show that with only some more effort we can achieve a search tree size of only $\mathcal{O}(1.3803^n)$. We introduce the notion of a reduction rule.

Definition 2.1.1. A *reduction rule* is a polynomial-time algorithm, which, given an problem instance I , returns a second instance I' such that

1. $size(I) \geq size(I')$.
2. From any optimal solution S' for I' an optimal solution S for I can be constructed in polynomial time .

As an example we state the following reduction rule:

Deg1: Let $e = \{u, v\} \in E$ such that $d(u) = 1$. Then delete u and the edges incident to v

The correctness of this rule can be proven the following way. Let I' be the instance created by **Deg1** and S' be any optimal solution for I' . Let $S = S' \cup \{v\}$. Then S is a vertex cover. We claim that S is an optimal solution for the original instance I . Let S^* be any optimal solution for I with $|S^*| < |S|$. W.l.o.g, we can assume that $v \in S^*$ as otherwise $u \in S^*$. Then simply exchange v and u . Let $H := S^* \setminus \{u\}$ then H is a vertex cover for I' . As $|H| < |S'|$ this contradicts the optimality of S' .

This is a rather theoretical point of view. The above rule can also be explained in a more algorithmic fashion. Whenever there is an optimal solution S for I with $u \in S$ then $\tilde{S} := S \setminus \{u\} \cup \{v\}$ is also an optimal solution. This due to the fact that u only covers $\{u, v\}$, which v also covers (and possibly even more edges). Coming from this algorithmic point of view one can also state the reduction rule this way.

Deg1': Let $e = \{u, v\} \in E$ such that $d(u) = 1$. Then adjoin v to the solution, delete u and the edges incident to v

This means that by **Deg1'** we fix some vertex to be in the optimal solution. The reduction rules throughout this dissertation will be presented in this algorithmic manner. Nevertheless, they will obey Definition 2.1.1.

Let us now consider Algorithm 3.

In Algorithm 3 we recognize the typical structure of a recursive algorithm. Firstly, upon entering the procedure a set of reduction rules will be carried out exhaustively. We find this part in the first line. Then there is a check if the instance is already empty or, in some cases, if the instance is solvable in polynomial time. This relates to line two. Afterwards the algorithm considers a finite number of recursions. This can be

Algorithm 3 $\text{SimpVC2}(G(V, E), \text{Sol})$

```

1: Apply Deg1' exhaustively.
2: if  $\exists e = \{u, v\} \in E$  with  $d(u) \geq 2$  then
3:    $S_1 = \text{SimpVC2}(G[V \setminus \{u\}], \text{Sol} \cup \{u\})$ .
4:    $S_2 = \text{SimpVC2}(G[V \setminus \{N[u]\}], \text{Sol} \cup N(u))$ .
5:   if  $|S_1| < |S_2|$  then
6:     return  $S_1$ 
7:   else
8:     return  $S_2$ 
9:   end if
10: else
11:   return  $\text{Sol}$ .
12: end if

```

observed in lines three and four. Algorithms which obey this structure are often called *Branch&Reduce Algorithms*.

We now come to the correctness of Algorithm 3. We above already showed that **Deg1'** applied to any instance is correct. If there is no edge then we are done and we can return the constructed solution Sol . If there is a edge $\{u, v\}$ such that $d(u) = d(v) = 1$ this would contradict the fact that **Deg1'** has been applied exhaustively. Therefore, the remaining interesting case is the one which applies in the **if**-statement in line two. The two recursive calls are based on the following fact. If u is not in the optimal solution then every neighbor must be contained there. Otherwise an edge remains uncovered. As we are looking for a minimum solution the procedure returns the smaller solution obtained from the recursive calls (i.e. the smaller set out of $\{S_1, S_2\}$).

We now come to the complexity. The reduction rule phase only consists in applying **Deg1'**. This can be done in $\mathcal{O}(n)$ steps by simply checking whether each vertex has one or more than one neighbor. Now the two recursive calls are the part where exponential growth in the running time comes from. In the case u is adjoined to the solution n is decreased by one. In the second call where $N(u)$ is adjoined n decreases by three. This due to the fact that $d(u) \geq 2$. Hence, this implies a branching vector of the form $(1, 3)$ whose characteristical polynom has the as the biggest positive root 1.4656. Hence, the number of generated subproblems is $\mathcal{O}(1.4656^n)$.

The Polynomial Time Solvable Case

A further run time improvement comes from identifying a graph class where VERTEX COVER is polynomial time solvable. We consider graphs with maximum degree two. Clearly, this class consist exactly of simple paths and cycles of length ℓ , which we abbreviate by $P_\ell = p_1 \dots p_\ell$ and $C_\ell = c_1 \dots c_\ell$. For this class it is straightforward to see that the sets $(\bigcup_{0 \leq i \leq \lfloor \ell/3 \rfloor - 1} p_{2+3 \cdot i}) \cup \{p_{\ell-1}\}$ and $(\bigcup_{0 \leq i \leq \lfloor \ell/3 \rfloor - 1} c_{2+3 \cdot i}) \cup \{c_{\ell-1}\}$, respectively, are solutions. Basically, starting from the second vertex we take every third vertex into the solution. With this knowledge we realize that Algorithm 4 is correct.

Algorithm 4 $\text{SimpVC3}(G(V, E), \text{Sol})$

```

1: Apply Deg1' exhaustively.
2: if  $\forall v \in V : d(v) \leq 2$  then
3:   Solve the instance in polynomial time.
4: else
5:   if  $\exists e = \{u, v\} \in E$  with  $d(u) \geq 3$  then
6:      $S_1 = \text{SimpVC3}(G[V \setminus \{u\}], \text{Sol} \cup \{u\})$ .
7:      $S_2 = \text{SimpVC3}(G[V \setminus \{N[u]\}], \text{Sol} \cup N(u))$ .
8:     if  $|S_1| < |S_2|$  then
9:       return  $S_1$ 
10:    else
11:      return  $S_2$ 
12:    end if
13:  else
14:    return  $\text{Sol}$ .
15:  end if
16: end if

```

The run time of Algorithm 4 improves as we can assume that $d(u) \geq 3$ in the recursive calls. By the discussion of this part with respect to Algorithm 3 we see that this time a $(1, 4)$ branching vector can be exhibited. Therefore, the number of generated subproblems can be bounded from above by $\mathcal{O}(1.3803^n)$. We also mention that by reduction rule **Deg1**' the graphs appearing in step 3 of Algorithm 4 are 2-regular, i.e., the graph consists of cycle components.

We have seen that with some moderate effort it is possible to design an algorithm which breaks to trivial bound of 2^n (which simply enumerates all vertex subsets). With refined case distinctions, more sophisticated branching rules and a *Measure&Conquer*-Analysis the run time of the search tree algorithm has been improved to $\mathcal{O}^*(2^{0.287n}) \subseteq \mathcal{O}^*(1.2202^n)$ by F.V. Fomin, F. Grandoni and D. Kratsch [74].

2.2. Case Study: A Search Tree Algorithm for Dominating Set

We come to a second problem:

DOMINATING SET

Given: $G(V, E)$.

Task: Find a minimum subset $D \subseteq V$ such that for all $u \in V$ we have $N[u] \cap D \neq \emptyset$.

Alternatively, we also could say that every vertex $u \notin D$ must have a neighbor in D . A set D with this property is called a *dominating set*.

Now we could think of applying the same branching strategy as in the case of VERTEX COVER. So we might pick a vertex $v \in V$ and argue that either v must be in the

dominating set we look for or one of its neighbors. As the degree of v could be very large, say $d(v) = 3$, this implies a branching vector of the form $(1, \dots, 1)$ of length $d(x) + 1$, i.e., $(1, 1, 1, 1)$ in our case. Thus, the run time upper bound is far from being less than $\mathcal{O}^*(2^n)$. Here we have to take another approach: *Measure&Conquer* (see [74]). In this approach, the complexity of an algorithm is not analyzed with respect to $n = |V|$ (or $m = |E|$) for a graph instance $G = (V, E)$. Rather, one chooses a tailored measure, call it μ , which should reflect the progress of the algorithm. Nevertheless, in the end we desire an upper bound of the form c^n . Hence, we must assure that there is some constant ℓ such that $\mu \leq \ell n$ during the whole algorithm. Then, a proven upper bound c^μ entails the desired upper bound $c^{\ell n}$.

A simple algorithm

We will focus on the following annotated variant of DOMINATING SET.

ANNOTATED DOMINATING SET

Given: $G(V, E)$ and two disjunctive subsets $A, I \subseteq V$.

Task: Find a minimum set $D \subseteq V \setminus I$ such that for all $u \in V$ we have $N[u] \cap (D \cup A) \neq \emptyset$.

The sets A and I can be interpreted in the following way. The set A consists of vertices, which are predetermined to be part of the future dominating set. The set I is composed of vertices, which do not appear there. We might think of the vertices in A as *active* and the ones in I as *inactive*. We also see that each instance of DOMINATING SET is also an instance of ANNOTATED DOMINATING SET. Here the sets A and I are empty. We give an algorithm for ANNOTATED DOMINATING SET using less than 2^n steps.

It branches on vertices by deciding whether they should be in the solution (*active*) or not (*inactive*). Regarding this we call them *active* and *inactive*. A vertex for which this decision has not been made is called *blank*. If a vertex is active, then its neighbors and itself are *dominated*. Let $\text{DO} := \{v \in V \mid v \text{ is dominated}\}$, $\text{ND} := V \setminus \text{DO}$, $\text{BLND} = \{v \in V \mid v \text{ is blank \& not dominated}\}$, $\text{INND} = \{v \in V \mid v \text{ is inactive \& not dominated}\}$, $\text{INDO} = \{v \in V \mid v \text{ is inactive \& dominated}\}$ and $\text{BLDO} = \{v \in V \mid v \text{ is blank \& dominated}\}$.

We first give a set of reduction rules:

RR1: If there is $u \in \text{INND}$ such that $N(u) \cap (\text{BLDO} \cup \text{BLND}) = \{q\}$ then set q active.

RR2: Let $u \in \text{BLDO}$ such that $|N(u) \cap \text{ND}| \leq 1$ then set u inactive

RR3: Let $u \in \text{BLND}$ such that all $v \in N(u)$ are inactive, then u becomes active

RR4: Let $u \in \text{BLND}$ such that all $v \in N(u)$ are dominated. Then u becomes inactive

If we assume that **RRi** is always carried out before **RR(i+1)** the next proposition holds.

Proposition 2.2.1: The reduction rules are sound and after their exhaustive application we have the next two properties:

1. For all $u \in \text{BLDO}$ we have $|N(u) \cap \text{ND}| \geq 2$.
2. For all $u \in \text{BLND}$ we have that $|N(u) \cap \text{ND}| \geq 1$.

Algorithm 5 A simple algorithm for DOMINATING SET

-
- 1: Apply the reduction rules exhaustively.
 - 2: **if** possible choose a $v \in \text{BLND}$ such that $|N(v) \cap (\text{BLND} \cup \text{INND})| \geq 2$. **then**
 - 3: Binary branch on v (i.e., set v active in one branch, inactive in the other)
 - 4: **else if** possible choose a $v \in \text{BLDO}$ such that $|N(v) \cap (\text{BLND} \cup \text{INND})| \geq 3$. **then**
 - 5: Binary branch on v .
 - 6: **else**
 - 7: Solve the remaining instance in polynomial time using an EDGE COVER algorithm.
 - 8: **end if**
-

Proof. We begin with the soundness of the rules. In **RR1** the vertex q is the only blank neighbor of u . Hence, q is the only vertex which can dominate u and thus it must be active. We turn to **RR2**. If $|N(u) \cap \text{ND}| = 0$ then all vertices in $N[u]$ are dominated. There is no need for u being active. If $N(u) \cap \text{ND} = \{q\}$ then observe that there must be some $t \in N[q]$ which is blank and $t \neq u$ (\star). If q is blank then one can choose $t = q$. If q is inactive then **RR1** assures (\star). But then any solution where u is active can be transformed to an equal sized solution where u is inactive. Simply set t active and u inactive. In **RR3** no vertex different from u can dominate it and thus u must be set active. In **RR4** u will only dominate itself. Due to the higher priority of **RR3** there must be some $t \in N(u) \cap \text{BLDO}$. If u is active in some solution S then we get an equivalent solution S' by setting t active and u inactive.

Property one is assured by **RR2** and property two by **RR4**. □

Now Algorithm 5 which solves ANNOTATED DOMINATING SET is introduced.

A Polynomial Time solvable Case

In step 7 of Algorithm 5 we claim that the remaining instance can be transformed into an EDGE COVER-instance. The task in EDGE COVER is to identify a set $\tilde{E} \subseteq E$ minimum in cardinality such that $V = V(\tilde{E})$. This problem can be solved by matching techniques: Firstly, compute in $\mathcal{O}(n^3)$ a maximum matching $M \subseteq E$. Let $E' := M$. Secondly, if there is a $u \in V$ with $u \notin V(M)$ then simply adjoin one of its incident edges to E' . When this procedure ends E' is the desired edge cover, see J. Plesník [128] for details.

We first recognize that in step seven every vertex $u \in \text{BLND} \cup \text{BLDO}$ can dominate at most two vertices possibly including itself. Thus, $|N[u] \cap \text{ND}| \leq 2$. On the other hand due to proposition 2.2.1 we must have $|N[u] \cap \text{ND}| \geq 2$ and therefore $|N[u] \cap \text{ND}| = 2$. Now create an EDGE COVER instance $G_{EC} = (V(E_{EC}), E_{EC})$ the following way:

For all $v \in \text{BLND}$ with $N(v) \cap (\text{BLND} \cup \text{INND}) = \{q\}$, adjoin $e = \{v, q\}$ to E_{EC} and let $\alpha(e) = v$; for all $v \in \text{BLDO}$ with $N(v) \cap (\text{BLND} \cup \text{INND}) = \{x, y\}$, note that $x \neq y$, put $e = \{x, y\}$ into E_{EC} and (re-)define $\alpha(e) = v$.

If C is a minimum edge cover of G_{EC} we can extract a solution for the original problem: set all v active where $v = \alpha(e)$ for some $e \in C$.

The Exponential Part

Now we come to the analysis of the branching process in steps 3 and 5. We now define our measure:

$$\mu = |\text{BLND}| + \omega \cdot (|\text{INND}| + |\text{BLDO}|) \leq n$$

Let $n_{bl} = |N(v) \cap \text{BLND}|$ and $n_{in} = |N(v) \cap \text{INND}|$.

In **step 3** we recurse on two cases by setting v to be active in one and to be inactive in the other.

v becomes active: We first reduce μ by one as v vanishes from μ . Then all vertices in $N(v) \cap \text{BLND}$ will be dominated and hence moved to the set BLDO . Thus, μ is reduced by an amount of $n_{bl} \cdot (1 - \omega)$. In the same way the vertices in $N(v) \cap \text{INND}$ are dominated. Therefore, these do not appear in μ anymore. Hence, μ is lowered by $n_{in}\omega$.

v becomes inactive: We reduce μ by $(1 - \omega)$, as v is moved from BLND to INND .

Hence, the branching vector is:

$$(1 + n_{bl}(1 - \omega) + n_{in}\omega, (1 - \omega)) \tag{2.1}$$

where $n_{bl} + n_{in} \geq 2$ due to step 2.

In **step 4**, we have chosen $v \in \text{BLDO}$ for branching. Here, we must consider that in the first and second branch we only get ω as reduction from v (v disappears from μ). But the analysis with respect to $N(v) \cap (\text{BLND} \cup \text{INND})$ remains valid. Thus,

$$(\omega + n_{bl}(1 - \omega) + n_{in}\omega, \omega) \tag{2.2}$$

is the branching vector with respect to $n_{bl} + n_{in} \geq 3$ due to step 3.

Unfortunately, depending on n_{bl} and n_{in} we have an infinite number of branching vectors. But it is only necessary to consider the worst case branches. For (2.1) these are the ones with $n_{bl} + n_{in} = 2$ and for (2.2) the ones where $n_{bl} + n_{in} = 3$. For any other branching vector, we can find one among those giving a worse upper bound. Thus, we have a finite set of recurrences $R_1(\omega), \dots, R_7(\omega)$ depending on ω . The next task is to choose ω in a way such that the maximum root of the evolving characteristic polynomials is minimum. In this case we easily see that $\omega := 0.5$. Then the worst case branching vector for (2.1) and (2.2) is $(2, 0.5)$. Thus, the number of leaves of the search tree evolving from this branching vector can be bounded by $\mathcal{O}^*(1.9052^\mu)$. Thus, our algorithm breaks the 2^n -barrier using a simple measure.

Refining The Measure One might argue that we should use a more elaborated measure to get a better upper bound:

$$\mu' = |\text{BLND}| + \omega_1 \cdot (|\text{INND}|) + \omega_2 \cdot (|\text{BLDO}|)$$

Under μ' , (2.1) becomes $(1 + n_{bl}(1 - \omega_2) + n_{in}\omega_1, (1 - \omega_1))$; (2.2) becomes $(\omega_2 + n_{bl}(1 - \omega_2) + n_{in}\omega_1, \omega_2)$.

The right choice for the weights turns into a tedious task. In fact, if $\omega_2 = 0.637$ and $\omega_1 = 0.363$, then we get an upper bound of $\mathcal{O}^*(1.8899^{\mu'})$. Nevertheless, the current best upper bound is $\mathcal{O}(1.5048^n)$, see [153].

Generally, one wants the measure to reflect the progress made by the algorithm best possible. This leads to more and more complicated measures with lots of weights to be chosen. So at a certain point, this task cannot be done by hand and is an optimization problem of its own. It can only be solved in reasonable time with the help of a computer. We will give some details on this in section 2.6.

The approach of using non-standard measures for estimating the complexity of exact exponential time algorithms is called *Measure&Conquer*. With this technique it was possible to prove run time upper bounds $\mathcal{O}^*(c^n)$ with $c < 2$ for several hard vertex selection problems. Among these problems (where for years nothing better than the trivial 2^n -algorithm was known) are many variants of DOMINATING SET [67] like CONNECTED or POWER DOMINATING SET [73, 134] and FEEDBACK VERTEX SET [64]. The methodology resulted in simplifying algorithms (INDEPENDENT SET [72]) and in speeding up existent non-trivial ones (DOMINATING SET [67, 151], INDEPENDENT DOMINATING SET [79] and MAX-2-SAT [133]). This approach also served for algorithmically proving upper bounds on the number of minimal dominating [70] and feedback vertex sets [64].

2.3. Case Study: Parameterized Measure & Conquer for Nonblocker

It is known that DOMINATING SET parameterized by the size of solution set is $W[1]$ -hard [35]. Thus, it is not likely that there is an \mathcal{FPT} -algorithm for this problem. In lieu thereof we rather consider the so-called dual NONBLOCKER of DOMINATING SET. Consequently we call $N \subseteq V$ a nonblocker set if $V \setminus N$ is a dominating set.

k-NONBLOCKER

Given: A graph $G(V, E)$, and the parameter k .

We ask: Is there a nonblocker set $N \subseteq V$ such that $|N| \geq k$.

In our section 2.3 we will exhibit an algorithm solving this problem in $\mathcal{O}^*(3.07^k)$. Thereby we demonstrate how to transfer the *Measure&Conquer*-approach to parameterized algorithmics.

2.3.1. A Simple Approach

Every dominating set D can be portioned into the sets $D_{\geq 1} := \{v \in D \mid (N(v) \setminus N[D \setminus \{v\}]) \neq \emptyset\}$ and $D_0 := D \setminus D_{\geq 1}$. Let also $NB(D) := V \setminus D$ be the corresponding Nonblocker set.

2.3. Case Study: Parameterized Measure & Conquer for Nonblocker

Lemma 2.3.1: Suppose N is a maximum Nonblocker set. If $D := V \setminus N$ then the following holds:

1. There are no $u, v \in D_0$ such that $\{u, v\} \in E$.
2. There is no $u \in D_0$ and $h \in D_{\geq 1}$ such that $\{u, h\} \in E$.
3. There are no $u, v \in D_0$ and no $h \in N$ such that $\{u, h\}, \{v, h\} \in E$.

Proof. 1. Assume the contrary. Then let $D' = D \setminus \{u\}$. D' is a dominating set such that $v \in D'_{\geq 1}$ and $|D'| = |D| - 1$. Thus $NB(D') > NB(D) = N$, a contradiction to the maximality of N .

2. Let $D' = D \setminus \{u\}$. Then analogously has in the previous item D' is a dominating set such that $NB(D') > NB(D)$, a contradiction.

3. Consider $D' = (D \setminus \{u, v\}) \cup \{h\}$ Then D' is a dominating set with $h \in D'_{\geq 1}$ and $|D'| < |D|$. Hence, again the contradictory implication $NB(D') > NB(D)$ follows. \square

By Lemma 2.3.1 we directly can justify the next local optimization rules (**OR**):

OR1 If there are $u, v \in D_0$ such that $\{u, v\} \in E$ then set $D := D \setminus \{u\}$ and $N := N \cup \{u\}$.

OR2 If there are $u \in D_0$ and $h \in D_{\geq 1}$ such that $\{u, h\} \in E$ then set $D := D \setminus \{u\}$ and $N := N \cup \{u\}$.

OR3 If there are $u, v \in D_0$ and $h \in N$ such that $\{u, h\}, \{v, h\} \in E$ then set $D := (D \setminus \{u, v\}) \cup \{h\}$ and $N := (N \setminus \{h\}) \cup \{u, v\}$.

The Algorithm We first present Algorithm 6:

In the algorithm the vertices in the set I are predetermined not to be dominating and A contains vertices who will be dominating. Observe that in what follows we will use the same notation as in section 2.2, i.e. the active vertices are in D , the inactive in N and $D = A$ as well as $N = I$.

We use the a complexity measure which is inspired by the *Measure&Conquer*-Approach:

$$\kappa(A, I) = k - \omega_{nb} \cdot |I| - (1 - \omega_{nb}) \cdot |A| \text{ where } \omega_{nb} = 0.5$$

It is very important that no step of Algorithm 6 ever increases $\kappa(A, I)$. The next lemma suffices to show this.

Lemma 2.3.2: Step 3 of Algorithm 6 and the optimization rules (**OR**) do not increase $\kappa(A, I)$.

Proof. Note that step 3 only adds vertices from $V \setminus (A \cup I)$ either to A or I . This only decreases $\kappa(A, I)$. As $\omega_{nb} = (1 - \omega_{nb})$ the claim follows for the optimization rules. \square

Algorithm 6**Input:** A connected graph $G(V, E)$ and the parameter k .**Output:** A Nonblocker set $N \subseteq V$ such that $|N| \geq k$ if it exists. $A \leftarrow \emptyset, I \leftarrow \emptyset.$ **SolveNB**(A, I)Procedure: **SolveNB**(A, I)

- 1: If there is $u \in I$ such that $N[u] \subseteq I$ then halt and answer **NO**.
- 2: **if** $\kappa(A, I) \leq 0$ **then**
- 3: Greedily augment A to a dominating set D .
- 4: Determine the sets $D_{\geq 1}$, D_0 and $N := NB(D)$.
- 5: Apply **OR1**, **OR2** and **OR3** exhaustively resulting in the sets D^* and $N^* := NB(D^*)$.
- 6: **return YES**.
- 7: **else**
- 8: Choose a vertex $v \in V \setminus (A \cup I)$.
- 9: Recursively consider **SolveNB**($A \cup \{v\}, I$) and **SolveNB**($A, I \cup \{v\}$).
- 10: **end if**

As we stop the algorithm in step 6 we must be able to determine in polynomial time if a nonblocker set extending I exists.

Lemma 2.3.3: In step 6 of Algorithm 6 $N^* := NB(D^*)$ is a Nonblocker set such that $|N^*| \geq k$.

Proof. Let $u \in D_{\geq 1}^*$ then choose an arbitrary $v \in N(u) \setminus N[D^* \setminus \{v\}]$ and fix v to be the private neighbor of u denoted $pn(u)$, i.e., $pn(u) = v$. Note that this private neighbor must exist by definition. By **OR1** and **OR2** we see that for all $z \in D_0^*$ we have $N(z) \subseteq N$. Also $|N(z)| \geq 1$ as we have no isolated vertices. By **OR3** we have that $N(z) \setminus N[D_0^* \setminus \{z\}] = N(z)$. Thus, choose some $v \in N(z) \setminus N[D_0^* \setminus \{z\}]$ and let $\tilde{pn}(z) = v$, i.e., v is a private neighbor of z with respect to D_0^* . Then the following holds:

$$\begin{aligned}
k &\leq \omega_{nb} \cdot |N| + (1 - \omega_{nb}) \cdot |D| \leq \omega_{nb} \cdot |N^*| + (1 - \omega_{nb}) \cdot |D^*| \\
&= \omega_{nb} \cdot |N^*| + (1 - \omega_{nb}) \cdot |D_{\geq 1}^*| + (1 - \omega_{nb}) \cdot |D_0^*| \\
&\stackrel{(*_1)}{=} \omega_{nb} \cdot |N^*| + (1 - \omega_{nb}) \cdot \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + (1 - \omega_{nb}) \cdot \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| \\
&\stackrel{(*_2)}{=} \omega_{nb} \cdot \left| N^* \setminus \left(\bigcup_{u \in D_{\geq 1}^*} pn(u) \cup \bigcup_{u \in D_0^*} \tilde{pn}(u) \right) \right| + \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| \\
&\leq \left| N^* \setminus \left(\bigcup_{u \in D_{\geq 1}^*} pn(u) \cup \bigcup_{u \in D_0^*} \tilde{pn}(u) \right) \right| + \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| = |N^*|
\end{aligned}$$

2.3. Case Study: Parameterized Measure & Conquer for Nonblocker

Note that in $(*_1)$ and $(*_2)$ we used the fact that by $priv_{D^*}(x) := \begin{cases} pn(x) : x \in D_{\geq 1}^* \\ \tilde{pn}(x) : x \in D_0^* \end{cases}$ an injective function is defined. \square

The run time of algorithm 6 can easily be seen to be $\mathcal{O}^*(4^{\kappa(\emptyset, \emptyset)})$ as the algorithm entails a branching vector of the form $(0.5, 0.5)$.

The question is if from this a run time of $\mathcal{O}^*(4^k)$ follows. We provide the following discussion:

We followed a more general approach of how to measure the running time in a parameter k than in the, say, traditional way where during the branching process usually recursive calls will be made with parameters $k' < k$ where k' is an integer. Our point of view is that we are given an initial budget k . During the execution of the algorithm this budget will be decremented due to obtained structural information. This structural information does not necessarily refer to the case that some objects are fixed to be in the future solution. It can comprise much more (i.e degree-one vertices, four-cycles). In our case we also counted the objects who were determined not to be in the future solution, namely the dominating vertices. We allow to count such structural information only fractional. Clearly, we have to show that the budget never increases on applying reduction rules. This has been done in our case with Lemma 2.3.2. We even have to guarantee that the measure decreases in case of recursive calls. In our case this easily can be observed. But additionally once our budget has been consumed we must be able to give an appropriate answer in polynomial time. As in general we counted more than only future solution objects this might become a hard and tedious task. In our case this was handled by Lemma 2.3.3. If we are able to fulfill all the recited conditions we can prove a running time of the form $\mathcal{O}^*(c^k)$.

By the above discussion we can conclude that Algorithm 6 solves k -NONBLOCKER in time $\mathcal{O}^*(4^k)$.

2.3.2. An Approach with a fine-grained Measure

We first state a new more detailed measure. Note that we use the notation of section 2.2.

$$\Psi(A, I) = k - \omega_{nb1}|INND| - \omega_{nb2}|INDO| - \omega_d|BLDO| - (1 - \omega_{nb2})|A|$$

where $\omega_d = 0.2725$, $\omega_{nb1} = 0.4550$ and $\omega_{nb2} = 0.7275$. We will use $\Psi(A, I)$ to measure the complexity of the now presented Algorithm 7. Note that Algorithm 7 is a parameterized version of Algorithm 5 in section 2.2. The main differences are the lines 2-7 where the algorithm is stopped once our measure $\psi(A, I) \leq 0$. The next lemma justifies this.

Lemma 2.3.4: In step 6 of Algorithm 6 $N^* := NB(D^*)$ is a Nonblocker set such that $|N^*| \geq k$.

Proof. Let the functions $pn : D_{\geq 1}^* \rightarrow N$ and $\tilde{pn} : D_0^* \rightarrow N$ be defined as in Lemma 2.3.3. Also note that Lemma 2.3.2 is valid with respect $\Psi(A, I)$. Then the following holds:

Algorithm 7 A simple algorithm for NONBLOCKER

- 1: Apply the reduction rules **RR1-RR4** from section 2.2 exhaustively in the order given by their increasing numbers.
 - 2: If there is $u \in I$ such that $N[u] \subseteq I$ then halt and answer NO.
 - 3: **if** $\psi(A, I) \leq 0$ **then**
 - 4: Greedily augment A to a dominating set D .
 - 5: Determine the sets $D_{\geq 1}$, D_0 and $N := NB(D)$.
 - 6: Apply **OR1**, **OR2** and **OR3** exhaustively which results in the dominating set D^* and $N^* = NB(D)$.
 - 7: **return YES**.
 - 8: **else**
 - 9: **if** possible choose a $v \in \text{BLND}$ such that $|N(v) \cap (\text{BLND} \cup \text{INND})| \geq 2$. **then**
 - 10: Binary branch on v (i.e., set v active in one branch, inactive in the other)
 - 11: **else if** possible choose a $v \in \text{BLDO}$ such that $|N(v) \cap (\text{BLND} \cup \text{INND})| \geq 3$. **then**
 - 12: Binary branch on v .
 - 13: **else**
 - 14: Solve the remaining instance in polynomial time using an EDGE COVER algorithm resulting in a dominating set D .
 - 15: Apply **OR1**, **OR2** and **OR3** exhaustively on D which results in the dominating set D^* and $N^* = NB(D)$.
 - 16: **end if**
 - 17: **end if**
-

$$\begin{aligned}
 k &\leq \omega_d |\text{BLDO}| + \omega_{nb1} |\text{INND}| + \omega_{nb2} |\text{INDO}| + (1 - \omega_{nb2}) |A| \\
 &= \omega_d (|\text{BLDO} \cap D| + |\text{BLDO} \cap N|) + \omega_{nb1} |\text{INND}| + \omega_{nb2} |\text{INDO}| + (1 - \omega_{nb2}) |A| \\
 &\leq (1 - \omega_{nb2}) (|\text{BLDO} \cap D| + |A|) + \omega_{nb2} (|\text{BLDO} \cap N| + |\text{INND}| + |\text{INDO}|) \\
 &\stackrel{\star}{\leq} (1 - \omega_{nb2}) |D| + \omega_{nb2} |N| \leq (1 - \omega_{nb2}) |D^*| + \omega_{nb2} |N^*| \\
 &= \omega_{nb2} \cdot |N^*| + (1 - \omega_{nb2}) \cdot |D_{\geq 1}^*| + (1 - \omega_{nb2}) \cdot |D_0| \\
 &= \omega_{nb2} \cdot |N^*| + (1 - \omega_{nb2}) \cdot \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + (1 - \omega_{nb2}) \cdot \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| \\
 &= \omega_{nb2} \cdot \left| N^* \setminus \left(\bigcup_{u \in D_{\geq 1}^*} pn(u) \cup \bigcup_{u \in D_0^*} \tilde{pn}(u) \right) \right| + \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| \\
 &\leq \left| N^* \setminus \left(\bigcup_{u \in D_{\geq 1}^*} pn(u) \cup \bigcup_{u \in D_0^*} \tilde{pn}(u) \right) \right| + \left| \bigcup_{u \in D_{\geq 1}^*} pn(u) \right| + \left| \bigcup_{u \in D_0^*} \tilde{pn}(u) \right| = |N^*|
 \end{aligned}$$

Note that in \star we used the fact that $((\text{BLDO} \cap D) \cup A) \subseteq D$ and $((\text{BLDO} \cap N) \cup \text{INND} \cup \text{INDO}) \subseteq N$. \square

Before we provide the run time analysis we show that the reduction rules **RR1-RR1** do not increase $\Psi(A, I)$. In each case we state the total decrease in $\Psi(A, I)$:

RR-1: A vertex $q \in (\text{BLDO} \cup \text{BLND})$ is set active, thus, the decrease is at least $(1 - \omega_{nb2}) - \omega_d = 0$.

RR-2: A vertex $u \in \text{BLDO}$ is set inactive yielding an decrease of $\omega_{nb2} - \omega_d = 0.455$.

RR-3: A vertex $u \in \text{BLND}$ is set active yielding a decrease of $(1 - \omega_{nb2}) = 0.2725$.

RR-4: A vertex $u \in \text{BLND}$ is moved to **INND** yielding a decrease of $\omega_{nb1} > 0$.

Run Time Analysis We will provide a run time analysis similar to the one in section 2.2 but with respect to $\Psi(A, I)$. Let v be the vertex chosen by Algorithm 7 in step 9 or 11, respectively. Let $n_{bl} = |N(v) \cap \text{BLND}|$ and $n_{in} = |N(v) \cap \text{INND}|$.

a) **The vertex v was chosen in step 9** The branching vector is $((1 - \omega_{nb2}) + n_{bl}\omega_d + n_{in}(\omega_{nb2} - \omega_{nb1}), \omega_{nb1})$. Note also that $n_{bl} + n_{in} \geq 2$.

b) **The vertex v was chosen in step 11** The branching vector is $((1 - \omega_{nb2} - \omega_d) + n_{bl}\omega_d + n_{in}(\omega_{nb2} - \omega_{nb1}), \omega_{nb2} - \omega_d)$. Note also that $n_{bl} + n_{in} \geq 3$.

Now by considering only finitely many branching vectors we can determine the run time. Therefore the run time discussion entails that Algorithm 7 solves k -NONBLOCKER in $\mathcal{O}^*(3.07^k)$ steps.

In this section we demonstrated how a direct algorithm solving k -NONBLOCKER can be derived. The main ingredient was to use a measure which was adapted to the problem. It especially reflected that every dominating vertex is related to an exclusive vertex in the nonblocker set. Due to this relation we were able to decrement our initial budget even in the case when a vertex has been set active, i.e., it was made dominating. This case seemed to be the main obstacle for deriving such a direct algorithm. Running the algorithm of [153] on the kernel for k -NONBLOCKER of F. Dehne *et al.* [34] provides an algorithm with running time $\mathcal{O}^*(1.5088^{1.75k}) \subseteq \mathcal{O}^*(2.045^k)$. Nevertheless, Algorithms 6 and 7 are the first direct approaches solving k -NONBLOCKER.

2.4. Case Study: Connected Vertex Cover

In this section we will present a parameterized algorithm for connected vertex cover. Given a graph $G(V, E)$ a set $V' \subseteq V$ of minimum cardinality has to be found such that V' is a vertex cover which is connected at the same time. More formally:

CONNECTED VERTEX COVER (CVC)

Given: $G(V, E)$, and the parameter k .

We ask: Find $V' \subseteq V$ such that for all $e \in E$ we have $V' \cap e \neq \emptyset$, $|V'| \leq k$ and $G[V']$ is connected.

On this topic there has been done already numerous work. We only mention two articles: The one of H. Fernau and D. Manlove [52] and the other of D. Mölle, S. Richter and P. Rossmanith [119] who achieved run times of $\mathcal{O}^*(2.9316^k)$ and $\mathcal{O}^*(2.7606^k)$, respectively. Both approaches are based on listing all minimal vertex covers of size up to k . Then in a second step they add additional vertices to the vertex cover to receive connected sets. The same approach is also followed here. But in contrast we also use a more flexible measure which enables us to better balance the two phases.

As in the course of the forthcoming recursive algorithm vertices will be fixed to belong to the future solution we focus on an annotated version of the above problem:

ANNOTATED CONNECTED VERTEX COVER (ACVC)

Given: $G(V, E)$, a subset $Q \subseteq V$, and the parameter k .

We ask: Find $V' \subseteq V$ such that for all $e \in E$ we have $V' \cap e \neq \emptyset$, $|V'| \leq k$, $Q \subseteq V'$ and $G[V']$ is connected.

The next lemma will help us to justify some of the branching rules of the forthcoming algorithm.

Lemma 2.4.1: Let V' be a connected vertex cover and assume that $t \in V'$ is not a cut-vertex. If $G_t := G[V' \setminus \{t\}]$ consists of exactly two components V'_1 and V'_2 and $N(t) \subseteq V'$, then there is a connected vertex cover V'' with $t \notin V''$ and $|V''| \leq |V'|$.

Proof. Observe that $V'_1 \cup V'_2$ is a vertex cover by $N(t) \subseteq V'$. If it is also connected then the claim is shown. Otherwise as t is not a cut-vertex there must be another vertex $z \notin V'$ such that $V'_1 \cup V'_2 \cup \{z\}$ is a connected vertex cover. \square

Reduction Rules We introduce the following reduction rules:

Deg1: Let $u \in V \setminus Q$ such that $N(u) = \{v\}$. Then delete u and set $Q := Q \cup \{v\}$.

CutVertex: Let $u \in V \setminus Q$ such that $G[V \setminus \{u\}]$ contains at least two components. Then set $Q := Q \cup \{u\}$.

Deg2a: Let $u \in V \setminus Q$ such that $d(u) = 2$. Then delete u and set $Q := Q \cup N(u)$.

Contract: If there are $u, v \in Q$ such that $\{u, v\} \in E$ then contract $\{u, v\}$ (substituting double edges by simple ones).

Deg2b: Let $u \in Q$ such that $N(u) = \{x_1, x_2\} \cap Q = \emptyset$. Then delete u and introduce the edge $\{x_1, x_2\}$.

Lemma 2.4.2: The reduction rules are sound.

Proof. In the following arguments, let C be a solution to ACVC of minimum cardinality.

Deg1: Any connected vertex set C with $u \in C$ (and necessarily $v \in C$) is not minimum as $C \setminus \{u\}$ is smaller.

CutVertex: Otherwise, we cannot reach a connected solution.

Deg2a: We will show that in all cases, $N(u) \subseteq C$ and $u \notin C$ holds without loss of generality, hence validating that we set $Q := Q \cup N(u)$. (a) If $u \notin C$, then clearly $N(u) \subseteq C$, since C is a vertex cover. So, suppose (b) $u \in C$. Due to Lemma 2.4.1 in order to falsify our claim, $N(u) \subseteq C$ must be false, i.e., there is one $h \in N(u) \setminus C$. Since C is connected and $u \in C$, h is unique. Note that $N(h) \cap (C \setminus \{u\}) \neq \emptyset$ by **Deg1**. Thus, $(C \setminus \{u\}) \cup \{h\}$ is a connected vertex cover of the same cardinality, leading us back to case (a).

Contract: Let G' be the graph obtained by applying the reduction rule. It is a straight forward task to show that G' contains a solution of size at most k to ACVC iff G does.

Deg2b: Let G' be the graph evolved from the application of the rule. If C' is a connected vertex cover for G' then $|C' \cap \{x_1, x_2\}| \geq 1$. Thus, $C' \cup \{u\}$ is a connected vertex cover for G of size $|C'| + 1$. If C is a connected vertex cover for G then $|C \cap \{x_1, x_2\}| \geq 1$. Thus $C \setminus \{u\}$ is connected vertex cover for G' .

\square

Let $\mathcal{B}_{ij} := \{u \in V \setminus Q \mid |N(u) \cap Q| = i, |N(u)| = j\}$ and $\mathcal{B}_{ij}^{\geq} := \bigcup_{\substack{i' \geq i \\ j' \geq j}} \mathcal{B}_{i'j'}$.

Note that in a reduced instance $\mathcal{B}_{01} = \mathcal{B}_{11} = \mathcal{B}_{02} = \mathcal{B}_{12} = \mathcal{B}_{22} = \emptyset$ due to reduction rules **Deg1**, **CutVertex** and **Deg2a**. Now, we can present Algorithm 8.

Observe that we can solve CVC by calling Algorithm 8 for all $h \in V$ with the sets $Q = \{h\}$ and $I = \emptyset$.

Correctness The lemma below is needed as an intermediate step towards showing correctness of Algorithm 8.

Lemma 2.4.3: In step 8 of Algorithm 8 we always have that a) $Q \neq \emptyset$ and b) G is connected.

Proof. The claim is true for the first invocation of Algorithm 8 as we have $Q = \{h\}$ for some $h \in V$.

a) Note that the only reduction rules who actually delete vertices from Q are **contract** and **Deg2b**. In the first case an edge consisting of two vertices from Q is contracted. Thus, afterwards Q contains at least one vertex. In the second case note that **Deg2b** only is applied in step 1 in a very restricted setting. The set I has been constructed in step 15 of the last recursive call. By this $I \subset Q$ in the current call of Algorithm 8. Also in the previous recursive call any $u \in I$ had a neighbor v (the branching vertex) such that there is some $d \in N(v) \cap Q$. Note that the vertex d is not affected by step 1. Thus, in step 2 $Q \neq \emptyset$.

b) Otherwise, step 4 failed. □

Lemma 2.4.4: Algorithm 8 solves ACVC correctly.

Proof. By Lemma 2.4.2 the reduction rules are sound. We now show that if no case in Phase I applies then the Steiner-Tree-Algorithm in Phase II can be used. Firstly, if the first part of the or-statement in step 4 applies then the size of the vertex cover is greater than k . Note that any **Contract** or **Deg2b**-application implicitly puts a vertex (which is not present in the current graph) in the final vertex cover. Thus, we correctly answer **NO**. Secondly, if $V = Q$ then by **Contract** and Lemma 2.4.3.b) we have $|V| = 1$. Thus, we have trivial instance and can return **YES**.

Otherwise there is a vertex $h \in V \setminus Q$. Additionally we can require that $N(v) \cap Q \neq \emptyset$ by Lemma 2.4.3. If there is at least one such vertex h where $N(h) \not\subseteq Q$ then we have $h \in \bigcup_{\substack{\ell \geq 3 \\ z < \ell}} \mathcal{B}_{z\ell}$ due to the reduction rules. Thus, some case in Phase I applies. If for all such h we have $N(h) \subseteq Q$ then Q is an independent vertex cover by **Contract**. The final task is to find some minimum cardinality set $B \subseteq V \setminus Q$ such that $G[Q \cup B]$ is connected. This is exactly the Steiner-Tree problem and therefore we can apply an appropriate existing algorithm for this task in Phase II.

Any branching is exhaustive except the ones in step 13 and step 20.

Consider step 13. Note that here we skipped the possibility that 1.) $v, x_1, x_2 \in Q$ and 2.) $v \in Q, x_1, x_2 \notin Q$. 1.) Suppose that $v, x_1, x_2 \in C$ where C is an solution to ACVC. By Lemma 2.4.1 we find a second in cardinality no worse solution \tilde{C} where $v \notin \tilde{C}$. Now \tilde{C} can be found in the first branch a) where v is deleted. Thus the possibility in 1.) can be skipped. 2.) can be skipped as $\{x_1, x_2\}$ must be covered.

We turn to step 20. Here in the last recursive call d) we delete r and therefore any solution C with $v, b, c, r \in C$ is not considered. But notice once more by Lemma 2.4.1 a

Algorithm 8 An Algorithm for ANNOTATED CONNECTED VERTEX COVER**Input:** A connected graph $G(V, E)$, a subset $Q \subseteq V$, an integer k and a vertex set I .**Output:** A set $V' \subset V$ such that $Q \subseteq V'$, $G[V']$ is connected and V' is a vertex cover.**Procedure:** SolveCVC(G, Q, k, I)

- 1: Apply **Deg2b** on every $u \in I$ (delete u and add $\{x_1, x_2\}$ where $N(u) = \{x_1, x_2\}$).
- 2: $I := \emptyset$.
- 3: Apply the reduction rules **CutVertex**, **Deg1**, **Deg2a** and **Contract** exhaustively with priorities corresponding to the given order.
- 4: **if** $|Q| + \#\{\mathbf{Contract} \text{ applications}\} + \#\{\mathbf{Deg2b} \text{ applications}\} > k$ **or** $G(V, E)$ is disconnected **then**
- 5: **return** NO
- 6: **else if** $V = Q$ **then**
- 7: **return** YES.
- 8: **else if** $\exists v \in \mathcal{B}_{14}^{\geq}$ such that $N(v) \cap (V \setminus Q) \neq \emptyset$ **then** {Begin Phase I}
- 9: Branch binary by a.) setting $Q := Q \cup \{v\}$
 b) deleting v and setting $Q := Q \cup N(v)$.
- 10: **else if** $\mathcal{B}_{13} \cup \mathcal{B}_{23} \neq \emptyset$ **then**
- 11: Choose $v \in (\mathcal{B}_{13} \cup \mathcal{B}_{23})$ according to the next priorities:
- 12: 1. $v \in \mathcal{B}_{13}$ and $\{x_1, x_2\} \in E$ where $N(v) \setminus Q = \{x_1, x_2\}$.
 a) Delete v , $Q := Q \cup \{x_1, x_2\}$
 b) Delete x_2 , $Q := Q \cup \{v, x_1\} \cup N(x_2)$
 c) Delete x_1 , $Q := Q \cup \{v, x_2\} \cup N(x_1)$.
- 13: 2. $\exists u \in (N(v) \setminus Q) : d(u) = 3$
 Branch binary by a.) setting $Q := Q \cup \{v\}$
 b) deleting v , setting $I := \{u \mid u \in N(v) \wedge |N(u) \setminus \{v\}| = 2 \wedge N(u) \cap Q = \emptyset\}$
 and $Q := Q \cup N(v)$.
- 14: 3. $v \in \mathcal{B}_{23}$
 Choose $u \in (N(v) \setminus Q)$ and branch binary by
 a.) setting $Q := Q \cup \{u\}$
 b) deleting u and setting $Q := Q \cup N(u)$.
- 15: 4. $v \in \mathcal{B}_{13}$ and thus $N(v) \setminus Q = \{b, c\}$.
- 16: 4.1 $\exists r \in N(b) \cap N(c)$ where $r \neq v$.
 a) Delete v , $Q := Q \cup \{b, c\}$
 b) Delete b , $Q := Q \cup \{v\} \cup N(b)$
 c) Delete c , $Q := Q \cup \{v, b\} \cup N(c)$
 d) Delete r , $Q := Q \cup \{v, b, c\} \cup N(r)$
- 17: 4.2 $N(b) \cap N(c) = \emptyset$.
 a) Delete v , $Q := Q \cup \{b, c\}$
 b) $Q := Q \cup \{v, b\}$
 c) Delete b , $Q := Q \cup \{v, c\} \cup N(b)$
 d) Delete b, c , $Q := Q \cup \{v\} \cup N(b) \cup N(c)$
- 18: **else** {Begin Phase II}
- 19: Apply the Steiner-Tree-Algorithm of [9] with Q as the terminal set.
- 20: **end if**

solution \tilde{C} no greater in size is guaranteed with $v \notin \tilde{C}$. \tilde{C} is found in the recursive call a) in step 20. \square

Run Time Analysis We use a rather simple measure:

$$\phi(V, Q) := k - \omega \cdot |Q| - c \text{ where } \omega < 1$$

Here $c = \#\{\mathbf{Contract}\text{ applications}\} + \#\{\mathbf{Deg2b}\text{ applications}\}$, i.e., it counts the number of **Contract** and **Deg2b** applications. Observe that c counts the number of vertices of the original instance fixed to be in Q which are not present anymore in the current instance. Therefore, a decrease of the initial budget k by an amount of c is justified.

Lemma 2.4.5: The reduction rules do not increase the measure $\phi(V, Q)$

Proof. Observe that the deletion of a vertex $u \in V \setminus Q$ leaves $\phi(V, Q)$ unchanged and its addition to Q decreases the measure. Thus, the claim is true for **Deg1**, **Deg2a** and **CutVertex**. After the application of **Contract** or **Deg2b** the set Q decreased by one element. On the other hand c went up by one. Hence, there is a decrease of $(1 - \omega)$. \square

Here we like to point out the following observation: Once we have $\phi(V, Q) < 0$ during our algorithm, line 4 of Algorithm 8 applies and **YES** is returned. Thus, $\phi(V, Q)$ indeed can be used to derive a run time of the form $\mathcal{O}^*(c^k)$.

Phase II By Björklund *et al.* [9] the STEINER TREE problem can be solved in time $\mathcal{O}^*(2^\ell)$ where $\ell = |Q|$ is the number of given terminals. But note that by

$$\phi(V, Q) := k - \omega \cdot \ell - c \geq k - \omega \cdot \ell - (k - \ell) = (1 - \omega) \cdot \ell$$

we can upper bound step 24 in Algorithm 8 by $\mathcal{O}^*(2^{\frac{1}{1-\omega}\phi(V, Q)})$. Observe that we used the fact $k - \ell \geq c$ which is due to step 4.

Phase I We further have to find the correct branching vectors in case we are branching in Algorithm 8:

Step 9 For $v \in \mathcal{B}_{ij}$, we derive the branching vector $(i \cdot (1 - \omega) + \omega, (j - i) \cdot \omega)$, where $i \geq 1, j \geq 4$ and $i < j$. Observe that we only have to consider branching vectors up to $j = 4$. Any branching vector where $j > 4$ is dominated by one of the latter.

Step13 We have that $d(x_1) \geq 3$ and thus by deleting x_1 in case c) at least one further vertex $y \notin \{v, x_2\}$ is adjoined to Q , i.e., $|N(x_1) \setminus \{v, x_2\}| \geq 1$. Analogous arguments hold for x_2 in case b). Thus a $(1 + \omega, 2 + \omega, 2 + \omega)$ branching vector is entailed. Note that in any branch **Contract** can be applied.

Step 15

1. Assume $v \in \mathcal{B}_{13}$. Then let $\{u, z\} = N(v) \setminus Q$. By the previous priority there is no edge $\{u, z\} \in E$. Then let $u \in \mathcal{B}_{s3}$ where $0 \leq s \leq 2$. Then the following branching vectors are entailed if $s \geq 1$: $((1 - \omega) + \omega, 2\omega + (1 - \omega))$ and if $s = 0$: $((1 - \omega) + \omega, \omega + 1)$. Note that in the last derived branching vector in the second case **Deg2b** applies in the next recursive call as $I \neq \emptyset$.
2. Assume $v \in \mathcal{B}_{23}$. Consider $\{u\} = N(v) \setminus Q$, $u \in \mathcal{B}_{s3}$. By the same analysis as in the previous case we arrive at the following branching vectors:
 If $s \geq 1$, $(2 \cdot (1 - \omega) + \omega, \omega + (1 - \omega))$ and
 if $s = 0$, $(2 \cdot (1 - \omega) + \omega, 1)$.

Step17 Note that $d(u) \geq 4$ due to step 15 and due to step 9 $N(u) \cap Q = \emptyset$. Hence, the branching vector $(\omega, 4\omega + 2 \cdot (1 - \omega))$ is entailed. Note the application of **Contract** in the second part of the branch.

Step 20 Note by the previous branching cases $d(b) \geq 4$, $d(c) \geq 4$ and $(N(b) \cup N(c)) \cap Q = \emptyset$. Firstly, suppose $N(r) \cap Q \neq \emptyset$ then a branching vector $(2\omega, 2 + 2\omega, 3 + 2\omega, 3)$ follows due to the application of **Contract** with respect to r in the second and third part of the branching. Secondly, if $N(r) \cap Q = \emptyset$ we get a branching vector $(2\omega, 1 + 3\omega, 2 + 3\omega, 3 + \omega)$. Note that the additional amount of ω in the last entry is due to the fact that $|N(r) \setminus (\{c, b\} \cup Q)| \geq 1$ which is due to **Deg2a**.

Step 22 This case entails a straight-forward $(2\omega, 2, 2 + 3\omega, 1 + 6\omega)$ branching vector as $N(b) \cap N(c) = \{v\}$.

On the basis of the above discussion we get the next lemma

Lemma 2.4.6: CONNECTED VERTEX COVER can be solved in time $\mathcal{O}^*(2.4882^k)$ and $\mathcal{O}^*(1.8698^n)$.

Proof. By choosing $\omega = 0.23956$ the maximum branching number of the above branching vectors is 2.4882. Additionally, by the choice of ω the run time of Phase II can be upper bounded by $\mathcal{O}^*(2.4882^k)$. To achieve the run time upper bound in terms of n we use a method of [139]: By invoking Algorithm 8 for every $1 \leq k' \leq n/2 + \alpha$ where $\alpha = 0.1842n$ and by iterating over any non-separating independent set with maximum size $n/2 - \alpha$ we get the bound of $\mathcal{O}^*(1.8658^n)$. \square

In this case study we have seen that we some moderate effort we could achieve an improved run time for a already broadly studied problem. This was possible even though the basic method did not change substantially. We first branched towards creating a vertex cover and then in a second phase additional vertices had to be added for the sake of connectivity. This case study showed what an impact a proper chosen measure can have for the understanding of the problem and the designed algorithm. Let us also mention the cases which determine the run time: 1. Step 9 with $j = 4$ and $i = 1$ and 2. Phase II.

2.5. Case Study: Edge Dominating Set

In a further case study we show how to use a problem-tailored measure to improve the parameterized run time of an already considered problem. We focus on:

k-EDGE DOMINATING SET (*k*-EDS)

Given: $G(V, E)$, and the parameter k .

We ask: Find $E' \subseteq E$ such that for all $e \in E$ we have $e \cap (\bigcup_{e' \in E'} e') \neq \emptyset$, $|E'| \leq k$ and E' is minimal.

Note that in the definition of *k*-EDS we required that any solution is also minimal. This is no restriction as this variant is polynomial time equivalent to the variant where this property is dropped. Also note that the size of a minimum (and thus minimal) edge dominating set equals the size of a minimum maximal matching [18, 115, 158]. In fact a minimal edge dominating set can be transformed to a maximal matching of the same size in polynomial time (\star).

Note that any solution E' to *k*-EDS has the property that $V(E')$ is a vertex cover of size at most $2|E'|$. This fact was used by Fernau [47] to design an $\mathcal{O}^*(2.6181^k)$ -algorithm. This algorithm is based on enumerating all minimal vertex covers of size no more than $2k$. A similar exponential time algorithm was developed by J.M.M. van Rooij and H. L. Bodlaender [152] consuming $\mathcal{O}^*(1.3226^n)$ time. We basically analyze a parameterized variant of their algorithm enriched by two special branching cases. Thus, during the course of the algorithm we handle a set L of vertices which is supposed to be covered by the final edge dominating set. Therefore we present an annotated version of *k*-EDS.

k-ANNOTATED EDGE DOMINATING SET (*k*-AEDS)

Given: $G(V, E)$, a set $L \subseteq V$, and the parameter k .

We ask: Find $E' \subseteq E$ such that for all $e \in E$ we have $e \cap (\bigcup_{e' \in E'} e') \neq \emptyset$, $|E'| \leq k$, $L \subset (\bigcup_{e' \in E'} e')$ and E' is minimal (with respect to the edge-domination-property).

As indicated above (\star) the next problem is polynomially equivalent to *k*-AEDS:

k-ANNOTATED MAXIMAL MATCHING (*k*-AMM)

Given: $G(V, E)$, a set $L \subseteq V$, and the parameter k .

We ask: Find $M \subseteq E$ such that $L \subset (\bigcup_{e' \in M} e')$, $|M| \leq k$ and M is a maximal matching in $G(V, E)$.

In the forthcoming algorithm *k*-AMM will be the base problem.

We call a given annotated instance *redundant* if $G[V \setminus L]$ exclusively consists of components C_1, \dots, C_ℓ which are either single vertices or edges, i.e., these components are K_1 's and K_2 's.

Lemma 2.5.1 ([47]): A minimum cardinality solution to a redundant instance of *k*-AEDS without edge-domination-minimality can be found in polynomial time.

Note that a solution to *k*-AEDS might not exist. The set L might constrain that every augmenting set $E' \supset L$ which is an edge dominating set is not minimal.

We now present Algorithm 9. The Algorithm follows the strategy to list all minimal vertex covers VC of size at most $2k$ until a redundant instance is reached. This redundant

instance will be solved in polynomial time using Lemma 2.5.1 yielding a solution $S \subseteq E$. If S is not minimal then VC was not the right choice and we can answer **NO**. Otherwise S can be transformed to a solution of k -AMM. Details will be clarified in Lemma 2.5.3.

Algorithm 9 An Algorithm for Independent k -Edge Dominating Set, a.k.a. k -maximal Matching

Input: A Graph $G(V, E)$ and the parameter k .

Output: YES if a solution to k -AMM exists, NO otherwise.

SolveEDS(G, \emptyset, \emptyset)

Procedure: SolveEDS(G, U, U^s)

```

1: if  $\lambda(G, U, U^s) \leq 0$  then
2:   return NO
3: else if  $\exists v \in V \setminus (U \cup U^s)$  such that  $d_{G[V \setminus (U \cup U^s)]}(v) \geq 3$  then {Begin Phase I}
4:   Branch by a)  $U := U \cup \{v\}$  and b.) Delete  $v$ ,  $U := U \cup N(v)$ 
5: else if  $\exists$  a cycle component  $p_1, \dots, p_\ell$ ,  $\ell \geq 4$  in  $G[V \setminus (U \cup U^s)]$  then {Begin Phase II}
6:   if  $\ell = 4$  then
7:     Branch a.) Delete  $p_2, p_4$ ,  $U := U \cup \{p_1, p_3\}$  b) Delete  $p_1$ ,  $U := U \cup \{p_2, p_4\}$ 
8:   else if  $\ell = 6$  then
9:     Branch a)  $U := U \cup \{p_1, p_4\}$  b) Delete  $p_4$ ,  $U := U \cup \{p_1, p_3, p_5\}$  c) Delete  $p_1$ ,
        $U := \{p_2, p_6\}$ 
10:  else
11:    Branch a.)  $U := U \cup \{p_1\}$  b) Delete  $p_1$ ,  $U := U \cup \{p_2, p_\ell\}$ 
12:  end if
13: else if  $\exists$  a path component  $p_1, \dots, p_\ell$ ,  $\ell \geq 4$  in  $G[V \setminus (U \cup U^s)]$  then
14:   Branch a.)  $U := U \cup \{p_3\}$  b) Delete  $p_3$ ,  $U := U \cup \{p_2, p_4\}$ 
15: else if  $\exists$  a path or cycle component  $p_1, \dots, p_3$  in  $G[V \setminus (U \cup U^s)]$  then
16:   Branch a.)  $U^s := U^s \cup \{p_2\}$  b) Delete  $p_2$ ,  $U^s := U^s \cup \{p_1, p_3\}$ 
17: else
18:   Solve the problem in polynomial time using a maximum matching algorithm.
19: end if

```

Correctness We first point out that any recursive branching step in Algorithm 9 is exhaustive except the one in step 7. Here case *a*) should be considered. Additionally to p_1 also p_3 is adjoined to U which has to be justified. Note that if $p_3 \notin U$ then it follows that p_2, p_4 have to be adjoined yielding a non-minimal solution. If both $p_1, p_3 \in U$ then p_2, p_4 have to be deleted for the sake of minimality.

Notice also that vertices that are not taken into the vertex cover during the enumeration phase (and hence go into the complement, i.e., an independent set) can be deleted whenever it is guaranteed that all their neighbors (in the current graph) are moved into the vertex cover.

Let $X^\ell(B) := \{\{p_1, \dots, p_\ell\} \mid p_1, \dots, p_\ell \text{ is a path component in } G[V \setminus B]\}$. For example, $X^1(B)$ collects the isolated vertices in $G[V \setminus B]$. In a sense, $\ell = 1$ is the smallest meaningful value of ℓ ; however, the case $\ell = 0$ will be useful as a kind of boundary case in the following.

Then the measure is defined as follows:

$$\begin{aligned} \lambda(G, U, U^s) &:= 2k - \omega \cdot |U| - (2 - \omega)|U^s| \\ &- \chi(\forall v \in V \setminus (U \cup U^s) : d_{G[V \setminus (U \cup U^s)]}(v) \leq 2) \cdot \\ &\quad (\sum_{i \geq 0} \gamma_i \cdot |X^i(U \cup U^s)|) \end{aligned}$$

where $\omega = 0.8909$, $\gamma_0 = \gamma_1 = 0$, $\gamma_2 = 1$, $\gamma_3 = 0$, $\gamma_4 = 0.2365$ and $\gamma_j = 0.3505$ for $j \geq 5$. Note that the initial budget $2k$ is only decreased by adjoining vertices to U or U^s .

To show the correctness of the algorithm it remains to show that steps 1 and 18 are correct. This is done by the next two lemmas.

Lemma 2.5.2: If in step 1 of Algorithm 9 $\lambda(G, U, U^s) < 0$ then there is no annotated k -maximal matching M extending $L := U \cup U^s$ with $|M| \leq k$.

Proof. Suppose the contrary and let M be a solution to k -AMM. Let $V_M = (\bigcup_{e' \in M} e')$ and note that $L \subseteq V_M$ (where here $L = U \cup U^s$). Let $m : V \rightarrow V$ be the partial matching function induced by M that easily extends to sets by setting $m(B) = \{m(v) \mid v \in B\}$. If $\lambda(G, U, U^s) < 0$ happens after a recursive call of Phase I, then $2k < \omega \cdot |U| < |U| \leq |V_M|$.

So we should discuss in details if $\lambda(G, U, U^s) < 0$ happens after a recursive call of the second Phase. Since M is a maximal matching, we know that for each $i \geq 2$ and for all paths $p_1, \dots, p_i \in X^i(U \cup U^s)$, there exists some $1 \leq j \leq i$ such that $p_j \in V_M$. Hence, $|X^i(U \cup U^s)| \leq |V(X^i(U \cup U^s)) \cap V_M|$ (\star_1). Consider now $u \in U^s$; then there exists a path p_1, p_2, p_3 in $G[V \setminus U]$. Firstly, suppose $u = p_1$. Thus, u has been adjoined to U^s in case *b*) of step 16. Since p_2 got deleted, $p_2 \notin V_M$. Let $m(u)$ be the distinct vertex u is matched to under M . By $p_2 \notin V_M$ we have $m(u) \in U$. Thus, summing up for u and $m(u)$ the vertices are counted by two in V_M whereas we decremented λ by $\omega + (2 - \omega) = 2$. (\star_2). Secondly, if $u = p_2$ then if $m(p_2) \in U$ the same reasoning applies. Otherwise, w.l.o.g., $m(p_2) = p_1$. Then, in λ both vertices were counted by a fraction of $(2 - \omega)$ where in V_M they both together have been counted by two (\star_3). This seems to be an invalid argument at first glance, but will be justified below. All

this has to be considered more formally. Notice that, since we are in Phase II, the term $\chi(\forall v \in V \setminus (U \cup U^s) : d_{G[V \setminus (U \cup U^s)]}(v) \leq 2)$ equals one.

$$\begin{aligned}
 2k &< \omega \cdot |U| + (2 - \omega)|U^s| + \left(\bigcup_{i \geq 0} \gamma_i \cdot |X^i(U \cup U^s)|\right) \\
 &\stackrel{\star_1, \star_4}{\leq} \omega \cdot |U| + (2 - \omega)|U^s| + \left(\bigcup_{i \geq 2} \gamma_i \cdot |V(X^i(U \cup U^s)) \cap V_M|\right) \\
 &\stackrel{\star_5}{\leq} \omega \cdot |U \cap V_M| + (2 - \omega)|U^s \cap V_M| + \left(\bigcup_{i \geq 2} |V(X^i(U \cup U^s)) \cap V_M|\right) \\
 &\stackrel{\star_2/\star_3}{\leq} |U \cap V_M| + |(U^s \cap m(U)) \cap V_M| + |(U^s \setminus m(U)) \cap V_M| + \\
 &\quad |m(U^s) \setminus U| + \left(\bigcup_{i \geq 2} |V(X^i(U \cup U^s)) \cap V_M|\right) \\
 &\leq |V_M|
 \end{aligned}$$

We were also using that $\gamma_0 = \gamma_1 = 0$ (\star_4) and that M covers $U \cup U^s$ (\star_5). Now, let us explain the second last inequality in more detail. Observe that any vertex t from U^s is counted by an amount of one in $|V_M|$ but by an amount of $(2 - \omega)$ in λ . Thus, this might violate the inequality. Note that $m(t) \in (U \cup (m(U^s) \setminus U))$. Either t finds its partner in U or in the vertices in $m(U^s)$ which are not matched to a vertex in U . Therefore the drop of $(1 - \omega)$ with respect to t is compensated by the increase due to $m(t)$. In other words,

$$(1 - \omega)|U^s| = (1 - \omega)(|U^s \cap m(U)| + |U^s \setminus m(U)|) \leq (1 - \omega)(|U| + |m(U^s) \setminus U|).$$

Also note the set equality $(U^s \setminus m(U)) = (m(U^s) \setminus U)$. For the last inequality to hold we mention that $m(U^s) \subseteq V_M$. \square

Lemma 2.5.3: In step 18 the instance is polynomial time solvable.

Proof. Let $L = U \cup U^s$. First note that at this point the instance is redundant. Then by Lemma 2.5.1 we can compute an minimum size edge set E' which covers L and is an edge dominating set. If E' is not minimal with respect to the edge domination property then along the search tree we made a wrong choice on L . Then we can safely answer NO. Otherwise we apply the following transformation. Pick a component C of at least two edges in $G[E']$ which necessarily forms a star with center v . Let $e = \{u, v\} \in C$ and due to edge-domination-minimality there is a $z \in N_G(u)$ such that $z \notin V(E')$. Then substitute $\{u, v\}$ by $\{z, u\}$. Observe that the resulting edge set still covers L and is also edge-dominating. Finally, this procedure results in a set E'' which is a maximal matching (as it also is an edge dominating set) which additionally covers L . Clearly, $|E'| = |E''|$ is true. If $|E''| \leq k$, then we can answer YES and NO otherwise. \square

Run Time Analysis We will consider each step in Algorithm 9 where recursive calls are made:

Step 4 Here we adjoin one vertex in the first and three vertices in the second branching case to U . Thus, a branching vector $(\omega, 3 \cdot \omega)$ is entailed.

Step 7 Here we adjoin two vertices in the first and two vertices in the second branching case to U . Thus, a branching vector $(2 \cdot \omega, 2 \cdot \omega)$ can be derived.

Step 9 Here we adjoin two vertices in the first, three vertices in the second and two vertices in the third branching case to U . In the first case also two K_2 's in $G[V \setminus (U \cup U^s)]$ are produced and thus $|X^2|$ is increased by two. Thus, a branching vector $(2 \cdot \omega + 2, 3\omega, 2\omega)$ can be deduced.

Step 11 $\ell = 5$ or $\ell \geq 7$: In the first case we adjoin a vertex to U , but also create a path with $\ell - 1$ vertices which means that $|X^{\ell-1}|$ increases by one. The second case adjoins to vertices to U and creates a path with $\ell - 3$ vertices. This yields the branching vector $(\omega + \gamma_{\ell-1}, 2\omega + \gamma_{\ell-3})$.

Step 14 The first case puts one vertex into U and creates a path with two vertices and another one with $\ell - 3$ vertices. On the other hand, a path with ℓ vertices is destroyed. The second case puts two vertices into U , creates a path with $\ell - 4$ vertices and destroys one with ℓ vertices. This yields as branching vector $(\omega - \gamma_{\ell} + 1 + \gamma_{\ell-3}, 2\omega - \gamma_{\ell} + \gamma_{\ell-4})$.

Step 16 The first case increases $|U^s|$ by one the second by two and hence $(2-\omega, 2 \cdot (2-\omega))$ is the corresponding branching vector.

Note that in Phase I the paths in $G[U \cup U^s]$ are not counted in λ . This is important as otherwise due to branching in step 4 of Algorithm 9 a P_5 can result in a P_4 by vertex deletion. Due to this the measure would be raised by an amount of $\gamma_5 - \gamma_4 > 0$ and the given branching vector would not be appropriate. Note that upon entering Phase II λ is not increased. By the above run time analysis we can conclude the next theorem. We point out that once again only a finite set of branching vectors has to be considered. Namely, in the run time analysis of steps 11 and 14 we can restrict our considerations only to cases where $\ell \leq 9$.

Theorem 2.5.4: EDGE DOMINATING SET can be solved in time $\mathcal{O}^*(1.5433^{2k}) \subseteq \mathcal{O}^*(2.3819^k)$ consuming polynomial space.

The last run time analysis showed once more that with an appropriate measure a better run time can be proven. Also due to this method the case where there is a cycle of length six in $G[V \setminus (U \cup U^s)]$ (step 9 in Algorithm 9) has been identified as critical. Therefore, a special branching rule for this case has been invented. This is a further point where we gained on run time.

2.6. Obtaining The Weights

In the *Measure&Conquer*-approach of analyzing search trees one always ends up with a set of recurrences depending on some variables $\omega_1, \dots, \omega_\ell$ (also called *weights*). Each recurrence represents a search tree which evolves from it. The size of a search tree generated by the algorithm is estimated by the greatest positive non-complex root of the characteristic polynomial (we always will refer to this root). Thus, given a set of characteristic polynomials $P_1(\omega_1, \dots, \omega_\ell), \dots, P_h(\omega_1, \dots, \omega_\ell)$ the task is to find assignments to the weights such that maximum root of the polynomials is minimized.

We give this problem a name:

POLYNOMIAL MINWEIGHT (PMW)

Given: A set of polynomials $P_1(\omega_1, \dots, \omega_\ell), \dots, P_h(\omega_1, \dots, \omega_\ell)$ of the form $P_i(\omega_1, \dots, \omega_\ell) = 1 - (\sum_{j=1}^{n_i} x^{-(f_i(\omega_1, \dots, \omega_\ell))})$ (where f_i is linear) and a set of linear constraints C on the weights.

Task: Find an assignment to $\omega_1, \dots, \omega_\ell$ which obeys C such that

$$\max_{1 \leq i \leq h} \{root(P_i(\omega_1, \dots, \omega_\ell))\}$$

is minimum.

Here $root()$ returns the largest non-complex root of the polynomial given as argument. The specific form of the polynomials is due to fact that they are derived from branching algorithms. The number n_i expresses how many subproblems have been created.

The functions f_i correspond to the branching vectors. If we have a branching vector $(\omega_1, \omega_2 + 1 - \omega_3)$ then $f_1 = \omega_1$ and $f_2 = 1 + \omega_2 - \omega_3$. Generally, these functions have the form $f_i = C + \sum_{j=1}^{\ell} c_j \cdot \omega_j$ where the C and the c_j 's are constants.

One way of obtaining good weights is to use local search (as already briefly introduced in chapter 1.1.2). Starting from initial weights, we examine the direct neighborhood to see if we can find an weight assignment which provides a better upper bound. In practice, this approach works quite well, especially if we use compiled programs. Then an amount of hundreds of characteristic polynomials and several weights can be handled. There is also a formulation as a convex program [78]. For this problem class there are efficient solvers available. An alternative is the approach of Eppstein [38]. We will consider the first two approaches in more detail.

2.6.1. Local Search

Local Search is a simple but in practice powerful meta-heuristic. The basic ingredient is that we start from some initial assignment to the weights. Then we calculate the root of each polynomial $P_i(\omega_1, \dots, \omega_\ell)$. The maximum root is the current solution. This is the *value* of the assignment. We then try to locally improve this solution by marginally varying the weight assignments. The current solution can be interpreted as a point in \mathbb{R}^ℓ . Around this point we create a local mesh of finite radius, see Figure 2.1. Every point in this mesh will be seen as possible assignment. We calculate the value of that point. If none of these values is smaller than the current value, we create a finer mesh.

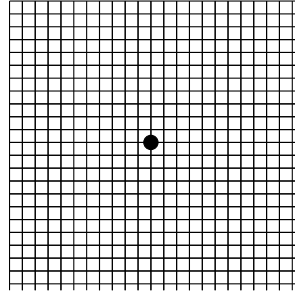


Figure 2.1.: The finite mesh around a current solution.

Otherwise we take the point with the smallest value as current solution. We apply this procedure a finite number of times.

To fulfill this task we use a simple matlab program whose usage is described in Appendix 11.1.

2.6.2. Convex Programming

A very general technique to determine the weights in a measure is to use convex programming which first was used by S. Gaspers [78]. We will show how PMW can be modeled by a convex program. Remember that in PMW we are given a set of polynomials obeying a certain form:

$$P_i(\omega_1, \dots, \omega_\ell) = 1 - \left(\sum_{j=1}^{n_i} x^{-f_i(\omega_1, \dots, \omega_\ell)} \right)$$

We derive the following convex program Γ :

$$\begin{aligned} & \min z \\ & \text{subject to} \\ & C_i : 1 \geq \sum_{j=1}^{n_i} 2^{-z f_i(\omega_1, \dots, \omega_\ell)} \quad \forall 1 \leq i \leq n \\ & \text{and the given constraints } C \end{aligned}$$

Clearly, the target function is convex. By deriving the second derivative of the constraints we see that also they are convex as it is strictly positive.

Proposition 2.6.1: Let $\lambda_1, \dots, \lambda_\ell$ be a weight assignment to $\omega_1, \dots, \omega_\ell$. Then $\alpha = \max_{1 \leq i \leq h} \{\text{root}(P_i(\lambda_1, \dots, \lambda_\ell))\} \iff \lambda_1, \dots, \lambda_\ell$ together with $z := \log_2(\alpha)$ is a solution to Γ such that at least one constraint C_i is tight.

Proof. \Rightarrow : Let $\alpha_i := \text{root}(P_i(\lambda_1, \dots, \lambda_\ell))$.

$$\begin{aligned} 0 = 1 - \sum_{j=1}^{n_i} 2^{-\log_2(\alpha_i)(f_i(\lambda_1, \dots, \lambda_\ell))} &= 1 - \sum_{j=1}^{n_i} \alpha_i^{-(f_i(\lambda_1, \dots, \lambda_\ell))} \\ &\leq 1 - \sum_{j=1}^{n_i} \alpha^{-(f_i(\lambda_1, \dots, \lambda_\ell))} \\ &= 1 - \sum_{j=1}^{n_i} 2^{-\log_2(\alpha)(f_i(\lambda_1, \dots, \lambda_\ell))} \end{aligned}$$

\Leftarrow : Let $1 \leq t \leq n$ such that C_t is tight and z' is the value of the objective function.

$$1 - \sum_{j=1}^{n_i} 2^{-z'(f_i(\lambda_1, \dots, \lambda_\ell))} \geq 1 - \sum_{j=1}^{n_t} 2^{-z'(f_i(\lambda_1, \dots, \lambda_\ell))} = 0$$

Thus, $\text{root}(P_i(\lambda_1, \dots, \lambda_\ell)) \leq 2^{-z'}$. It follows $2^{-z'} = \max_{1 \leq i \leq h} \{\text{root}(P_i(\lambda_1, \dots, \lambda_\ell))\}$. \square

By Proposition 2.6.1 we see that PMW can be solved by means of convex programming. We used the AMPL-language to formulate the convex program. Then a standard solver like MINOS or IPOPT can be used to solve the program to optimality. Nevertheless, if the convex program involves several hundreds of inequalities then the system noticeable slows down. Due to practical purposes we derive a second tighter formulation Γ' of PMW. Due to their specific form the functions f_i can be expressed as $g_i + K_i$ where g_i does not contain a constant independent of the ω_i 's and K_i is such a constant.

Let Γ' be the following convex program:

$$\begin{aligned} &\min z \\ &\text{subject to} \\ &B_i \quad z \geq \omega_i \quad \forall 1 \leq i \leq \ell \\ &C_i : \quad 1 \geq \sum_{j=1}^{n_i} 2^{-(g_i(\omega_1, \dots, \omega_\ell))} + \sum_{j=1}^{n_i} 2^{-K_i \cdot z} \quad \forall 1 \leq i \leq n \end{aligned}$$

and the given constraints C

Proposition 2.6.2: If for all $1 \leq i \leq \ell$ we have $0 \leq \omega_i \leq 1$ as additional constraints then Γ has a solution with objective value h iff the same holds for Γ' .

Proof. \Rightarrow : Let $\omega_1, \dots, \omega_\ell, z$ be a solution for Γ . Then let $\omega'_i = \omega_i \cdot z$. As $0 \leq \omega_i \leq 1$ the constraints B_i in Γ' hold. Moreover $\sum_{j=1}^{n_i} 2^{-(g_i(\omega'_1, \dots, \omega'_\ell))} + \sum_{j=1}^{n_i} 2^{-K_i \cdot z} = \sum_{j=1}^{n_i} 2^{-z(g_i(\omega_1, \dots, \omega_\ell))} + \sum_{j=1}^{n_i} 2^{-K_i \cdot z} = \sum_{j=1}^{n_i} 2^{-z(f_i(\omega_1, \dots, \omega_\ell))} \leq 1$ \Leftarrow : If $\omega_1, \dots, \omega_\ell, z$ is a solution for Γ' then $\tilde{\omega}_i = \omega_i/z$, z is a solution for Γ : $\sum_{j=1}^{n_i} 2^{-z(f_i(\tilde{\omega}_1, \dots, \tilde{\omega}_\ell))} = \sum_{j=1}^{n_i} 2^{-z(g_i(\tilde{\omega}_1, \dots, \tilde{\omega}_\ell))} + \sum_{j=1}^{n_i} 2^{-K_i \cdot z} = \sum_{j=1}^{n_i} 2^{-(g_i(\omega_1, \dots, \omega_\ell))} + \sum_{j=1}^{n_i} 2^{-K_i \cdot z} \leq 1$ \square

Now in many cases the additional constraint $0 \leq \omega_i \leq 1$ naturally arises from the problem structure. Therefore this is no real limitation. See Appendix 11.2 where a convex program of the form of Γ' has been implemented. This convex program finds the optimal weights for the recurrences derived for Algorithm-5 in chapter 2.2.

Part I.

Measure & Conquer applied to **Exponential-Time- and Parameterized Algorithms**

2.6. Obtaining The Weights

Chapter 3.

A New Upper Bound for Max-2-SAT: A Graph-Theoretic Approach

3.1. Introduction

3.1.1. Our Problem

MAXSAT is an optimization version of the well-known decision problem SATISFIABILITY, or SAT for short: given a Boolean formula in conjunctive normal form (CNF), we ask for an assignment to the variables which satisfies the maximum number of clauses. Applications for MAXSAT range over such fields as combinatorial optimization, artificial intelligence and database-systems as mentioned in [102]. We put our focus on MAX-2-SAT, where every formula is constrained to have at most two literals per clause, to which problems as MAXIMUM CUT [84, 129] and MAXIMUM INDEPENDENT SET [129] are reducible. Therefore, MAX-2-SAT is \mathcal{NP} -complete.

3.1.2. Results So Far

An upper bound of $\mathcal{O}^*(2^{\frac{K}{6}})$ has been achieved by A. S. Kulikov and K. Kutzkov in [105] consuming only polynomial space. They build up their algorithm on the one of by A. S. Kulikov and K. Kutzkov [104] ($\mathcal{O}^*(2^{\frac{K}{5.88}})$) and A. Kojevnikov and A. S. Kulikov [102] who were the first who used a non-standard measure yielding a run time of $\mathcal{O}^*(2^{\frac{K}{5.5}})$. This result as well as the one obtained in this chapter has been out-dated by S. Gaspers and G. B. Sorkin [80]. That paper gives a further slight improvement for MAX-2-SAT, exhibiting an algorithm with run time $\mathcal{O}^*(2^{\frac{K}{6.321}})$. The main idea extends the present results by allowing for the creation of non-MAX-2-SAT instances by certain reduction rules. So, strictly speaking, it leaves the realm of MAX-2-SAT. Otherwise, this new paper uses the same type of analysis applied in this chapter.

If we measure the complexity in the number n of variables the current fastest and single algorithm beating the 2^n trivial upper bound is the one of R. Williams [156] having run time $\mathcal{O}^*(2^{\frac{\omega}{3}n})$, where $\omega < 2.376$ is the matrix-multiplication exponent. A drawback of this algorithm is its requirement of exponential space. A. Scott and G. B. Sorkin [143] presented a $\mathcal{O}^*(2^{(1-\frac{2}{d+1})n})$ -algorithm consuming polynomial space, where d is the average degree of the variable graph. MAX-2-SAT has also been studied with respect

to approximation (T. Hofmeister [92] and M. Lewin, D. Livnat, U. Zwick [110]) and parameterized algorithms (J. Gramm, E. A. Hirsch, R. Niedermeier, P. Rossmanith [84] and J. Gramm and R. Niedermeier [85]).

3.1.3. Our Results

The major result we present is an algorithm solving MAX-2-SAT in time $\mathcal{O}^*(2^{\frac{K}{6.2158}})$. Basically, it is a refinement of the algorithm in [102], which also in turn builds up on the results of [84]. The run time improvement is twofold. In [102] an upper bound of $\mathcal{O}^*(1.1225^n)$ is obtained if the variable graph is cubic. Here n denotes the number of variables. We could improve this to $\mathcal{O}^*(1.11199^n)$ by a more accurate analysis. Secondly, in the case where the maximum degree of the variable graph is four, we choose a variable for branching according to some heuristic priorities. These two improvements already give a run time of $\mathcal{O}^*(2^{\frac{K}{6.1489}})$. Moreover, we like to point out that these heuristic priorities can be implemented such that they only consume $\mathcal{O}(n)$ time. A. S. Kulikov and K. Kutzkov [104] improve the algorithm of A. Kojevnikov and A. S. Kulikov [102] by having a new branching strategy when the variable graph has maximum degree five. Now combining our improvements with the ones from [104] gives a run time of $\mathcal{O}^*(2^{\frac{K}{6.2158}})$.

3.1.4. Problem Statement

Finally, we formally define the problem which forms the basis of this chapter.

MAX-2-SAT

Given: A formula in CNF with only 1- and 2-clauses.

Task: Find an assignment satisfying the maximum number of clauses.

3.2. Reduction Rules & Basic Observations

We state (without proof) well-known reduction rules from previous work [84, 102]:

RR-0 Delete empty clauses.

RR-1 Replace any 2-clause C with $x, \bar{x} \in C$, for a variable x , with $\{\mathcal{T}\}$.

RR-2 If for two clauses C, D and a variable x we have $C \setminus \{x\} = D \setminus \{\bar{x}\}$, then substitute C and D by $C \setminus \{x\}$ and $\{\mathcal{T}\}$.

RR-3 If a variable x occurs only positively (negatively, resp.) then consider $F[x]$ ($F[\bar{x}]$, resp.) instead of F .

RR-4 Suppose that l is a literal and that it occurs in at least one 1-clause. If \bar{l} does not occur in more 2-clauses than l in 1-clauses, then consider $F[l]$ instead of F .

RR-5 Let x_1 and x_2 be two variables, such that x_1 appears at most once in another clause without x_2 . In this case, we call x_2 the *companion* of x_1 . **RR-3** or **RR-4** will set x_1 in $F[x_2]$ to α and in $F[\bar{x}_2]$ to β , where $\alpha, \beta \in \{\text{true}, \text{false}\}$. Depending on α and β , the following actions will be carried out:

If $\alpha = \text{false}$, $\beta = \text{false}$, then consider $F[\bar{x}_1]$ instead of F .

If $\alpha = \text{true}$, $\beta = \text{true}$, then consider $F[x_1]$ instead of F .

If $\alpha = \text{true}$, $\beta = \text{false}$, substitute every occurrence of x_1 by x_2 .

If $\alpha = \text{false}$, $\beta = \text{true}$, substitute every occurrence of x_1 by \bar{x}_2 .

For example, if $F = \{\{x\}, \{\bar{x}\}\}$, then **RR-2** would produce the new formula $F' = \{\{\}, \{\mathcal{T}\}\}$, which would be further reduced to $F'' = \{\{\mathcal{T}\}\}$ by **RR-0**. Obviously, we can consider an instance as solved if all clauses that the formula contains are of the form $\{\mathcal{T}\}$.

We introduce a new reduction rule:

RR-6: Let $e \in E(G_{var})$ such that $G_{var} - e$ contains a component Co of at most eight vertices. If C_e is the clause corresponding to e with literal $u \in C_e$ such that $u \in Co$, do the following:

1. Let S_1 be an assignment with respect to Co such that the maximum number a of clauses is satisfied under the restriction that u becomes true.

2. Let S_2 be an assignment with respect to Co such that the maximum number b of clauses is satisfied under the restriction that u becomes false.

If $b \geq a + 1$ then consider $F[S_2]$ instead of F . Otherwise consider $F[S_1]$ instead of F .

Lemma 3.2.1: **RR-6** is sound.

Proof. Let S be an assignment for F such that the maximum number z of clauses is satisfied. If u is false under S , w.l.o.g, we can assume that S restricted to Co is S_2 (as S_2 satisfies the maximum number of clauses in Co). Analogously, if u is true under S then S restricted to Co is S_1 . Note that if u becomes true then C_e is satisfied.

1. Case $b \geq a + 1$: If u becomes true under S then set the variables in Co according to S_2 . This yields an assignment S' which satisfies at least $z - (a + 1) + b \geq z$ clauses.

2. Case $b < a + 1$: If u becomes false under S then set the variables in Co according to S_1 . This yields an assignment S'' which satisfies at least $z - b + a \geq z$ clauses. \square

From now on we will only consider reduced formulas F . This means that to a given formula F we apply the following procedure: **RR-i** is always applied before **RR-(i+1)**, each reduction rule is carried out exhaustively and after **RR-6** we restart again with **RR-0** if the formula changed. For example, before we ever apply Rule **RR-4**, we know that whenever we find a literal l that occurs in a 1-clause, then there is no 1-clause that equals $\{\bar{l}\}$ due to **RR-2**. A formula for which this procedure does not restart will be called *reduced*.

Concerning the reduction rules we have the following properties:

Lemma 3.2.2 ([102] Lemma 3.1):

1. If $\#_2(v) = 1$, then v will be set.
2. For any $u \in V(F)$ in a reduced formula F we have $\#_2(u) \geq 3$.
3. If the variables a and x are neighbors and $\#_2(a) = 3$, then in at least one of the formulas $F[x]$ and $F[\bar{x}]$, the reduction rules set a .

We need some auxiliary notions:

Definition 3.2.1. A sequence of distinct vertices $a_1, v_1, \dots, v_j, a_2$ ($j \geq 0$) is called a *lasso* if $\#_2(v_i) = 2$ for $1 \leq i \leq j$, $a_1 = a_2$, $\#_2(a_1) \geq 3$ and $G_{var}[\{a_1, v_1, \dots, v_j, a_2\}]$ is a cycle.

A *quasi-lasso* is a lasso with the difference that $\#_2(v_j) = 3$. A lasso is called *3-lasso* (resp. *4-lasso*) if $\#_2(a_1) = 3$ ($\#_2(a_1) = 4$, resp.). A *3-quasi-lasso* (*4-quasi-lasso*, resp.) is a quasi-lasso with $\#_2(a_1) = 3$ ($\#_2(a_1) = 4$, resp.).

- Lemma 3.2.3:**
1. Let $v, u, z \in V(F)$ be pairwise distinct with $\#_2(v) = 3$ such that there are clauses C_1, C_2, C_3 with $u, v \in V(C_1) \cap V(C_2)$ and $v, z \in V(C_3)$. Then either v is set or the two common edges of u and v will be contracted in G_{var} by the reduction rules.
 2. The reduction rules delete the variables v_1, \dots, v_j of a lasso (quasi-lasso, resp.) and the weight of a_1 drops by at least two (one, resp.).

Proof. 1. If v is not set it will be substituted by u or \bar{u} due to **RR-5**. The emerging clauses C_1, C_2 will be reduced either by **RR-1** or become 1-clauses. Also we have an edge between u and z in G_{var} as now the variables $u, z \in V(C_3)$.

2. We give the proof by induction on j . In the lasso case for $j = 0$, there must be a 2-clause $C = \{a_1, \bar{a}_1\}$, which will be deleted by **RR-1**, so that the initial step is shown. So now $j > 0$. Then on any v_i , $1 \leq i \leq j$, we can apply **RR-5** with any neighbor as companion, so, w.l.o.g., it is applied to v_1 with a_1 as companion. **RR-5** either sets v_1 , then we are done with Lemma 3.2.2.1 (namely, since the neighbor v_1 of a_1 is set, the weight of a_1 drops by one; moreover, Lemma 3.2.2.1 sets v_2, \dots, v_j one after the other, so that finally the weight of a_1 drops again by one), or v_1 will be substituted by a_1 . By applying **RR-1**, this leads to the lasso $a_1, v_2, \dots, v_j, a_2$ in G_{var} and the claim follows by induction. In the quasi-lasso case for $j = 0$, the arguments from above hold. For $j = 1$, item 1. yields the claim. For $j > 1$, the inductive argument from the lasso case can be transferred to the quasi-lasso situation. \square

3.3. The Algorithm

The Measure We set $d_i(F) := |\{x \in V(F) \mid \#_2(x) = i\}|$. To measure the run time, we choose the measure γ defined as follows:

$$\gamma(F) = \sum_{i=3}^n \omega_i \cdot d_i(F) \text{ with } \omega_3 = 0.94165, \omega_4 = 1.80315, \omega_i = \frac{i}{2} \text{ for } i \geq 5.$$

The number of clauses K in a formula upper bounds the number of edges in the variable graph, which in turn equals half of the sum of all $d_i(F)$. Hence, $\gamma(F)$ never exceeds the number of clauses K in the corresponding formula. So, by showing an upper bound of $c^{\gamma(F)}$ we can infer an upper bound c^K . We set $\Delta_3 := \omega_3$, $\Delta_i := \omega_i - \omega_{i-1}$ for $i \geq 4$. Concerning the ω_i 's we have $\Delta_i \geq \Delta_{i+1}$ for $i \geq 3$ and $\omega_4 \geq 2 \cdot \Delta_4$. For a more detailed view on the *Measure&Conquer*-approach we refer to [74]. Note that the measure only differs from the one used in [102] by the choice of coefficients. We will see later that graphs with maximum degree three can be handled quite more efficiently than the general case. This fact is represented in the measure as the Δ_i increase with decreasing index i . We point out that no reduction rule and no branching step will ever increase γ .

Proposition 3.3.1: Let F be a MAX-2-SAT-instance. Then $\gamma(F) > \gamma(F[x])$ ($\gamma(F) > \gamma(F[\bar{x}])$, resp.) and no reduction rule increases γ .

Proof. Note that in **RR-0** - **RR-2** no 2-clause is added and thus γ cannot increase due to applying this rules. W.l.o.g. we examine only $F[x]$. Note the number of 2-clauses in $F[x]$ is no more than in F . Thus, the measure cannot increase. With this observation it follows immediately that **RR-3**, **RR-4** and **RR-6** do not increase γ . The same is also true for the first to cases of **RR-5**. In the last two cases we actually are identifying x_1 and x_2 (x_1 and \bar{x}_2 , resp.) to a new vertex d in G_{var} and remove the resulting loops immediately afterwards (either a loop is a 1-clause or **RR-1** applies). Note that $\#_2(d) \leq \#_2(x_1)$ and $\#_2(d) \leq \#_2(x_2)$ and the resulting formula F' has one variable less. Thus $\gamma(F') < \gamma(F)$. \square

The Basic Strategy The algorithm presented proceeds as follows: After applying the above-mentioned reduction rules exhaustively, it will branch on a variable v . That is, we will reduce the problem to the two formulas $F[v]$ and $F[\bar{v}]$. In each of the two branches, we must determine by how much the original formula F will be reduced in terms of $\gamma(F)$. Reduction in $\gamma(F)$ can be due to branching on a variable or to the subsequent application of reduction rules.

By an (a_1, \dots, a_ℓ) -branch, we mean that in the i -th branching case of the algorithm $\gamma(F)$ is reduced by an amount of at least a_i , i.e., the branching vector of the recurrence implied by the algorithm is (a_1, \dots, a_ℓ) . The i -th component of a branch refers to the search tree evolving from the i -th branching case (i.e., a_i). By writing $(\{a_1\}^{i_1}, \dots, \{a_\ell\}^{i_\ell})$ -branch we mean a $(a_1^1, \dots, a_1^{i_1}, \dots, a_\ell^1, \dots, a_\ell^{i_\ell})$ -branch where $a_j^s = a_j$ with $1 \leq s \leq i_j$. A (a_1, \dots, a_ℓ) -branch *dominates* a (b_1, \dots, b_ℓ) -branch if $a_i \geq b_i$ for $1 \leq i \leq \ell$, i.e., the branching number of (a_1, \dots, a_ℓ) is no greater than the one of (b_1, \dots, b_ℓ) .

3.3.1. Heuristic Priorities

Heuristic priorities guide the choice of variables to branch on by either setting them true or false. One possible priority is to prefer branching at variables of high degree (weight) in G_{var} ; further refinements of this strategy are presented below.

If the maximum degree of G_{var} is four, variables v with $\#_2(v) = 4$ will be called *limited* if there is another variable u appearing with v in two 2-clauses (i.e., we have two edges between v and u in G_{var}). We call such u, v a *limited pair*. Note that also u is limited and that at this point by **RR-5** no two weight 4 variables can appear in more than two clauses together. Any vertex which is not limited is called *unlimited*.

We call u_1, \dots, u_ℓ a *limited sequence* if $\ell \geq 3$ and u_i, u_{i+1} with $1 \leq i \leq \ell - 1$ are limited pairs. A *limited cycle* is a limited sequence with $u_1 = u_\ell$.

To obtain an asymptotically fast algorithmic behavior we introduce heuristic priorities, concerning the choice of the variable used for branching:

Heuristic Priorities (HP):

1. Choose any v with $\#_2(v) \geq 7$.
2. Choose any v with $\#_2(v) = 6$, preferably with $\#_2(N(v)) < 36$.
3. Choose any v with $\#_2(v) = 5$, preferably with $\#_2(N(v)) < 25$.
4. Choose any unlimited v with $\#_2(v) = 4$ and a limited neighbor.
5. Choose the vertex u_1 in a limited sequence or cycle u_1, \dots, u_ℓ .
6. Pick a limited pair u_1, u_2 . Let $c \in N(u_1) \setminus \{u_2\}$ with $s(c) := |N(c) \cap (N(u_1))|$ maximum. If $s(c) > 1$, then choose a vertex in $N(u_1) \setminus \{u_2, c\}$, else choose u_1 .
7. From $Y := \{v \in V(F) \mid \#_2(v) = 4, \exists z \in N(v) : \#_2(z) = 3 \wedge N[z] \not\subseteq N[v]\}$ choose v , preferably such that $\#_2(N(v))$ is maximum.
8. Choose any v , with $\#_2(v) = 4$, with $\#_2(N(v))$ minimum.
9. Choose any v , with $\#_2(v) = 3$, such that there is $a \in N(v)$, which forms a triangle a, b, c and $b, c \notin N[v]$ (we say v has *pending triangle* a, b, c).
10. Choose any v , such that we have a $(6\omega_3, 8\omega_3)$ - or a $(4\omega_3, 10\omega_3)$ -branch.

In Section 3.4.4 we will show that we can always choose $v \in V(F)$ with the claimed properties in item 10. So, our list of **HP** catches all cases. From now on v denotes the variable picked according to **HP**.

Some further comments on priorities 6 and 7 may be in order here. By definition of an open neighborhood, $s(c)$ does not count c , nor u_1 . Since at this stage, all variables, who are neighbored to a weight 4 variable, have weight 3, $s(c) \leq 2$. The two cases that are considered are either $s(c) = 2$ or $s(c) < 2$. In the first case, the chosen vertex (from $N(u_1) \setminus \{u_2, c\}$) is unique as u_1 has exactly three neighbors due to being limited.

The set Y defined in priority 7 contains weight 4 variables u with an weight 3 neighbor z such that z has a private neighbor with respect to u .

3.3.2. Key Ideas

The main idea is to have some priorities on the choice of a weight 4 variable such that the branching behavior is beneficial. For example limited variables tend to be unstable in the following sense: If their weight is decreased due to branching they will be reduced due to Lemma 3.2.3.1. This means we can get an amount of ω_4 instead of Δ_4 . In a graph lacking limited vertices we want a variable v with a weight 3 neighbor u such that $N[u] \not\subseteq N[v]$. This means that u should have a private neighbor with respect to v . In the branch on v where u is set (Lemma 3.2.2.3) we can gain some extra reduction (at least Δ_4) from $N[u] \setminus N[v]$.

If we fail to find a variable according to priorities 5-7 we show that either v as four weight 4 variables and that the graph is 4-regular, or otherwise we have two distinct situations which can be handled quite efficiently.

Further, the most critical branches are when we have to choose v such that all variables in $N[v]$ have weight ω_i . Then the reduction in $\gamma(F)$ is minimal (i.e., $\omega_i + i \cdot \Delta_i$). We analyze this regular case together with its immediate preceding branch. Thereby we prove a better branching behavior compared to a separate analysis. In [143] similar ideas were used for MAX-2-CSP.

We are now ready to present our algorithm, see Alg. 10. Reaching step 7 we can rely on the fact that G_{var} has at least 10 vertices. We call this the *small component property* (*scp*) which is crucial for some cases of the analysis. Namely, small components can be solved with an arbitrary algorithm, affecting only multiplicative constants that do not matter in the \mathcal{O}^* -notation.

Algorithm 10 An algorithm for solving MAX-2-SAT.

Procedure: SolMax2SAT(F)

- 1: Apply SolMax2SAT on every component of G_{var} separately.
 - 2: Apply the reduction rules exhaustively to F .
 - 3: Search exhaustively on any sub-formula being a component of at most 9 variables.
 - 4: **if** $F = \{\mathcal{T}\} \dots \{\mathcal{T}\}$ **then**
 - 5: **return** $|F|$
 - 6: **else**
 - 7: Choose a variable v according to **HP**.
 - 8: **return** $\max\{\text{SolMax2SAT}(F[v]), \text{SolMax2SAT}(F[\bar{v}])\}$.
 - 9: **end if**
-

3.4. The Analysis

In this section, we investigate the cases when we branch on vertices picked according to items 1-10 of **HP**. For each item we will derive a branching vector which upper bounds this case in terms of K . In the rest of section 3.4 we show:

Theorem 3.4.1: Algorithm 10 has a run time of $\mathcal{O}^*(2^{\frac{K}{6.1489}})$.

Regular Branches We call a branch *h-regular* if we branch on a variable v such that for all $u \in N[v]$ we have $\#_2(u) = h$. In a *non-regular* branch we can find a $u \in N(v)$ with $\#_2(u) < \#_2(v)$.

3.4.1. G_{var} has High Maximum Degree

3.4.1.1. Priority 1

If $\#_2(v) \geq 7$, we first obtain a reduction of ω_7 because v will be deleted. Secondly, we get an amount of at least $7 \cdot \Delta_7$ as the weights of v 's neighbors each drops by at least one and we have $\Delta_i \geq \Delta_{i+1}$. Thus, γ is reduced by at least 7 in either of the two branches (i.e., we have a $(\{7\}^2)$ -branch).

The following Priorities are analyzed in a way that we assume that they are non-regular branches. We dedicate an extra section to regular branches. Note that we already handled *h-regular* branches for $h \geq 7$ above.

3.4.1.2. Priorities 2 and 3

Choosing $v \in V(F)$ with $\#_2(v) = 6$, there is a $u \in N(v)$ with $\#_2(u) \leq 5$ due to non-regularity. Then by deletion of v , there is a reduction by ω_6 and another of at least $5\Delta_6 + \Delta_5$, resulting from the dropping weights of the neighbors. Especially, the weight of u must drop by at least Δ_5 . This leads to a $(\{6.19685\}^2)$ -branch. If $\#_2(v) = 5$, the same observations as in the last choice lead to a reduction of at least $\omega_5 + 4 \cdot \Delta_5 + \Delta_4$. Thus we have a $(\{6.1489\}^2)$ -branch.

3.4.2. G_{var} has Maximum Degree Four

We often will stress the fact that in a reduced formula for every $v \in V(F)$ we have $\#(v) \geq 3$ due to Lemma 3.2.2.2. Any variable violating this property will be set or replaced by another variable (see **RR-5**).

3.4.2.1. Priority 4

Let $u_1 \in N(v)$ be the limited variable. The vertex u_1 forms a limited pair with some u_2 , see Figure 3.1(a).

After branching on v , the variable u_1 has weight at most 3. At this point, u_1 appears only with one other variable z in a 2-clause. Then, **RR-5** is applicable to u_1 with u_2 as its companion. According to Lemma 3.2.3.1, either u_1 is set or the two edges of u_1 and u_2 will be contracted. In the first case, we receive a total reduction of at least $3\omega_4 + 2\Delta_4$, in the second of at least $2\omega_4 + 4\Delta_4$ (even if $d_i = h$, for some $i \in \{1, 2, 3\}$, since $\omega_4 \geq 2\Delta_4$). A proper estimate is a $(\{2\omega_4 + 4\Delta_4\}^2)$ -branch, i.e., a $(\{7.0523\}^2)$ -branch.

3.4.2.2. Priority 5

If u_1, \dots, u_ℓ is a limited cycle, then $\ell \geq 10$ due to *scp*. By **RR-5** this yields a $(10\omega_4, 10\omega_4)$ -branch. If u_1, \dots, u_ℓ is a limited sequence, then due to Priority 4 the neighbors of u_1 lying outside the sequence have weight 3. By **RR-5**, the branch on u_1 is a $(\{3\omega_4 + 2\omega_3\}^2)$ -branch, i.e., a $(\{7.29275\}^2)$ -branch.

3.4.2.3. Priority 6

At this point every limited variable u_1 has two neighboring variables y, z with weight 3 and a limited neighbor u_2 with the same properties (due to priorities 4 and 5). We now examine the local structures arising from this fact and by the values of $|N[y] \setminus N[u_1]|$ and $|N[z] \setminus N[u_1]|$. W.l.o.g., we only discuss the cases when $|N[y] \setminus N[u_1]| \leq |N[z] \setminus N[u_1]|$.

1. We rule out $|N[y] \setminus N[u_1]| = |N[z] \setminus N[u_1]| = 0$ (see Figure 3.1(b)) due to *scp*.
2. $|N[y] \setminus N[u_1]| = 0, |N[z] \setminus N[u_1]| = 1$: Then, $N(y) = \{u_2, z, u_1\}, N(u_2) = \{u_1, y, s_1\}$ and $N(z) = \{u_1, y, s_2\}$, see Figure 3.1(c). In this case we branch on z , as z is the only vertex that is neighbor of u_1 but not of u_2 . Thus, $s(y) = 2 > s(z) = 1$. Then due to **RR-5** y and u_1 disappear; either by being set or replaced. Thereafter due to **RR-1** and Lemma 3.2.2.1 u_2 will be set. Additionally we get an amount of $\min\{2\Delta_4, \omega_4, \omega_3 + \Delta_4\}$ from s_1, s_2 . This depends on whether $s_1 \neq s_2$ or $s_1 = s_2$ and in the second case on the weight of s_1 . If $\#_2(s_1) = 3$ we get a reduction of $\omega_3 + \Delta_4$ due to setting s_1 and *scp*. In total we have at least a $(\{2\omega_4 + 2\omega_3 + 2\Delta_4\}^2)$ -branch.
3. $|N[y] \setminus N[u_1]| = 1, |N[z] \setminus N[u_1]| = 1$: Here two possibilities occur, depending on the neighbor of u_1 present in $N(y)$: (a) $N(y) = \{u_1, u_2, s_1\}, N(z) = \{u_1, u_2, s_2\}, N(u_2) = \{u_1, y, z\}$, see Figure 3.1(d): Then w.l.o.g., we branch on z . After setting z the vertices u_1, u_2 and y will disappear either by being set or replaced by another variable due to **RR-5** or Lemma 3.2.2.2. Similarly to item 2. we obtain a $(\{2\omega_4 + 2\omega_3 + 2\Delta_4\}^2)$ -branch.
(b) $N(y) = \{u_1, z, s_1\}, N(z) = \{u_1, y, s_2\}$, see Figure 3.1(e): W.l.o.g., we branch on z . Basically we get a total reduction of $\omega_4 + 2\omega_3 + 2\Delta_4$. That is $2\omega_3$ from y and z , ω_4 from u_1 and $2\Delta_4$ from s_2 and u_2 . In the branch where y is set (Lemma 3.2.2.3), we additionally get Δ_4 from s_1 and ω_4 from u_2 as it will disappear (Lemma 3.2.2.2). This is a $(2\omega_4 + 2\omega_3 + 2\Delta_4, \omega_4 + 2\omega_3 + 2\Delta_4)$ -branch.
4. $|N[y] \setminus N[u_1]| = 1, |N[z] \setminus N[u_1]| = 2$, see Figure 3.1(f): We branch on z and due to **RR-5** the variables u_1, u_2 and y will disappear. This yields a $(\{2\omega_4 + 2\omega_3 + 2\Delta_4\}^2)$ -branch.
5. $|N[y] \setminus N[u_1]| = 2, |N[z] \setminus N[u_1]| = 2$, see Figure 3.1(g): In this case we choose u_1 for branching. Essentially, we get a reduction of $2\omega_4 + 2\omega_3$. In the branch setting z we receive an extra amount of $2\Delta_4$ from z 's two neighbors outside $N(u_1)$. Hence, we have a $(2\omega_4 + 2\omega_3 + 2\Delta_4, 2\omega_4 + 2\omega_3)$ -branch.

We have at worst a $(2\omega_4 + 2\omega_3 + 2\Delta_4, \omega_4 + 2\omega_3 + 2\Delta_4)$ -branch, i.e., a $(7.2126, 5.40945)$ -branch.

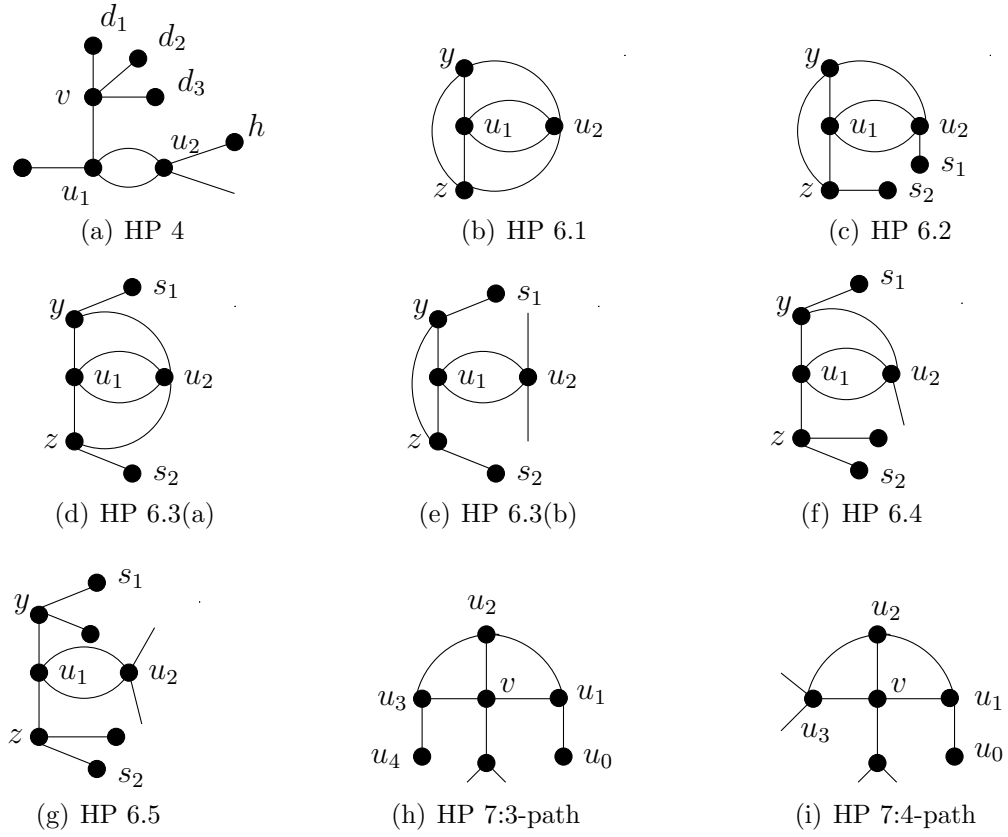


Figure 3.1.: Local structures that arise when branching in a variable multi-graph of maximum weight four. These structures appear in heuristic priorities 4, 6, and 7 as indicated.

3.4.2.4. Priority 7

We need further auxiliary notions:

Definition 3.4.1. A *3-path* for an unlimited weight 4 vertex v is a sequence of vertices $u_0u_1 \dots u_lu_{l+1}$ forming a path, such that $1 \leq l \leq 4$, $u_i \in N(v)$ for $1 \leq i \leq l$, $\#_2(u_i) = 3$ for $1 \leq i \leq l$ and $u_0, u_{l+1} \notin N(v)$, see Figure 3.1(h) for the case $l = 3$. A *4-path* for an unlimited weight 4 vertex v is a sequence of vertices $u_0u_1 \dots u_l$ forming a path, such that $2 \leq l \leq 4$, $\#_2(u_i) = 3$ for $1 \leq i \leq l - 1$, $\#_2(u_l) = 4$ and $u_0 \notin N(v)$, see Figure 3.1(i) for the case $l = 3$.

Due to the absence of limited vertices, every vertex v , chosen due to Priority 7, must have a 3- or 4-path.

3-path

$u_0 \neq u_{\ell+1}$ If $u_0 \neq u_{\ell+1}$ we basically get a reduction of $\omega_4 + l\omega_3 + (4-l)\Delta_4$. In the branch where u_1 is set, $u_2 \dots u_\ell$ will be also set due to Lemma 3.2.2.1. Therefore, we gain an extra amount of at least $2\Delta_4$ from u_0 and $u_{\ell+1}$, leading to a $(\omega_4 + l\omega_3 + (6-l)\Delta_4, \omega_4 + l\omega_3 + (4-l)\Delta_4)$ -branch.

$u_0 = u_{\ell+1}$ In $F[v]$ and in $F[\bar{v}]$, $u_0 u_1 \dots u_\ell u_{\ell+1}$ is a lasso such that $l \geq 2$ (**RR-5**). So by Lemma 3.2.3.2, u_1, \dots, u_ℓ are deleted and the weight of u_0 drops by 2. If $\#_2(u_0) = 4$ this yields a reduction of $l\omega_3 + \omega_4$. If $\#_2(u_0) = 3$ the reduction is $(l+1)\omega_3$ but then u_0 is set. If $N[u_0] \setminus N[v]$ is not empty then we obtain a reduction of Δ_4 in addition due to setting u_0 . Otherwise there is a unique $r \in N(u_0) \setminus \{u_1, \dots, u_\ell\}$ with $r \in N(v) \setminus \{u_1, \dots, u_\ell\}$. If $\#_2(r) = 4$ we get a $(\{2\omega_4 + (l+1)\omega_3 + (3-l)\Delta_4\}^2)$ -branch. If $\#_2(r) = 3$, then r is set. As $(4-l) \leq 2$ and by applying the same arguments to r which previously were applied to u_0 we get at least a $(\{\omega_4 + (l+1)\omega_3 + (5-l)\Delta_4\}^2)$ -branch. Observe that we used the fact that $\omega_4 \geq 2\Delta_4$.

4-path We get an amount of $\omega_4 + (l-1)\omega_3 + (5-l)\Delta_4$ by deleting v . In the branch where u_1 is set we get a bonus of Δ_4 from u_0 due to the decreasing degree of u_0 in G_{var} . Further u_ℓ will be deleted completely. Hence we have a $(2\omega_4 + (l-1)\omega_3 + (5-l)\Delta_4, \omega_4 + (l-1)\omega_3 + (5-l)\Delta_4)$ -branch.

The first branch is worst for $l = 1$, the second and third for $l = 2$ (as $l = 1$ is impossible by definition). Thus, we have a $(\{7.2126\}^2)$ -branch for the second and a $(7.0523, 5.3293)$ -branch for the first and third case which is sharp, i.e., the branching number for the branching vector $(7.0523, 5.3293)$ is $2^{\frac{\kappa}{6.1489}}$.

3.4.2.5. Priority 8

Lemma 3.4.2: If we have chosen a variable v with $\#_2(v) = 4$ according to Priority 8, such that $\#_2(N(v)) < 16$, then we have two distinct situations, see Figures 3.2(a) and 3.2(b).

Proof. Note that when we are forced to pick a variable v according to Priority 8, then either v has four neighbors of weight 4 or for every weight 3 neighbor z we have $N[z] \subseteq N[v]$. Namely, if $\#_2(N(v)) < 16$, then v has a neighbor of weight 3, since neighbors of lower weight would have been set or replaced: notice that the set Y defined in **HP 7** must be empty. It follows that, for every weight 3 neighbor z , we have $N[z] \subseteq N[v]$ due to the choice of v according to **HP**.

Let N^4 (resp. N^3) be the set of weight 4 (3, resp.) neighbors of v . We analyze different cases induced by the cardinality $|N^3|$.

— If $N^3 = \{b\}$, then there are vertices $a, c \in N^4$, such that $b \in N(a)$ and $b \in N(c)$. We must have $a \in N(c)$, or else a would violate our assumption: it would be a variable with $\#_2(a) = 4$ and it would have a neighbor, namely b , with $\#_2(b) = 3$ and $N(b) \not\subseteq N(a)$. Thus, we get the situation of Figure 3.2(a).

— Consider $N^3 = \{b, c\}$. Case (1): b and c are neighbors. If $b, c \in N(a)$ for $a \in N^4$, we have a situation as depicted in Figure 3.2(b). Otherwise, $b \in N(a)$ and $c \in N(d)$

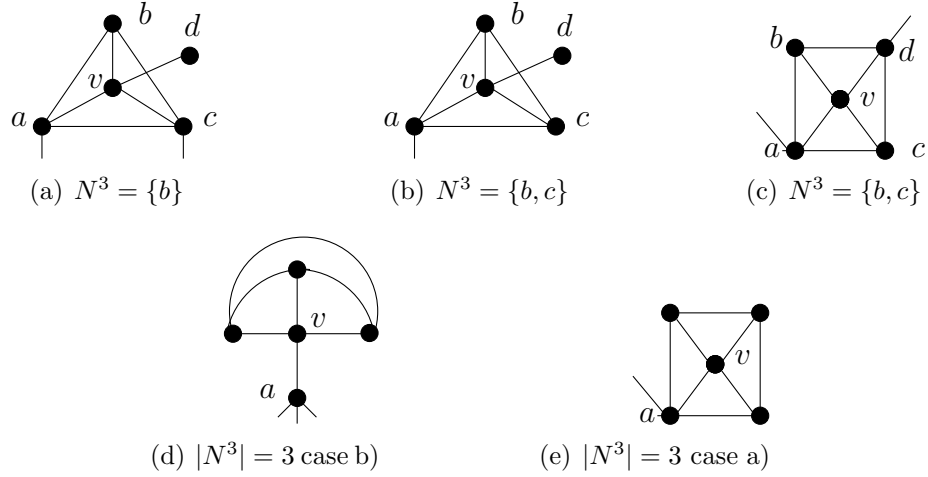


Figure 3.2.: Local structures that arise when branching in a variable multi-graph of maximum weight four, ctd.

for $a, d \in N^4$. But then, Priority 7 applies to both a and d , which is a contradiction to the fact that Alg. 10 is already in Priority 8. Case (2): b and c are not neighbors, it can be easily observed that we must have the situation in Figure 3.2(c), where Priority 7 applies to a and d , also a contradiction.

— *a*) If $|N^3| = 3$ and no vertex from N^3 is neighbored to the weight 4 vertex $a \in N(v) \setminus N^3$ then the situation in Figure 3.2(d) emerges. It is easily seen that **RR-6** applies to this case, a contradiction

b) If there is a vertex from N^3 neighboring a then it is easy to verify that we must have situation 3.2(e) in Figure 3.1. But then Priority 7 applies to a .

If $|N^3| = 4$, then clearly $N[v]$ forms a component of five vertices which cannot appear after step 3 of Alg. 10. \square

In Figure 3.2(a) in either branch $F[v]$ or $F[\bar{v}]$, the variables a, b, c form a 3-quasi-lasso, so by Lemma 3.2.3.2 we get a reduction of $\omega_3 + 3\omega_4 + \Delta_4 = 4\omega_4$.

In Figure 3.2(b) in both branches the variables a, b, c form a 3-lasso, so by Lemma 3.2.3.2 b, c are deleted and a is set due to Lemma 3.2.2.1. We get a reduction of $\omega_4 + 2\omega_3$ from this. If $d \notin N(a)$ we additionally get $2\Delta_4$, otherwise ω_4 . Altogether, we reduce $\gamma(F)$ by at least $2\omega_4 + 2\omega_3 + 2\Delta_4$ in each of the both branching cases.

3.4.3. The 4- 5- and 6-regular case

The part of the algorithm when we branch on variables of weight $h \neq 4$ will be called *h-phase*. Branching according to priorities 4-8 is the *4-phase*, according to priorities 9 and 10 the *3-phase*. In the following we have $4 \leq h \leq 6$. The case $h \geq 7$ has been already covered by Priority 1. Later on also a special section is devoted to the case $h = 3$. Any *h-regular* branch which was preceded by a branch from the $(h + 1)$ -phase can be neglected. This situation can only occur once on each path from the root to a leaf in the search tree. Hence, the run time is only affected by a constant multiple.

We now classify h -regular branches: An *internal h -regular branch* is a h -regular branch such that another h -regular branch immediately follows in the search tree in at least one component. A *final h -regular branch* is a h -regular branch such that no h -regular branch immediately succeeds in either of the components. When we are forced to do an h -regular branch, then according to **HP** the whole graph must be h -regular at this point.

Observation 3.4.3: If a branch is followed by a h -regular branch in one component, say in $F[v]$, then in $F[v]$ any $u \in V(F)$ with $\#_2(u) < h$ will be reduced.

Due to Observation 3.4.3 every vertex in $N(v)$ is completely deleted by the reduction rules in $F[v]$.

3.4.3.1. Internal h -regular branches

Proposition 3.4.4: $\mathcal{O}^*(1.1088^K)$ upper bounds any internal h -regular branch.

Proof. By Observation 3.4.3 for $h = 4$ this yields at least a $(5\omega_4, \omega_4 + 4\Delta_4)$ -branch if in exactly one component a h regular branch follows. The vertex v must have been chosen due to Priority 8. Thus, v has 4 different weight 4 neighbors. If both components are followed by an h -regular branch we get a total reduction of $5\omega_4$ in both cases. The same way we can analyze internal 5- and 6-regular branches. This yields $(3\omega_5, \omega_5 + 5\Delta_5)$ -, $(\{3\omega_5\}^2)$ -, $(3\omega_6, \omega_6 + 6\Delta_6)$ - and $(\{3\omega_6\}^2)$ -branches as for any $v \in V(F)$ we have $|N(v)| \geq 2$. Note that we can have multiple edges as v is chosen due to **HP** 2 or 3. \square

3.4.3.2. Final h -regular branches

We will consider branches which are immediately followed by a h -regular branch in at least one component. In this component of the branch we can delete any variable in $N(v)$ additionally due to Observation 3.4.3. The Propositions 3.4.5 and 3.4.6 will explore by how much we additionally can decrement $\gamma(F)$ in the corresponding component in case $h \in \{5, 6\}$. Let k_{ij} denote the number of weight j variables occurring i times in a 2-clause with some $v \in V(F)$ chosen for branching. Observe that it is impossible that $|j - i| \leq 1$ due to **RR-5**.

Proposition 3.4.5: Let $v \in V(F)$ be the variable chosen due to **HP** such that $\#_2(v) = 5$. If this branch is followed by a 5-regular branch in one component, then we can decrement $\gamma(F)$ by at least $\omega_5 + \omega_4$ in addition to the weight of v in that component.

Proof. According to [104] we must have the following relation:

$$k_{13} + k_{14} + k_{15} + 2k_{24} + 2k_{25} + 3k_{35} = 5 \quad (3.1)$$

We now have to determine an integer solution to (3.1) such that $\omega_3 k_{13} + \omega_4 k_{14} + \omega_5 k_{15} + \omega_4 k_{24} + \omega_5 k_{25} + \omega_5 k_{35}$ is minimal. We can assume $k_{14} = k_{15} = 0$ as we have $\omega_3 < \omega_4 < \omega_5$. For any solution violating this property we can find a smaller solution by setting $k'_{13} = k_{13} + k_{14} + k_{15}$, $k'_{14} = 0$ and $k'_{15} = 0$ and keeping the other coefficients. The same

way we find that $k_{25} = 0$ must be the case as $\omega_4 < \omega_5$.

If $k_{13} \geq 2$ we set $k'_{13} = k_{13} - 2\lfloor \frac{k_{13}}{2} \rfloor$, $k'_{24} = k_{24} + \lfloor \frac{k_{13}}{2} \rfloor$ and keep the other coefficients. By $2\omega_3 > \omega_4$ this is a smaller solution. Now suppose $k_{13} = 1$, then we have $k_{24} = 0$ in a minimal solution as $\omega_3 + \omega_4 > \omega_5$ (i.e., if $k_{24} \geq 1$ we set $k'_{13} = 0$, $k'_{24} = k_{24} - 1$ and $k'_{35} = k_{35} + 1$). But then no k_{35} could satisfy (3.1). Thus, we have $k_{13} = 0$. Then the only integer solution is $k_{24} = 1$ and $k_{35} = 1$. Hence, the minimal reduction we get from $N(v)$ is $\omega_5 + \omega_4$. \square

Proposition 3.4.6: Let $v \in V(F)$ be the variable chosen due to **HP** such that $\#_2(v) = 6$. If this branch is followed by a 6-regular branch in one component, then we can decrement $\gamma(F)$ by at least $\omega_6 + \omega_4$ in addition to the weight of v in that component.

Proof. In this case the following relation holds:

$$k_{13} + k_{14} + k_{15} + k_{16} + 2k_{24} + 2k_{25} + 2k_{26} + 3k_{35} + 3k_{36} + 4k_{46} = 6 \quad (3.2)$$

We now have to determine an integer solution to (3.2) such that $\omega_3 k_{13} + \omega_4 k_{14} + \omega_5 k_{15} + \omega_6 k_{16} + \omega_4 k_{24} + \omega_5 k_{25} + \omega_6 k_{26} + \omega_5 k_{35} + \omega_6 k_{36} + \omega_6 k_{46}$ is minimal. As $\omega_3 < \omega_4 < \omega_5 < \omega_6$ we conclude that $k_{1\ell} = 0$ for $4 \leq \ell \leq 6$, $k_{2\ell} = 0$ for $5 \leq \ell \leq 6$ and $k_{36} = 0$. We also must have $k_{13} \leq 1$ as in the previous proof. By $2\omega_4 > \omega_6$ and the same arguments we must have $k_{24} \leq 1$. By (3.2) we also have $k_{35} \leq 2$ and $k_{46} \leq 1$.

If $k_{13} = 0$ the only integer solutions under the given restrictions are $k_{35} = 2$ and $k_{24} = 1, k_{46} = 1$. If $k_{13} = 1$ the only integer solution is $k_{35} = 1, k_{24} = 1$. Thus, the minimal amount we get by reduction from $N(v)$ is $\omega_6 + \omega_4$ due to $\omega_6 + \omega_4 < 2\omega_5$ and $\omega_6 + \omega_4 < \omega_3 + \omega_5 + \omega_4$. \square

We now analyze a final h -regular ($\{b\}^2$)-branch with its preceding (a_1, a_2) -branch. The final h -regular branch might follow in the first, the second or both components of the (a_1, a_2) -branch. So, the *combined analysis* would be a $(\{a_1 + b\}^2, a_2)$, a $(a_1, \{a_2 + b\}^2)$ - and a $(\{a_1 + b\}^2, \{a_2 + b\}^2)$ -branch. For any final h -regular we will apply a combined analysis with its preceding branch.

Proposition 3.4.7: Any final h -regular branch ($h \in \{5, 6\}$) considered together with its preceding branch can be upper bounded by $\mathcal{O}^*(1.1172^K)$.

Proof. We will apply a combined analysis for both branches. Due to Observation 3.4.3 $N(v)$ will be deleted in the corresponding component of the preceding branch. The least amount we can get by deleting $N(v)$ is $\omega_5 + \omega_4$ in case $h = 5$ and $\omega_6 + \omega_4$ in case $h = 6$ (due to Propositions 3.4.5 and 3.4.6). Hence, we get four different branches: A $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^2, \omega_5 + 5\Delta_5)$ -, a $(\{3\omega_6 + \omega_4 + 6\Delta_6\}^2, \omega_6 + 6\Delta_6)$ -, a $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^4)$ - and a $(\{3\omega_6 + \omega_4 + 6\Delta_6\}^4)$ -branch, respectively. \square

Proposition 3.4.8: Any final 4-regular branch considered with its preceding branch can be upper bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}}) \approx \mathcal{O}^*(1.11933^K)$.

Proof. We must analyze a final 4-regular branch together with any possible predecessor. These are all branches derived from priorities 4-8.

Internal 4-regular branch The two corresponding branches are a $(\{6\omega_4 + 4\Delta_4\}^2, \omega_4 + 4\Delta_4)$ -branch and a $(\{6\omega_4 + 4\Delta_4\}^4)$ -branch.

Priorities 4, 5 and 8 are all dominated by a $(\{2\omega_4 + 4\Delta_4\}^2)$ -branch. Analyzing these cases together with a succeeding final 4-regular branch gives a $(\{3\omega_4 + 8\Delta_4\}^2, 2\omega_4 + 4\Delta_4)$ -branch and a $(\{3\omega_4 + 8\Delta_4\}^4)$ -branch.

Priority 6 Subcases 2, 3(a) and 4 of our non-regular priority-6 analysis can be analyzed similarly to priorities 4, 5 and 8 as they have branching vectors which dominate $(\{2\omega_4 + 4\Delta_4\})$. We now analyze the remaining subcases.

Subcase 1 Here we deal with small components which are directly solved without any branching. Therefore we get a $(\{3\omega_4 + 2\omega_3 + 4\Delta_4\}^2)$ -branch in the combined analysis. Consider now cases 3(b) and 5. Let u_1, u_2 be the chosen limited pair. Due to **HP**, the variable u_2 has two weight 3 neighbors p_1 and p_2 . Thus, if a final 4-regular branch is following in these cases, then we get an additional reduction of $2\omega_3$ (with respect to the component of the branch). This means that $N[\{u_1, u_2\}]$ will be reduced. For both cases we derived a non-symmetric branch, e.g., an (a, b) -branch with $a \neq b$. Depending on whether the final 4-regular branch follows in the first, the second or both components we derive three combined branches: a) $(\{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2, 2\omega_3 + \omega_4 + 2\Delta_4)$, b) $(2\omega_3 + 2\omega_4 + 2\Delta_4, \{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$ - and c) $(\{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$. As $\mathcal{O}^*(2^{\frac{K}{6.1489}})$ does not properly upper bound a) we need a further discussion. Thus we will consider two subcases. Remember that in the first component of a) some weight 3 neighbor $t \in \{z, y\}$ of v is set where $v \in \{z, y\} \setminus \{t\}$

Subcase 3(b).1. First suppose that $N(z) \setminus (N(u_1) \cup N(u_2)) = \emptyset$ and $N(y) \setminus (N(u_1) \cup N(u_2)) = \emptyset$, see Figure 3.3(a). Then by either branching on y or z we get a $(\{2\omega_4 + 4\omega_3\}^2)$ -branch. Note that all the vertices in Figure 3.3(a) will disappear due to the reduction rules. In this case the combined analysis is similar to priorities 4, 5 and 8.

Subcase 3(b).2. Secondly, w.l.o.g., we have $N(z) \setminus (N(u_1) \cup N(u_2)) \neq \emptyset$, see Figure 3.3(b) and 3.3(c). In Figure 3.3(b) we might have picked $y = v$ or $z = v$ for branching. But observe that in both cases in the branch where the particular weight 3 neighbor t is set ($t = s$ if $v = z$ and $t = z$ if $v = y$) such that in this component a 4-regular branch follows we have at least a reduction of $2\omega_4 + 5\omega_3$. This entails a $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + 2\omega_3 + 2\Delta_4)$ -branch (called a') in the combined analysis instead of a). If the case in Figure 3.3(c) applies, then, w.l.o.g., we branch on z and we have $t = s$. Then in the branch where s is set, s, y and u_1 will be reduced due to **RR-5** and $N[u_2] \setminus \{u_1\}$ due to the fact that a 4-regular branch follows. Thus, the derived branch is the same as for the case of Figure 3.3(b).

Subcase 5 As the vertices in $N(u_1) \cup N(u_2)$ cannot form a component, w.l.o.g., we find a variable $q \in N(z) \setminus (N(u_1) \cup N(u_2))$. In this case we branch on u_1 . Now in the branch where we set z (i.e., $z = t$) such that a 4-regular branch follows in that component we get at least ω_3 from q in addition to $2\omega_4 + 4\omega_3$ from $N(u_1) \cup N(u_2)$. We have a $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, 2\omega_4 + 2\omega_3)$ -branch (called a'') in the combined analysis instead of

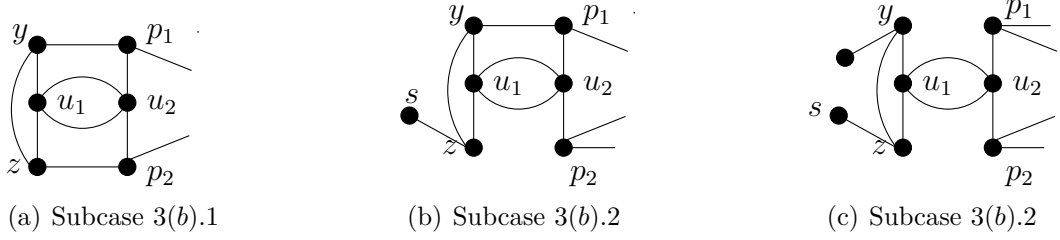


Figure 3.3.: Situations arising in Priority 6

$\#_2(u_0), \#_2(u_{l+1})$	left component	right component	both components
$\#_2(u_0) = 3$ $\#_2(u_{l+1}) = 3$	$(\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2,$ $\omega_4 + 4\omega_3)$	$(\omega_4 + 6\omega_3,$ $\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2)$	$(\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2)$
$\#_2(u_0) = 3$ $\#_2(u_{l+1}) = 4$	$(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2,$ $\omega_4 + 4\omega_3)$	$(\omega_4 + 5\omega_3 + \Delta_4,$ $\{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$	$(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$
$\#_2(u_0) = 4$ $\#_2(u_{l+1}) = 4$	$(\{4\omega_4 + 4\omega_3 + 4\Delta_4\}^2,$ $\omega_4 + 4\omega_3)$	$(\omega_4 + 4\omega_3 + 2\Delta_4,$ $\{2\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$	$(\{4\omega_4 + 4\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$

Table 3.1.:

a).

Both branches replacing a) have an upper bound of $\mathcal{O}^*(2^{\frac{K}{6.1489}})$.

Priority 7 Let o be the number of weight 4 vertices from $N(v)$. If in one component a final 4-regular branch follows, then the worst case is when $o = 0$ as any weight 4 vertex would be deleted and $\omega_4 > \omega_3$. On the other hand, if there is a component without an immediate 4-regular branch succeeding, then the worst case appears when o is maximal (i.e., $o = 4$) as $\omega_3 \geq \Delta_4$. So in the analysis we will consider for each case the particular worst case even though both together never appear.

3-path with $u_0 \neq u_{l+1}$: First if there is a weight 4 variable in $N(v)$ we have at worst the following branches: a) $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$, b) $(\omega_4 + \omega_3 + 5\Delta_4, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ and c) $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$. Any of those is upper-bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$. Now suppose for all $y \in N(v)$ we have $\#_2(y) = 3$. Table 3.1 captures the derived branches for certain combinations. Here we will also consider the weights of u_0 and u_l . Any entry is upper bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$ except $\alpha) (\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2, \omega_4 + 4\omega_3)$ the left upper entry and $\beta) (\omega_4 + 4\omega_3 + 2\Delta_4, \{2\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$ the middle entry of the last row.

For $U \subseteq V(F)$ we define $E_3(U) := \{\{u, z\} \mid u \in U, \#_2(u) = 3, z \notin U\}$. The set $E_3(U)$ contains the edges with one endpoint in U , such that this endpoint has weight 3, and the other endpoint is outside U . The next claim shows that we can do better in cases α and β .

Claim: 1. Suppose for all $y \in Q := N(v) \cup \{u_0, u_{l+1}\}$ we have $\#_2(y) = 3$. Then there must be some $y' \in V \setminus (N(v) \cup \{u_0, u_{l+1}\})$ with $\#_2(y') = 3$.

2. Suppose for all $y \in N(v)$ we have $\#_2(y) = 3$ and $\#_2(u_0) = \#_2(u_{l+1}) = 4$. Then

there must be some $y' \in V \setminus (N(v) \cup \{u_0, u_{l+1}\})$ with $\#_2(y') = 3$.

Proof. 1. Assume the contrary. For any $1 \leq l \leq 4$ (the length of the 3-path), we have $|E_3(Q \cup \{v\})| \leq 10$ which is sharp for $l = 1$. Due to *scp*, there is a weight 4 vertex r adjacent to some vertex in Q . Observe that we must have $r \in Y$ (where Y is defined in **HP** 7) as either there is $u \in N(v)$ with $u \in N(r)$ (and clearly $v \notin N(r)$ as $N(v)$ consists of weight 3 variables) or, w.l.o.g., $u_0 \in N(r)$ but $u_1 \notin N(r)$. Furthermore, r has 4 weight 3 neighbors from Q due to the choice of v according to **HP** and our assumption. Hence we must have $|E_3(Q \cup \{v, r\})| \leq 6$ due to r being incident to four vertices from Q . Using the same arguments again we find some $r' \in Y$ with $|E_3(Q \cup \{v, r, r'\})| \leq 2$. Again, due to *scp*, we find a $r'' \in Y$ with 4 weight 3 neighbors where at most two are from Q , a contradiction.

2. Assume the contrary. Observe that $u_0, u_{l+1} \in Y$ (u_1 and u_l have v as private neighbor and $u_1 \in N(u_0)$, $u_l \in N(u_{l+1})$) and due to the choice of v both have 4 weight 3 neighbors which must be from $N(v)$. From $|E_3(N[v])| \leq 8$, it follows that $|E_3(N[v] \cup \{u_0, u_{l+1}\})| = 0$ which contradicts *scp*. \square

Due to the last claim and Observation 3.4.3 we have a $(\{2\omega_4 + 7\omega_3 + 4\Delta_4\}^2, \omega_4 + 4\omega_3)$ -branch (α') instead of case α) and a $(\omega_4 + 4\omega_3 + 2\Delta_4, \{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$ -branch (β') instead of case β). Both are upper-bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$.

3-path with $u_0 = u_{l+1}$: In the case of a 3-path such that $u_0 = u_{l+1}$, the branch with $l = 2$ is dominated by all other choices. Since this is a $(\{7.21\}^2)$ -branch we refer to priorities 4, 5 and 8 from above.

4-path: In this case, we have the following worst-case branches for $l = 2$: a) $(\{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$, b) $(2\omega_4 + \omega_3 + 3\Delta_4, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$, c) $(\{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$. The cases a) and c) are not upper bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$ and hence need further discussion.

Suppose there is a vertex $y \in D := N(v) \cup \{u_0, \dots, u_{l-1}\}$ with weight 4. Then by Observation 3.4.3 we have branches a') $(\{4\omega_4 + 3\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$ and c') $(\{4\omega_4 + 3\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ instead of a) and c) which are both upper-bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$. For the remaining case we need the next proposition.

Claim: Suppose for all $y \in D$ we have $\#_2(y) = 3$. Then there must be some $y' \in V \setminus (D \cup \{v, u_l\})$ with $\#_2(y') = 3$.

Proof. Assume the contrary. Observe that if $l \geq 3$ then $u_l \in Y$ due to $u_{l-2} \notin N(u_l)$. If $l = 2$ and $u_0 \notin N(u_2)$ then also $u_l \in Y$ holds. Let us assume that $l \geq 2$ and $u_l \in Y$ as the remaining case, $l = 2$ and $u_l \notin Y$, will be treated separately.

Now due to the choice of v we have that u_l must be adjacent to v , u_{l-1} and to two further weight 3 vertices in D . Thus, for any $2 \leq l \leq 4$ we always have $|E_3(D \cup \{v, u_l\})| \leq 8 - (2(l-1) + 2) = 8 - 2l$ (\star). Therefore and as $D \cup \{v, u_l\}$ can not be a component we have $l < 4$. There must some weight 4 vertex $r \notin D \cup \{v, u_l\}$ adjacent to some weight 3 vertex $b \in D$ as we have no small components and u_l only has v as weight 4 neighbor. Note that $r \in Y$, as either $b \neq u_0$ and $v \notin N(r)$, or $b = u_0$ but $u_1 \notin N(r)$.

Due to the choice of v , r must have at least three weight 3 neighbors. Hence $l = 2$ due to (\star) . If r has 4 weight 3 neighbors then $(D \cup \{v, u_l, r\})$ forms a component which is a contradiction. Hence, we have $|E_3(D \cup \{v, u_l, r\})| = 1$ and therefore we find again some $r' \in Y \setminus (D \cup \{v, u_l, r\})$ which is adjacent to at least 3 weight 3 vertices where at most one is from D . Thus, there must be some weight 3 vertex in $V \setminus (D \cup \{v, u_l\})$, a contradiction.

Now suppose $l = 2$ and $u_0 \in N(u_2)$ and let $N(u_0) = \{z, u_1, u_2\}$. If $z \notin N(u_2)$ then $u_2 \in Y$ and the first part of the proof applies. Now suppose $z \in N(u_2)$. Then $\#_2(z) = 3$ and it follows that $z \in N(v)$ and $|E_3(\{D \cup \{v, u_2\}\})| \leq 2$. Now due to *scp* we can find an $r \in Y \setminus (D \cup \{v, u_l\})$ which is adjacent to at least three weight 3 vertices where only two can be from $D \cup \{v, u_l\}$, a contradiction. \square

If for all $y \in D$ we have $\#_2(y) = 3$ from the last claim and Observation 3.4.3 we can derive two branches a'') ($\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4$) and c'') ($\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2$) which are upper-bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$. \square

3.4.4. The Cubic Case

3.4.4.1. Priority 9

Observe that when we have arrived at this point, the graph G_{var} must be 3-regular and each variable has three different neighbors, due to G_{var} being reduced and due to Lemma 3.2.2.2. Also, any 3-regular graph has an even number of vertices, because we have $3n = 2m$. Thus: $(*)$ Any branching must be of the form $(2i\omega_3, 2j\omega_3)$ for some $1 \leq i, j$. Also, branching on any variable will at least result in a $(4\omega_3, 4\omega_3)$ -branch (see Lemma 3.2.2.2). Note that any $u \in N(v)$ will be either set in $F[v]$ or in $F[\bar{v}]$, due to Lemma 3.2.2.3.

Lemma 3.4.9: Any triangle that exists in G_{var} after Priority 8 must be pending with respect to some vertex v .

Proof. Consider some triangle $\Delta = \{a, b, c\}$. Due to *scp*, w.l.o.g., a has a neighbor $x \notin \{b, c\}$. If Δ is not pending with respect to x , then, w.l.o.g., $b \in N(x)$. After Priority 8, no vertex has weight 4 (or more). Due to *scp*, x must have a third neighbor which is not c . Due to Lemma 3.2.2.2 c must have another neighbor v (which is not x). Since neither a nor b are neighbors of v , Δ is pending with respect to v . \square

Thus, if we find a triangle in Priority 9 then it must be pending due to Lemma 3.4.9.

Lemma 3.4.10: Let v have a pending triangle a, b, c and $N(v) = \{a, p, q\}$. Then by branching on v , we have an $(6\omega_3, 8\omega_3)$ -branch.

Proof. In $F[v]$ and $F[\bar{v}]$, the variables a, b, c form a 3-quasi-lasso. Therefore, due to Lemma 3.2.3.2, w.l.o.g., only b remains in the reduced formula with $\#_2(b) = 2$. Also, in both branches, q and p are of weight two and therefore deleted. This is already a total of six deleted variables. Note that $N(\{q, p\}) \cup \{q, p\} \subseteq \{v, a, b, c, q, p\}$ contradicts *scp*, see Figure 3.4(a). Therefore, w.l.o.g., there is a variable $z \in N(q)$ such that $z \notin$

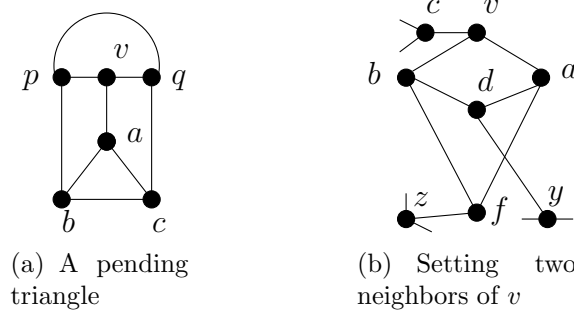


Figure 3.4.: Explaining $(6\omega_3, 8\omega_3)$ -branches, see the proofs of Lemmas 3.4.10 and 3.4.11

$\{v, a, b, c, q, p\}$. So, in the branch where q is set, also z will be deleted. Thus, seven variables will be deleted. The claim follows with $(*)$. \square

3.4.4.2. Priority 10

From now on, due to **HP**, G_{var} is triangle-free (see Lemma 3.4.9) and cubic. We show that if we are forced to choose a vertex v to which none of the priorities 1-9 fits, we can choose v such that we obtain either a $(6\omega_3, 8\omega_3)$ - or a $(4\omega_3, 10\omega_3)$ -branch.

Lemma 3.4.11: Let v be a vertex in G_{var} and $N(v) = \{a, b, c\}$. Suppose that, w.l.o.g., in $F[v]$ a, b and in $F[\bar{v}]$ c will be set when branching according to Priority 10. Then, we have a $(6\omega_3, 8\omega_3)$ -branch.

Proof. If $|(N(a) \cup N(b)) \setminus \{v\}| \geq 3$, then by setting a and b in $F[v]$, five variables will be reduced. Together with v and c , this is a total of seven. If $|(N(a) \cup N(b)) \setminus \{v\}| = 2$, then we must face a situation as depicted in Fig. 3.4(b) (note the absence of triangles). Observe that if $z = y$ ($z = c$, resp.) then also $z \neq c$ ($z \neq y$, resp.) due to *scp*. Then in $F[v]$ due to Lemma 3.2.2.1 v, a, b, c, d, f, z, y will be deleted. This are at least seven variables. In $F[\bar{v}]$ at least five variables disappear due to c being set and $|N(c) \setminus \{v, a, b\}| \geq 1$. Thus, we have a $(6\omega_3, 8\omega_3)$ -branch due to $(*)$. \square

Lemma 3.4.12: Consider branching according to Priority 10. If for any $v \in V(F)$, all its neighbors are set in one branch (say, in $F[v]$), we can perform a $(6\omega_3, 8\omega_3)$ - or a $(4\omega_3, 10\omega_3)$ -branch.

Proof. If $|N(\{a, b, c\}) \setminus \{v\}| \geq 5$, then in $F[v]$, 9 variables are deleted, so that we have a $(4\omega_3, 10\omega_3)$ -branch. Otherwise, either one of the two following situations must occur:

a) There is a variable $y \neq v$, such that $N(y) = \{a, b, c\}$, see Figure 3.5(a). Then branch on b . In $F[\bar{b}]$ v, y, a, c, z will disappear (due to **RR-5** and Lemma 3.2.3.1). In $F[b]$, due to z being set, additionally a neighbor $f \notin \{a, b, c, v, y, z\}$ of z will be deleted as the vertices a, b, c, v, y, z do not form a component by *scp*. This is a total of seven variables. By $(*)$, a $(6\omega_3, 8\omega_3)$ -branch is implied in the worst case.

b) There are variables p, q , such that $|N(p) \cap \{a, b, c\}| = |N(q) \cap \{a, b, c\}| = 2$. W.l.o.g.,

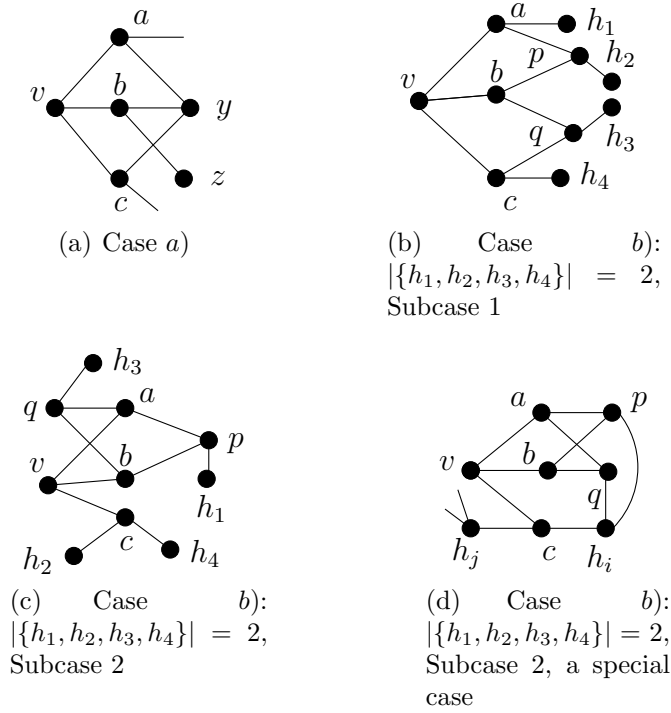


Figure 3.5.: Situations that may arise in Lemma 3.4.12, when all neighbors of v are set in one branch.

$\{a, b\} \subseteq N(p)$. Two subcases arise:

Subcase 1: W.l.o.g., $\{b, c\} \subseteq N(q)$, see Figure 3.5(b). Let h_1, h_2, h_3, h_4 be the hitherto unnamed neighbors of a, p, q, c , resp., that must exist due to Lemma 3.2.2.2.

Subcase 2: $\{a, b\} \subseteq N(q)$, see Figure 3.5(c). Let h_1, h_2, h_3, h_4 be the hitherto unnamed neighbors of p, q, c , resp., that must exist due to Lemma 3.2.2.2.

We will argue for both subcases in the following and point out where we distinguish between them. In both subcases, in the branch $F[v]$, the variables a, b, c, p, q will be set (Lemma 3.2.2.1). Let $H = \{h_1, h_2, h_3, h_4\}$. If $|H| \geq 3$ then at least nine variables are deleted in $F[v]$. This gives a $(4\omega_3, 10\omega_3)$ -branch due to (*). Otherwise, we have $|H| = 2$ due to degree restrictions. Therefore $|\{v, a, b, c, p, q\} \cup H| = 8$.

Now, consider Subcase 1. Due to triangle-freeness we must have $h_1 = h_3$ and $h_2 = h_4$ or $h_1 = h_4$ and $h_2 = h_3$. On the other hand, there is $z \in N(\{v, a, b, c, p, q\} \cup H)$ due to scp . Hence, at least nine variables vanish in $F[v]$ due to the setting of the H -vertices (Lemma 3.2.2.1), which again entails a $(4\omega_3, 10\omega_3)$ -branch due to (*).

In Subcase 2, if $|H| = 2$ then there is a h_j with $h_j \notin \{h_i \mid i = 1 \dots 4; i \neq j\}$. It follows that we have the special situation from Figure 3.5(d). We see at once that **RR-6** applies to this situation, which cannot be at that point. If we do not find such a h_j , then, w.l.o.g., $h_1 = h_2$ and $h_3 = h_4$, and we can argue as in Figure 3.5(b) to get a $(4\omega_3, 10\omega_3)$ -branch. \square

We remark that Theorem 4.2 of [102] contains also an alternative but less detailed

proof of case *b*) of the previous proof¹

Having arrived at Priority 10 we try to find a variable v such that Lemma 3.4.11 applies. If we fail to do so, then v is covered by Lemma 3.4.12. Due to the last three lemmas, branchings according to priorities 9 and 10 are upper bounded by $\mathcal{O}^*(2^{\frac{K}{6.1489}})$. Especially, the $(4\omega_3, 10\omega_3)$ -branch is sharp, i.e, the branching number is $2^{\frac{K}{6.1489}}$. Measured in n (the number of variables) the running time of these two priorities is upper-bounded by $\mathcal{O}^*(1.11199^n)$.

3.5. Combining Two Approaches

3.5.1. General Exposition

A. S. Kulikov and K. Kutzkov [104] achieved a run time of $\mathcal{O}^*(2^{\frac{K}{5.88}})$. This was obtained by speeding up the 5-phase by a concept called 'clause learning'. As in our approach the 3- and 4-phase was improved, we will show that if we use both strategies we can even beat our previous run time bound. This means that in **HP** we substitute Priority 3 by their strategy with one exception: we prefer variables v with a non-weight-5 neighbor (\star). Forced to violate this preference we do a simple branching of the form $F[v]$ and $F[\bar{v}]$.

For the analysis we redefine the measure $\gamma(F)$: we set $\omega_3 = 0.9521$, $\omega_4 = 1.8320$, $\omega_5 = 2.488$ and keep the other weights. We call this measure $\tilde{\gamma}(F)$. We will reproduce the analysis of [104] briefly with respect to $\tilde{\gamma}(F)$ to show that their derived branches for the 5-phase are upper bounded by $\mathcal{O}^*(2^{\frac{K}{6.2158}})$. It also can be checked that this is also true for the branches derived for the other phases by measuring them in terms of $\tilde{\gamma}(F)$, see section 3.5.3.

Let k_{ij} denote the number of weight j variables occurring i times in a 2-clause with some $v \in V(F)$ chosen for branching. Note that $i \leq j - 2$ due to **RR-5**. Then we must have:

$$k_{13} + k_{14} + k_{15} + 2k_{24} + 2k_{25} + 3k_{35} = 5$$

If F' is the formula obtained by assigning a value to v and by applying the reduction rules afterwards we have:

$$\begin{aligned} \tilde{\gamma}(F) - \tilde{\gamma}(F') &\geq 5\Delta_5 + \omega_5 + (\omega_3 - \Delta_5)k_{13} + (\Delta_4 - \Delta_5)k_{14} + \left(\frac{\omega_4}{2} - \Delta_5\right)2k_{24} \\ &\quad + (\Delta_4 - \Delta_5)k_{25} + \left(\frac{\omega_5}{2} - \frac{3}{2}\Delta_5\right)2k_{35} \\ &= 5.768 + 0.2961k_{13} + 0.2239(k_{14} + k_{25}) + 0.26 \cdot 2(k_{24} + k_{35}) \end{aligned}$$

Basically we reduce $\tilde{\gamma}(F)$ by at least $\omega_5 + 5\Delta_5$. Now the coefficients of the k_{ij} in the above equation express how the reduction grows if $k_{ij} > 0$.

¹Actually, in [102] it is claimed that branching on v in the situation of Figure 3.5(d) entails a $(4\omega_3, 10\omega_3)$ -branch. But in $F[v]$ (where v, a, b, c are set) it is possible that only $v, a, b, c, p, q, h_i, h_j$ will disappear. However, a $(8\omega_3, 8\omega_3)$ -branch can be shown.

case	one component	both components	upper bound
$k_{15} = 5$	$(\{7\omega_5 + 5\Delta_5\}^2, \omega_5 + 5\Delta_5)$	$(\{7\omega_5 + 5\Delta_5\}^4)$	$\mathcal{O}^*(1.0846^K)$
$k_{13} = 1, k_{15} = 4$	$(\{6\omega_5 + \omega_3 + 5\Delta_5\}^2, \omega_5 + 4\Delta_5 + \omega_3)$	$\{6\omega_5 + \omega_3 + 5\Delta_5\}^4)$	$\mathcal{O}^*(1.0878^K)$
$k_{25} = 1, k_{15} = 3$	$(\{6\omega_5 + 5\Delta_5\}^2, \omega_5 + 3\Delta_5 + (\omega_5 - \omega_3))$	$(\{6\omega_5 + 5\Delta_5\}^4)$	$\mathcal{O}^*(1.0914^K)$

Table 3.2.:

If $k_{13} + k_{14} + 2k_{24} + k_{25} + 2k_{35} \geq 2$ we are done as $\tilde{\gamma}(F) - \tilde{\gamma}(F') \geq 6.2158$.

If $k_{13} = 1$ and $k_{15} = 4$ then [104] stated a

$(5\Delta_5 + \omega_5 + (\omega_3 - \Delta_5), 5\Delta_5 + \omega_5 + (\omega_3 - \Delta_5) + 2\Delta_5)$ -branch and for

$k_{25} = 1$ and $k_{15} = 3$ a $(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), 5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5) + \omega_3)$ -branch.

If $k_{14} = 1$ and $k_{15} = 4$ a branching of the kind $F[v], F[\bar{v}, v_1], F[\bar{v}, \bar{v}_1, v_2, v_3, v_4, v_5]$ is applied, where $\{v_1, \dots, v_5\} = N(v)$. From this follows a

$(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), 4\Delta_5 + \omega_5 + \Delta_4 + \omega_4 + 3\Delta_4 + \Delta_5, 5\omega_5 + \omega_4)$ - and a

$(\omega_5 + 4\Delta_5 + \Delta_4, \omega_5 + 4\Delta_5 + \Delta_4 + \omega_4 + 4\Delta_5, 5\omega_5 + \omega_4 + 3\Delta_5)$ -branch.

This depends on whether v_1 has at least three neighbors of weight less than 5 in $F[\bar{v}]$ or not. We observed that we can get a additional reduction of Δ_5 in the third component of the first branch as $N[v]$ cannot be a component in $V(F)$ after step 3 of Alg. 10. The original amount of $5\omega_5 + \omega_4$ exclusively comes from the vertices in $N[v]$. This yields a $(4\Delta_5 + \omega_5 + \Delta_4, 5\Delta_5 + \omega_5 + 4\Delta_4 + \omega_4, 5\omega_5 + \omega_4 + \Delta_5)$ -branch.

The analysis of the 5-regular branch (i.e., $k_{15} = 5$) proceeds the same way as in the simple version of the algorithm except that we have to take into account the newly introduced branches, see Sec. 3.5.2. Due to (★) the case where $k_{15} = 1$ also only happens when the graph is 5-regular.

Theorem 3.5.1: MAX-2-SAT can be solved in time $\mathcal{O}^*(2^{\frac{K}{6.2158}}) \approx \mathcal{O}^*(1.118^K)$.

The following subsections provide the details missing in the general exposition above.

3.5.2. 5-regular Branches in the Combined Approach

Internal 5-regular branches yield the same recurrences as in the simple approach. Final 5-regular branches must be analyzed together with their immediate preceding branch. Thus they have to be analyzed together with the introduced branches of [104].

Table 3.2 captures some cases ($k_{15} = 5$; $k_{13} = 1, k_{14} = 4$; $k_{25} = 1, k_{15} = 3$). For the case $k_{14} = 1$ and $k_{15} = 4$ there are two recurrences for the branching of the form $F[v], F[\bar{v}, v_1], F[\bar{v}, \bar{v}_1, v_2, v_3, v_4, v_5]$. The first recurrence (**A**) assumes that v_1 has at least three neighbor of weight less than five in $F[\bar{v}]$:

$$(\mathbf{A}) (5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), 4\Delta_5 + \omega_5 + \Delta_4 + \omega_4 + 3\Delta_4 + \Delta_5, 5\omega_5 + \omega_4 + \Delta_5).$$

Branch-type	components	combined branch	upper bound
(A)	(1)	$(\{6\omega_5 + \omega_4 + 5\Delta_5\}^2, 5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5) + \omega_4 + 3\Delta_4 + \Delta_5, 5\omega_5 + \omega_4 + \Delta_5)$	$\mathcal{O}^*(1.0912^K)$
(A)	(2)	$(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), \{6\omega_5 + \omega_4 + 5\Delta_5\}^2, 5\omega_5 + \omega_4 + \Delta_5)$	$\mathcal{O}^*(1.1094^K)$
(A)	(3)	$(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), 5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5) + \omega_4 + 3\Delta_4 + \Delta_5, \{6\omega_5 + \omega_4 + 5\Delta_5 + \omega_3\}^2)$	$\mathcal{O}^*(1.1175^K)$
(B)	(1)	$(\{6\omega_5 + \omega_4 + 5\Delta_5\}^2, 5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5) + \omega_4 + 4\Delta_5, 5\omega_5 + \omega_4 + 3\omega_3)$	$\mathcal{O}^*(1.0894^K)$
(B)	(2)	$(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), \{6\omega_5 + \omega_4 + 5\Delta_5\}^2, 5\omega_5 + \omega_4 + 3\omega_3)$	$\mathcal{O}^*(1.1052^K)$
(B)	(3)	$(5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5), 5\Delta_5 + \omega_5 + (\Delta_4 - \Delta_5) + \omega_4 + 4\Delta_5, \{6\omega_5 + \omega_4 + 5\Delta_5 + 3\omega_3\}^2)$	$\mathcal{O}^*(1.1159^K)$
(A)/ (B)	(1) + (2)/ (1) + (3)/ (2) + (3)	$(\{6\omega_5 + \omega_4 + 5\Delta_5\}^4, \omega_5 + 4\Delta_5 + \Delta_4)$	$\mathcal{O}^*(1.1126^K)$
(A)/ (B)	(1) + (2) + (3)	$(\{6\omega_5 + \omega_4 + 5\Delta_5\}^6)$	$\mathcal{O}^*(1.0936^K)$

Table 3.3.: The second column indicates after which components a final 5-regular branch immediately follows.

$$(B) (\omega_5 + 4\Delta_5 + \Delta_4, \omega_5 + 4\Delta_5 + \Delta_4 + \omega_4 + 4\Delta_5, 5\omega_5 + \omega_4 + 3\omega_3).$$

(B) captures the remaining case. Both branches have three components. Table 3.3 captures the combined analysis of a immediately following final 5-branch and branches (A) and (B). This depends on whether the final 5-regular branch follows after the first (1), the second (2) or the third (3) component or in any combination of them.

We would like to comment the recurrences in the third row of Table 3.3. Here we get a reduction of ω_3 in addition to $5\omega_5 + \omega_4$ from v, v_1, \dots, v_5 in the third part of the branch. This additional amount comes from clauses C such that $|C \cap \{v, v_1, \dots, v_5\}| = 1$. Due to *scp*, $N[v]$ is not a component and thus at least one further variable must be deleted.

The next proposition serves for covering the remaining branching cases, which can precede a final 5-regular branch.

Proposition 3.5.2: Let $v \in V(F)$ be the variable chosen for branching by Alg. 10 such that $\#_2(v) = 5$. Assume v induces a solution to Equation (3.1) in Section 3.4.3 such that it is different from $k_{13} = 1, k_{15} = 4$; $k_{15} = 5$; $k_{25} = 1, k_{15} = 3$; $k_{14} = 1, k_{15} = 4$ (\star). If a 5-regular branch follows in one component we have at least a $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^2, \omega_5 + 3\Delta_5 + 2\Delta_4)$ -branch and if it follows in both a $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^4)$ -

branch in the combined analysis.

Proof. If a component is followed by a final 5-regular branch the least amount we get by reduction from $N(v)$ is $\omega_5 + \omega_4$. This refers to the case $k_{35} = 1$ and $k_{24} = 1$ which follows from Proposition 3.4.5.

The least reduction from $N(v)$ without a following final 5-regular branch can be found as follows: Consider any solution of Equation (3.1) except the ones in (\star) . Among them find one which minimizes

$$\Delta_3 k_{13} + \Delta_4 k_{14} + \Delta_5 k_{15} + (\Delta_4 + \Delta_3) k_{24} + (\Delta_5 + \Delta_4) k_{25} + (\Delta_5 + \Delta_4 + \Delta_3) k_{35} \quad (3.3)$$

Clearly, $k_{13} = 0$ because $\Delta_3 > \Delta_5$. We could decrease the amount of reduction by decreasing k_{13} by one and by increasing k_{15} . Let k'_{13} and k'_{15} be the new values. Observe that we might end up in the case $k'_{15} = 5$ but only if we started from $k_{13} = 1; k_{15} = 4$. This is forbidden as this excluded due to (\star) .

For the same reason we can assume $k_{24} = 0$ as we have $\Delta_4 + \Delta_3 > \Delta_5 + \Delta_4$ and the above modifications apply. In the case where $k_{24} = 1; k_{15} = 3$ we end up in the case $k'_{25} = 1; k'_{15} = 3$. We can rule out this case because $k_{14} = 2; k_{15} = 3$ is a smaller solution than $k_{24} = 1; k_{15} = 3$.

As we are excluding (\star) we must have $k_{15} \leq 4$. If $k_{15} = 4$ we conclude that either $k_{14} = 1$ or $k_{13} = 1$. These solutions are forbidden (see (\star)). Thus we must have $k_{15} \leq 3$.

If $k_{35} = 1$ then there is a better solution as $\Delta_5 + 2\Delta_4 < \Delta_5 + \Delta_4 + \Delta_3$: set $k'_{35} = 0, k'_{15} = k_{15} + 1, k'_{14} = k_{14} + 2$ and keep the other coefficients. Note that in this case we must have $k_{15} \leq 2$ otherwise $3k_{35} + k_{15} > 5$. This assures that the new solution is different from the ones in (\star) (even in case $k'_{15} = 3$). Therefore it follows that $k_{35} = 0$.

Now suppose $k_{25} = 2$, then $k_{15} = 1$ holds because $\Delta_5 < \Delta_4$. But then $k'_{25} = 1, k'_{15} = 2, k'_{14} = 1$ is a no worse solution. Thus $k_{25} \leq 1$.

If $k_{25} = 1$ then with $k_{15} = 2$ and $k_{14} = 1$ this is minimal under this condition (since $k_{15} = 3$ is forbidden (\star)). Suppose $k_{25} = 0$, then clearly the best solution is $k_{15} = 3$ and $k_{14} = 2$. Both solutions provide a reduction of $3\Delta_5 + 2\Delta_4$ which is minimal.

Now we analyze a final 5-regular branch and a branch different from (\star) satisfying Equation (3.1). If the final 5-regular branch follows in only one component then we have at least a $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^2, \omega_5 + 3\Delta_5 + 2\Delta_4)$ -branch in the combined analysis. If it follows in both, then a $(\{3\omega_5 + \omega_4 + 5\Delta_5\}^4)$ -branch upper-bounds correctly. \square

Due to Proposition 3.5.2 we can upper-bound the final 5-regular branches whose predecessors are different from $k_{13} = 1, k_{15} = 4; k_{15} = 5; k_{25} = 1, k_{15} = 3; k_{14} = 1, k_{15} = 4$ by $\mathcal{O}^*(1.1171^K)$ in their combined analysis.

3.5.3. Analysis of the 6- 4- and 3-phase in the Combined Approach

Here we provide the run times under $\tilde{\gamma}(F)$ for the cases we did not consider in Section 3.5. The run time has been estimated with respect to $\tilde{\gamma}(F)$. Names will refer to the corresponding ones in the analysis of Alg. 10.

case	branch	upper bound
Internal 6-regular	$(3\omega_6, \omega_6 + 6\Delta_6)$	$\mathcal{O}^*(1.0978^K)$
	$(\{3\omega_6\}^2)$	$\mathcal{O}^*(1.0802^K)$
Internal 5-regular	$(3\omega_5, \omega_5 + 5\Delta_5)$	$\mathcal{O}^*(1.1112^K)$
	$(\{3\omega_5\}^2)$	$\mathcal{O}^*(1.0974^K)$
Internal 4-regular	$(5\omega_4, \omega_4 + 4\Delta_4)$	$\mathcal{O}^*(1.103^K)$
	$(\{5\omega_4\}^2)$	$\mathcal{O}^*(1.079^K)$

 Table 3.5.: Internal h -regular cases ($h \in \{4, 5, 6\}$) an their upper bounds.

3.5.3.1. Non-regular Branches

In Table 3.4 we find the derived recurrences for each priority of **HP** if we have a non-regular branch. You can find them together with their run times. Priority 3 is not considered as the 5-phase has been analyzed in Sections 3.5 and 3.5.2.

Priorities	branch	upper bound
Priority 1	$(7, 7)$	$\mathcal{O}^*(1.1042^K)$
Priority 2	$(\{\omega_6 + 5\Delta_6 + \Delta_5\}^2)$	$\mathcal{O}^*(1.118^K)$
Priority 4	$(\{2\omega_4 + 4\Delta_4\}^2)$	$\mathcal{O}^*(1.102^K)$
Priority 5	$(\{3\omega_4 + 2\omega_3\}^2)$	$\mathcal{O}^*(1.1099^K)$
Priority 6	$(2\omega_4 + 2\omega_3 + 2\Delta_4, \omega_4 + 2\omega_3 + 2\Delta_4)$	$\mathcal{O}^*(1.1143^K)$
Priority 7 3-path, $u_0 \neq u_{l+1}$ 3-path, $u_0 = u_{l+1}$ 4-path	$(\omega_4 + \omega_3 + 5\Delta_4, \omega_4 + \omega_3 + 3\Delta_4)$	$\mathcal{O}^*(1.1172^K)$
	$(\{\omega_4 + 3\omega_3 + 3\Delta_4\}^2)$	$\mathcal{O}^*(1.1^K)$
	$(2\omega_4 + \omega_3 + 3\Delta_4, \omega_4 + \omega_3 + 3\Delta_4)$	$\mathcal{O}^*(1.1165^K)$
Priority 8	$(\{2\omega_4 + 2\omega_3 + 2\Delta_4\}^2)$	$\mathcal{O}^*(1.1^K)$
Priority 9	$(8\omega_3, 6\omega_3)$	$\mathcal{O}^*(1.1105^K)$
Priority 10	$(8\omega_3, 6\omega_3)$	$\mathcal{O}^*(1.1105^K)$
	$(4\omega_3, 10\omega_3)$	$\mathcal{O}^*(1.118^K)$

Table 3.4.: The non-regular cases

3.5.3.2. Regular Branches

Table 3.5 captures the run times of any internal 6, 5 or 4-regular branch. Table 3.6 considers final 4 or 6-regular branches together with their preceding branches. Note that final 5-regular branches have been considered in section 3.5.2. The case where we have chosen v due to Priority 7 such that v has a 3-path with $u_0 \neq u_l$ is treated separately.

3-path Finally we consider the case when a variable chosen to Priority 7 has a 3-path with $u_0 \neq u_{l+1}$. The cases a) $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$, b) $(\omega_4 + \omega_3 + 5\Delta_4, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ and c) $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ are upper-bounded by $\mathcal{O}^*(1.1152^K)$, $\mathcal{O}^*(1.1159^K)$ and $\mathcal{O}^*(1.1147^K)$. Table 3.7 captures the branches together

Preceding branch	branch	upper bound
Final 6-regular Branch		
Any 6-phase branch	$(\{3\omega_6 + \omega_4 + 6\Delta_6\}^2, \omega_6 + 6\Delta_6)$ $(\{3\omega_6 + \omega_4 + 6\Delta_6\}^4)$	$\mathcal{O}^*(1.11^K)$ $\mathcal{O}^*(1.105^K)$
Final 4-regular Branch		
Internal 4-regular	$(\{6\omega_4 + 4\Delta_4\}^2, \omega_4 + 4\Delta_4)$ $(\{6\omega_4 + 4\Delta_4\}^4)$	$\mathcal{O}^*(1.1115^K)$ $\mathcal{O}^*(1.1003^K)$
Priorities 4,5 and 8 Cases 2,3(a),4 of Priority 6	$(\{3\omega_4 + 8\Delta_4\}^2, 2\omega_4 + 4\Delta_4)$ $(\{3\omega_4 + 8\Delta_4\}^4)$	$\mathcal{O}^*(1.1115^K)$ $\mathcal{O}^*(1.117^K)$
Priority 6 Case 1 Case 5,3(b) b) and c) of the analysis Case 3b), case a') of the analysis Case 5, case a') of the analysis	$(\{3\omega_4 + 2\omega_3 + 4\Delta_4\}^2)$ $(\{2\omega_4 + 2\omega_3 + 2\Delta_4, \{3\omega_4 + 4\omega_3 + 4\Delta_4\}^2\} b)$ $(\{3\omega_4 + 4\omega_3 + 4\Delta_4\}^4) c)$ $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + 2\omega_3 + 2\Delta_4)$ $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, 2\omega_4 + 2\omega_3)$	$\mathcal{O}^*(1.07^K)$ $\mathcal{O}^*(1.109^K)$ $\mathcal{O}^*(1.1143^K)$ $\mathcal{O}^*(1.1145^K)$ $\mathcal{O}^*(1.1140^K)$
Priority 7 Case of a 4-path Case b) Case a') Case c') Case a'') Case c'') Case of a 3-path with $u_0 = u_{l+1}$	$(2\omega_4 + \omega_3 + 3\Delta_4, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ $(\{4\omega_4 + 3\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$ $\{4\omega_4 + 3\omega_3 + 4\Delta_4\}^2, \{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, \omega_4 + \omega_3 + 3\Delta_4)$ $(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2, (\{3\omega_4 + 3\omega_3 + 4\Delta_4\}^2)$ similar to priorities 4,5 and 8	$\mathcal{O}^*(1.1155^K)$ $\mathcal{O}^*(1.1156^K)$ $\mathcal{O}^*(1.115^K)$ $\mathcal{O}^*(1.1152^K)$ $\mathcal{O}^*(1.1147^K)$

 Table 3.6.: The final h -regular cases ($h \in \{4, 6\}$) and their combined analysis

$\#_2(u_0), \#_2(u_{l+1})$	left component	right component	both components
$\#_2(u_0) = 3$ $\#_2(u_{l+1}) = 3$ upper bounds	case α instead	$(\omega_4 + 6\omega_3,$ $\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2)$ $\mathcal{O}^*(1.1075^K)$	$(\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 6\omega_3 + 4\Delta_4\}^2)$ $\mathcal{O}^*(1.1136^K)$
$\#_2(u_0) = 3$ $\#_2(u_{l+1}) = 4$ upper bounds	$(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2,$ $\omega_4 + 4\omega_3)$ $\mathcal{O}^*(1.1136^K)$	$(\omega_4 + 5\omega_3 + \Delta_4,$ $\{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$ $\mathcal{O}^*(1.1138^K)$	$(\{3\omega_4 + 5\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$ $\mathcal{O}^*(1.1143^K)$
$\#_2(u_0) = 4$ $\#_2(u_{l+1}) = 4$ upper bounds	$(\{4\omega_4 + 4\omega_3 + 4\Delta_4\}^2,$ $\omega_4 + 4\omega_3)$ $\mathcal{O}^*(1.1088^K)$	case β instead	$(\{4\omega_4 + 4\omega_3 + 4\Delta_4\}^2,$ $\{2\omega_4 + 4\omega_3 + 4\Delta_4\}^2)$ $\mathcal{O}^*(1.1088^K)$

Table 3.7.:

with their run times in the combined algorithm if for all $y \in N(v)$ we have $\#_2(y) = 3$. We have a $(\{2\omega_4 + 7\omega_3 + 4\Delta_4\}^2, \omega_4 + 4\omega_3)$ -branch for case α') which $\mathcal{O}^*(1.1132^K)$ properly upper bounds. We also have a $(\omega_4 + 4\omega_3 + 2\Delta_4, \{2\omega_4 + 5\omega_3 + 4\Delta_4\}^2)$ -branch for case β') such that it is upper-bounded by $\mathcal{O}^*(1.1142^K)$. Any branch where $\#(v) \geq 7$ has an upper bound of $\mathcal{O}^*(1.1041^K)$.

3.6. Conclusion

We presented an algorithm solving MAX-2-SAT in $\mathcal{O}^*(2^{\frac{K}{6.2158}})$, with K the number of clauses of the input formula. This is currently the end of a sequence of polynomial-space algorithms each improving on the run time, strictly staying within the realm of MAX-2-SAT: beginning with $\mathcal{O}^*(2^{\frac{K}{2.88}})$ which was achieved by [126], it was subsequently improved to $\mathcal{O}^*(2^{\frac{K}{3.742}})$ by [85], to $\mathcal{O}^*(2^{\frac{K}{5}})$ by [84], to $\mathcal{O}^*(2^{\frac{K}{5.217}})$ by [99], to $\mathcal{O}^*(2^{\frac{K}{5.5}})$ by [102], to $\mathcal{O}^*(2^{\frac{K}{5.88}})$ by [104] and finally to the hitherto fastest upper bound of $\mathcal{O}^*(2^{\frac{K}{6}})$ by [105]. Our improvement has been achieved due to heuristic priorities concerning the choice of the variable for branching in case of a maximum degree four variable graph. As [104] improved the case where the variable graph has maximum degree five, it seems that the only way to speed up the generic branching algorithm is to improve the maximum degree six case. Our analysis also implies that the situation when the variable graph is regular is not that harmful. The reason for this that the preceding branch must have reduced the problem size more than expected. Thus considered together these two branches balance each other. Though the analysis is to some extent sophisticated and quite detailed the algorithm has a clear structure. The implementation of the heuristic priorities for the weight 4 variables should be a straightforward task.

Chapter 4.

Exact and Parameterized Algorithms for MAX INTERNAL SPANNING TREE

4.1. Introduction

Motivation.

We investigate the following problem:

MAX INTERNAL SPANNING TREE (MIST)

Given: A graph $G = (V, E)$ with n vertices and m edges.

Task: Find a spanning tree of G with a maximum number of internal vertices.

MIST is a generalization of the famous and well-studied HAMILTONIAN PATH problem. Here, one is asked to find a path in a graph such that every vertex is visited exactly once. Clearly, such a path, if it exists, is also a spanning tree, namely one with a maximum number of internal vertices. Whereas the running time barrier of 2^n has not been broken for general graphs, there are faster algorithms for cubic graphs (using only polynomial space). It is natural to ask if for the generalization, MIST, this can also be obtained.

A second issue is if we can find an algorithm for MIST with a running time of the form $\mathcal{O}^*(c^n)$. The very naïve approach gives only an upper bound of $\mathcal{O}^*(2^m)$. A possible application could be the following scenario. Suppose you have a set of cities which should be connected with water pipes. The possible connections between them can be represented by a graph G . It suffices to compute a spanning tree T for G . In T we may have high degree vertices that have to be implemented by branching pipes. These branching pipes cause turbulences and therefore pressure may drop. To minimize the number of branching pipes one can equivalently compute a spanning tree with the smallest number of leaves, leading to MIST. Vertices representing branching pipes should not be of arbitrarily high degree, motivating us to investigate MIST on degree-restricted graphs.

Previous Work.

It is well-known that the more restricted problem, HAMILTONIAN PATH, can be solved within $\mathcal{O}(n^2 2^n)$ steps and exponential space. This result has been independently obtained by R. Bellman [5], and M. Held and R.M. Karp [91]. The TRAVELING SALES-

MAN problem (TSP) is very closely related to HAMILTONIAN PATH. Basically, the same algorithm solves this problem, but there has not been any improvement on the running time since 1962. The space requirements have, however, been improved and now there are $\mathcal{O}^*(2^n)$ algorithms needing only polynomial space. In 1977, S. Kohn *et al.* [101] gave an algorithm based on generating functions with a running time of $\mathcal{O}(2^n n^3)$ and space requirements of $\mathcal{O}(n^2)$ and in 1982 R.M. Karp [96] came up with an algorithm which improved storage requirements to $\mathcal{O}(n)$ and preserved this run time by an inclusion-exclusion approach.

D. Eppstein [37] studied TSP on cubic graphs. He could achieve a running time of $\mathcal{O}(1.260^n)$ using polynomial space. K. Iwama and T. Nakashima [95] could improve this to $\mathcal{O}(1.251^n)$. A. Björklund *et al.* [10] considered TSP with respect to degree-bounded graphs. Their algorithm is a variant of the classical 2^n -algorithm and the space requirements are therefore exponential. Nevertheless, they showed that for a graph with maximum degree d there is a $\mathcal{O}^*((2 - \epsilon_d)^n)$ -algorithm. In particular for $d = 4$ there is a $\mathcal{O}(1.8557^n)$ - and for $d = 5$ a $\mathcal{O}(1.9320^n)$ -algorithm.

MIST was also studied with respect to parameterized complexity. The (standard) parameterized version of the problem is parameterized by k , and asks whether G has a spanning tree with at least k internal vertices. E. Prieto and C. Sloper [131] proved a $\mathcal{O}(k^3)$ -vertex kernel for the problem showing \mathcal{FPT} -membership. By the same authors [130, 132] the kernel size has been improved to $\mathcal{O}(k^2)$ and in F.V. Fomin *et al.* [65, 66] to $3k$. Parameterized algorithms for MIST have been studied in [28, 65, 66, 132] (N. Cohen *et al.*, F.V. Fomin *et al.*, E. Prieto and C. Sloper). E. Prieto and C. Sloper [132] gave the first FPT algorithm, with running time $2^{4k \log k} \cdot n^{\mathcal{O}(1)}$. This result was improved by N. Cohen *et al.* [28] who solve a more general directed version of the problem in time $49.4^k \cdot n^{\mathcal{O}(1)}$. The fastest algorithm has running time $8^k \cdot n^{\mathcal{O}(1)}$ [65, 66].

G. Salamon [141] studied the problem considering approximation. He could achieve a $\frac{7}{4}$ -approximation. A $2(\Delta - 2)$ -approximation for the node-weighted version is a by-product. Cubic and claw-free graphs were considered by G. Salamon and G. Wiener [142]. They introduced algorithms with approximation ratios $\frac{6}{5}$ and $\frac{3}{2}$, respectively.

Our Results

Two algorithms are presented:

- (a) A dynamic-programming algorithm solving MIST in time $\mathcal{O}^*(3^n)$. We extend this algorithm and show that for any degree-bounded graph a running time of $\mathcal{O}^*((3 - \epsilon)^n)$ with $\epsilon > 0$ can be achieved.
- (b) A branching algorithm solving the maximum degree 3 case in time $\mathcal{O}(1.8669^n)$. The space requirements are only polynomial in this case. We also analyze the same algorithm from a parameterized point of view, achieving a running time of $2.1364^k n^{\mathcal{O}(1)}$ to find a spanning tree with at least k internal vertices (if the graph admits such a spanning tree). The latter analysis is novel in a sense that we use a potential function analysis—*Measure&Conquer*—in a way that, to our knowledge,

is much less restrictive than any previous analysis for parameterized algorithms that were based on the potential function method.

Notions and Definitions.

For a (partial) spanning tree $T \subseteq E$ let $internal(T)$ be the set of its internal (non-leaf) vertices and $leaves(T)$ the set of its leaves. An i -vertex u is a vertex with $d_T(u) = i$ with respect to some spanning tree T , where $d_H(u) := |\{\{u, v\} \mid \{u, v\} \in H\}|$ for any $H \subseteq E$. The *tree-degree* of some $u \in V(T)$ is $d_T(u)$. We also speak of the T -degree $d_T(v)$ when we refer to a specific spanning tree. A *Hamiltonian path* is a sequence of pairwise distinct vertices v_1, \dots, v_n from V such that $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq n - 1$. A *triangle* in graph is a subgraph of the form $G(\{a, b, c\}, \{\{a, b\}\{b, c\}\{a, c\}\})$.

4.2. The Problem on General Graphs

We give a simple dynamic-programming algorithm to solve MIST within $\mathcal{O}^*(3^n)$ steps. Here we build up a table $M[I, L]$ with $I, L \subseteq V$ such that $I \cap L = \emptyset$. The set I represents the internal vertices and L the leaves of some tree with vertex set $I \cup L$ in G . If such a tree exists then we have $M[I, L] = 1$ and otherwise a zero-entry. In the beginning, we initialize all table-entries with zeros. In the initializing phase we iterate over all $e \in E$ and set $M[\emptyset, e] = 1$. Note that every edge is a tree with two leaves and no internal vertices. To compute further entries we use dynamic programming in stages $3, \dots, n$. Stage i consists in determining all table entries indexed by all $I, L \subseteq V$ with $|I| + |L| = i$ and $I \cap L = \emptyset$ such that $G[I \cup L]$ is connected and $M[I, L] = 1$. We obtain the table entries of stage i by inspecting the non-zero entries of stage $(i - 1)$. If $|I| + |L| = i - 1$ and $M[I, L] = 1$ then for every $x \in N(I \cup L)$ consider any possibility of attaching x as a leaf to the tree formed by $I \cup L$. There are two possibilities:

- a) x is adjacent to an internal vertex, then set $M[I, L \cup \{x\}] = 1$, and
- b) x is adjacent to a leaf y then set $M[I \cup \{y\}, (L \setminus \{y\}) \cup \{x\}] = 1$.

Recursively this can be expressed as follows:

$$M[I, L] = \begin{cases} 1: \exists x \in L \cap N(I) : M[I, L \setminus \{x\}] = 1 \\ 1: \exists x \in L, y \in N(x) \cap I : M[I \setminus \{y\}, (L \cup \{y\}) \setminus \{x\}] = 1 \\ 0: \text{otherwise} \end{cases} \quad (4.1)$$

Here we use the fact that, if we delete a leaf x of a tree T , then there are two possibilities for the resulting tree T' : Either T' has the same internal vertices as T but one leaf less, or the father y of x in T has become a leaf as $d_T(y) = 2$. These are exactly the two cases which are considered in Eq. (4.1). The number of entries in M is at most $\sum_{\substack{A, B \subseteq V \\ A \cap B = \emptyset}} 1 = \sum_{D \subseteq V} \sum_{C \subseteq D} 1 = 3^{|V|}$.

Lemma 4.2.1: MAX INTERNAL SPANNING TREE can be solved in time $\mathcal{O}^*(3^n)$.

Δ	3	4	5	6	7	8
Running Time	2.9680	2.9874	2.9948	2.9978	2.9991	2.9996

 Table 4.1.: Running times for graphs with maximum degree Δ .

Bounded Degree

We are particularly interested in solving MIST on graphs of bounded degree. The next lemma is due to [10].

Lemma 4.2.2: An n -vertex graph with maximum vertex degree Δ has at most $\beta_\Delta^n + n$ connected vertex sets with $\beta_\Delta = (2^{\Delta+1} - 1)^{\frac{1}{\Delta+1}}$.

In particular, n refers to the connected sets of size one, which is $\{\{x\} \mid x \in V\}$. Thus, the number of all connected sets of size greater than one is β_Δ^n . Using this we prove:

Lemma 4.2.3: For any n -vertex graph with maximum degree Δ there is an algorithm that solves MIST in time $\mathcal{O}^*(3^{(1-\epsilon_\Delta)n})$ with $\epsilon_\Delta > 0$.

Proof. As Lemma 4.2.2 bounds the number of connected subsets of V , we would like to skip unconnected ones. This is guaranteed by the approach of dynamic programming in stages. Let \mathcal{C} consist of the sets $F \subseteq V$ such that $G[F]$ is connected and $|F| \geq 2$. Then the number of visited entries of $M[I, L]$ with $|I| + |L| \geq 2$ and $I \cup L \in \mathcal{C}$ in all stages is at most

$$\sum_{\substack{A \subseteq V \\ A \in \mathcal{C}}} \sum_{\substack{I \subseteq A \\ I \in \mathcal{C}}} 1 \leq \sum_{\substack{A \subseteq V \\ A \in \mathcal{C}}} \beta_\Delta^{|A|} \leq \sum_{i=0}^n \binom{n}{i} \beta_\Delta^i = (\beta_\Delta + 1)^n$$

The visited entries $M[I, L]$ where $|I| + |L| = 1$ is n . As $\beta_\Delta < 2$ for any constant Δ , this shows Lemma 4.2.3. Table 4.1 gives an overview on the running times for small values of Δ . \square

A naïve approach to solve the degree restricted version of MIST is to consider each edge-subset. The running time is $\mathcal{O}^*(2^{\frac{\Delta}{2}n})$ where Δ is the maximum degree. Compared to Table 4.1, we see that for every $\Delta \geq 4$, this naïve algorithm is slower. A further slight improvement for $\Delta = 3$ provides the next observation. The line graph G_l of G has maximum degree four and hence there are no more than $\beta_4^{|V(G_l)|}$ connected vertex subsets. Clearly, G then has no more than $\beta_4^{|E(G)|}$ connected edge subsets. Having already a partial connected solution $T_E \subseteq E$ we only branch on edges $\{u, v\}$ with $u \in T_E$ and $v \notin T_E$. Thus, the run time is $\mathcal{O}^*(\beta_4^{\frac{3}{2}n}) = \mathcal{O}^*(2.8017^n)$. We can easily generalize this for arbitrary degree Δ to $\mathcal{O}^*(\beta_{2\Delta-2}^{\frac{\Delta}{2}n})$.

4.3. Subcubic Maximum Internal Spanning Tree

4.3.1. Observations

Let t_i^T denote the number of vertices u such that $d_T(u) = i$ for a spanning tree T . Then the following proposition can be proved by induction on $n_T := |V(T)|$.

Proposition 4.3.1: In any spanning tree T , $2 + \sum_{i \geq 3} (i - 2) \cdot t_i^T = t_1^T$.

Due to Proposition 4.3.1, MIST on subcubic graphs boils down to finding a spanning tree T such that t_2^T is maximum. Every internal vertex of higher degree would also introduce additional leaves.

Lemma 4.3.2: [131] An optimal solution T_o to MAX INTERNAL SPANNING TREE is a Hamiltonian path or the leaves of T_o are independent.

The proof of Lemma 4.3.2 shows that if T_o is not a Hamiltonian path and there are two adjacent leaves, then the number of internal vertices can be increased in polynomial time. Therefore, if some spanning tree T is not a Hamiltonian path then we can assume that its leaves induce an independent set. In the rest of the chapter we assume that T_o is not a Hamiltonian path due to the next lemma.

Lemma 4.3.3: HAMILTONIAN PATH can be solved in time $\mathcal{O}(1.251^n)$ on subcubic graphs.

Proof. Let $G = (V, E)$ be a subcubic graph. Run the algorithm of [95] to find a Hamiltonian cycle. If it succeeds G clearly also has a Hamiltonian path. If it does not succeed we have to investigate if G has a Hamiltonian path whose end points are not adjacent. Let $u, v \in V(G)$ be two non-adjacent vertices. To check whether G has a Hamiltonian path uPv , we check whether $G' = (V, E')$, where $E' := E \cup \{\{u, v\}\}$, has a Hamiltonian cycle. If G' has maximum degree at most 3, then run the algorithm of [95]. Otherwise, choose a vertex of degree 4, say u , and two neighbors x, z of u distinct from v . As $\{u, v\}$ belongs to every existent Hamiltonian cycle of G' (otherwise G has a Hamiltonian cycle, too), every Hamiltonian cycle of G' avoids $\{u, x\}$ or $\{u, z\}$. Recursively check if $(V, E' \setminus \{\{u, x\}\})$ or $(V, E' \setminus \{\{u, z\}\})$ has a Hamiltonian cycle. This recursion has depth at most 2 since G' has at most 2 vertices of degree 4. The HAMILTONIAN CYCLE algorithm of [95] is executed at most $4(n(n-1)/2 - m)$ times. This algorithm runs in $\mathcal{O}^*(2^{(31/96)n}) \subseteq \mathcal{O}^*(1.2509^n)$ steps. \square

At this point we prove an auxiliary lemma used for the analysis of the forthcoming algorithm.

Lemma 4.3.4: Let T be a spanning tree and $u, v \in V(T)$ two adjacent vertices with $d_T(u) = d_T(v) = 3$ such that $\{u, v\}$ is not a bridge. Then there is a spanning tree $T' \supset (T \setminus \{\{u, v\}\})$ with $|internal(T')| \geq |internal(T)|$ and $d_{T'}(u) = d_{T'}(v) = 2$.

Proof. By removing $\{u, v\}$, T is separated into two parts T_1 and T_2 . The vertices u and v become 2-vertices. As $\{u, v\}$ is not a bridge, there is another edge $e \in E \setminus T$ connecting

T_1 and T_2 . By adding e we lose at most two 2-vertices. Then let $T' := (T \setminus \{\{u, v\}\}) \cup \{e\}$ and it follows that $|internal(T')| \geq |internal(T)|$. \square

4.3.2. Reduction Rules

Let $E' \subseteq E$. Then, $\partial E' := \{\{u, v\} \in E \setminus E' \mid u \in V(E')\}$ are the edges outside E' that have a common end point with an edge in E' and $\partial_V E' := V(\partial E') \cap V(E')$ are the vertices that have at least one incident edge in E' and another incident edge not in E' . In the course of the algorithm we will maintain an acyclic subset of edges F which will be part of the final solution. The following invariant will always be true: $G[F]$ consists of a tree T and a set P of *pending tree edges* (*pt-edges*). Here a pt-edge $\{u, v\} \in F$ is an edge with one end point u of degree 1 and the other end point $v \notin V(T)$, see Figure 4.1(a) where $\{x, v\}$ is a pt-edge. $G[T \cup P]$ will always consist of $1 + |P|$ components.

Next we present a sequence of reduction rules. Note that the order in which they are applied is crucial. We assume that before a rule is applied the preceding ones were carried out exhaustively.

Bridge: If there is a bridge $e \in \partial E(T)$, then add e to F .

DoubleEdge: If there is a double edge delete one of them which is not in F

Cycle: Delete any edge $e \in E$ such that $T \cup \{e\}$ has a cycle.

Deg1: Let $u \in V \setminus V(F)$ with $d(u) = 1$. Then add its incident edge to F .

Pending: If there is a vertex v that is incident to $d_G(v) - 1$ pt-edges, then remove its incident pt-edges.

ConsDeg2: If there are edges $\{v, w\}, \{w, z\} \in E \setminus T$ such that $d_G(w) = d_G(z) = 2$, then delete $\{v, w\}, \{w, z\}$ from G and add the edge $\{v, z\}$ to G .

Deg2: If there is an edge $\{u, v\} \in \partial E(T)$ such that $u \in V(T)$ and $d_G(u) = 2$, then add $\{u, v\}$ to F .

Attach: If there are edges $\{u, v\}, \{v, z\} \in \partial E(T)$ such that $u, z \in V(T)$, $d_T(u) = 2$, $1 \leq d_T(z) \leq 2$, then delete $\{u, v\}$, see Fig. 4.1(a)

Attach2: If there is a vertex $u \in \partial_V E(T)$ with $d_T(u) = 2$ and $\{u, v\} \in E \setminus T$ such that v is incident to a pt-edge, then delete $\{u, v\}$.

Special: If there are two edges $\{u, v\}, \{v, w\} \in E \setminus F$ with $d_T(u) \geq 1$, $d_G(v) = 2$, and w is incident to a pt-edge, then add $\{u, v\}$ to F . See Fig. 4.1(b).

We mention that **ConsDeg2** is the only reduction rule which can create double edges. In this case **DoubleEdge** will delete one of them which is not in F . It will be assured by the reduction rules and the forthcoming algorithm that at most one can be part of F .



Figure 4.1.: Light edges may be not present. Double edges (dotted or solid, resp.) refer to edges which are either T -edges or not, resp. Edges attached to oblongs are pt-edges.

Lemma 4.3.5: The reduction rules stated above are sound.

Proof. Let $T_o \supset F$ be a spanning tree of G with a maximum number of internal vertices. The first four rules are correct for the purpose of connectedness and acyclicity of the evolving spanning tree.

Pending is correct as the other edge incident to v (which will be added to P by a subsequent **Deg1** rule) is a bridge and needs to be in any spanning tree.

ConsDeg2 Let G' be the graph after the reduction rule was applied. We implicitly assume that we can add $\{w, z\}$ to $T_o \supset F$ which is an optimal solution for G . If $\{w, z\} \notin T_o$ then $\{v, w\} \in T_o$. Then we can simply exchange the two edges giving a solution \tilde{T}_o with $\{w, z\} \in \tilde{T}_o$ and $t_2^{\tilde{T}_o} \leq t_2^{T_o}$. By contracting the edge $\{w, z\}$ we receive a solution T'_o such that $|\text{internal}(T'_o)| = |\text{internal}(T_o)| - 1$.

Now suppose T'_o is a spanning tree for G' . If $\{v, z\} \in T'_o$ then let $T_o = T'_o \setminus \{\{v, z\}\} \cup \{\{v, w\}, \{w, z\}\}$. T_o is a spanning tree for G and we have $|\text{internal}(T_o)| = |\text{internal}(T'_o)| + 1$. If $\{v, z\} \notin T'_o$ then by connectivity $d_{T'_o}(z) = 1$. Let $T_o = T'_o \cup \{\{v, w\}\}$ the $|\text{internal}(T_o)| = |\text{internal}(T'_o)| + 1$.

Deg2 Since the preceding reduction rules do not apply, we have $d_G(v) = 3$. Assume u is a leaf in T_o . There is exactly one incident edge, say $\{v, z\}$, $z \neq u$, that is not pending such that it is contained in the single cycle in $G[T \cup \{\{u, v\}\}]$. Define another spanning tree $T'_o \supset F$ by setting $T'_o = (T_o \cup \{\{u, v\}\}) \setminus \{v, z\}$. Since $|\text{internal}(T_o)| \leq |\text{internal}(T'_o)|$, T'_o is also optimal.

Attach If $\{u, v\} \in T_o$ then $\{v, z\} \notin T_o$ due to the acyclicity of T_o and as T_o is connected. Then by exchanging $\{u, v\}$ and $\{v, z\}$ we obtain a solution T'_o with at least as many 2-vertices.

Attach2 Suppose $\{u, v\} \in T_o$. Let $\{v, p\}$ be the pt-edge and $\{v, z\}$ the third edge incident to v (that must exist and is not pending, since **Pending** did not apply). Since **Bridge** did not apply, $\{u, v\}$ is not a bridge. Firstly, suppose $\{v, z\} \in T_o$. Due to Lemma 4.3.4, there is also an optimal solution $T'_o \supset F$ with $\{u, v\} \notin T'_o$. Secondly, assume $\{v, z\} \notin T_o$. Then $T' = (T_o \setminus \{\{u, v\}\}) \cup \{\{v, z\}\}$ is also optimal as u has become a 2-vertex.

Special Suppose $\{u, v\} \notin T_o$. Then $\{v, w\}, \{w, z\} \in T_o$ where $\{w, z\}$ is the third edge incident to w . Let $T'_o := (T_o \setminus \{\{v, w\}\}) \cup \{\{u, v\}\}$. In T'_o , w is a 2-vertex and hence T'_o is also optimal. \square

4.3.3. Triangles

In this section we will argue that triangles in G can be contracted to one vertex such that at least one optimal solution is preserved.

Triangle: Let a_1, a_2, a_3 form a triangle. Then contract them into one vertex r .

If $a_1, a_2, a_3 \in V$ form a triangle D then let $Q_D := \{v \in V(D) \mid |N_T(v) \setminus D| = 1\}$ for a given spanning tree T . Vertices a, b, c form an ℓ -triangle if $|Q_D| = \ell$.

Lemma 4.3.6: Let \hat{T} be an optimal spanning tree and let $D = \{a_1, a_2, a_3\} \subseteq V$ form a triangle. Then we can assume the following:

- a) If D is a 1-triangle then $|D \cap L(\tilde{T})| = 1$, see Figure 4.2(g)
- b) If D is a 2- or
- c) a 3-triangle then, w.l.o.g, $|D \cap L(\tilde{T})| = 0$, see Figures 4.2(h) and 4.2(i).

Proof. Due to Lemma 4.3.2 we have $|D \cap L(\hat{T})| \leq 1$ (\clubsuit).

a) W.l.o.g. $|N_T(a_3) \setminus D| = 1$ then a_1 and a_2 must be attached to a_3 . They must be attached as in Figure 4.2(g) by (\clubsuit).

b) Suppose D has one \hat{T} -leaf. W.l.o.g, $|N_T(a_2) \setminus D| = |N_T(a_1) \setminus D| = 1$ and with $a_3 \in L(\hat{T})$ being attached to a_1 . Due to (\clubsuit) we have $\{a_2, a_1\} \in E(\hat{T})$, see Figure 4.2(a). Then exchange $\{a_2, a_1\}$ with $\{a_2, a_3\}$ and obtain the situation in Figure 4.2(c) with one more 2-vertex.

c) W.l.o.g., suppose $a_3 \in L(\hat{T})$ and thus $\{a_1, a_3\}, \{a_2, a_3\} \notin E(\hat{T})$, but $\{a_2, a_1\} \in E(\hat{T})$ by (\clubsuit), see Figure 4.2(b). Exchange $\{a_2, a_1\}$ with $\{a_2, a_3\}$ and $\{a_3, v\}$ with $\{a_1, a_3\}$ (\star) arriving at a spanning tree \hat{T} with the situation in Figure 4.2(c). The vertices a_1, a_2, a_3 are now 2-vertices but note that v could loose this property and become a \hat{T} -leaf. Observe that if in this case v had a \hat{T} -leaf as a neighbor (\clubsuit) (possibly they occur both in a further triangle D') then due Lemma 4.3.2 the number of internal vertices could be increased, a contradiction .

We also have to show that a), b) and c) are not violated with respect to D' in \hat{T} where $v \in V(D')$. Observe that D' cannot be a 1-triangle in \hat{T} due to connectivity. If D' is a 2-triangle in \hat{T} then due optimality reasons and (\clubsuit) D' must be of the form as in Figure 4.2(h) with, w.l.o.g, $v = a_3$. Thus, after the exchange in (\star) D' is a 1-triangle and obeys a).

If D' is 3-triangle in T then due to optimality and by (\clubsuit) and (\clubsuit) it is of the form as in Figure 4.2(i) (as the situation in Figure 4.2(b) is not possible by (\clubsuit)).

4.3. Subcubic Maximum Internal Spanning Tree

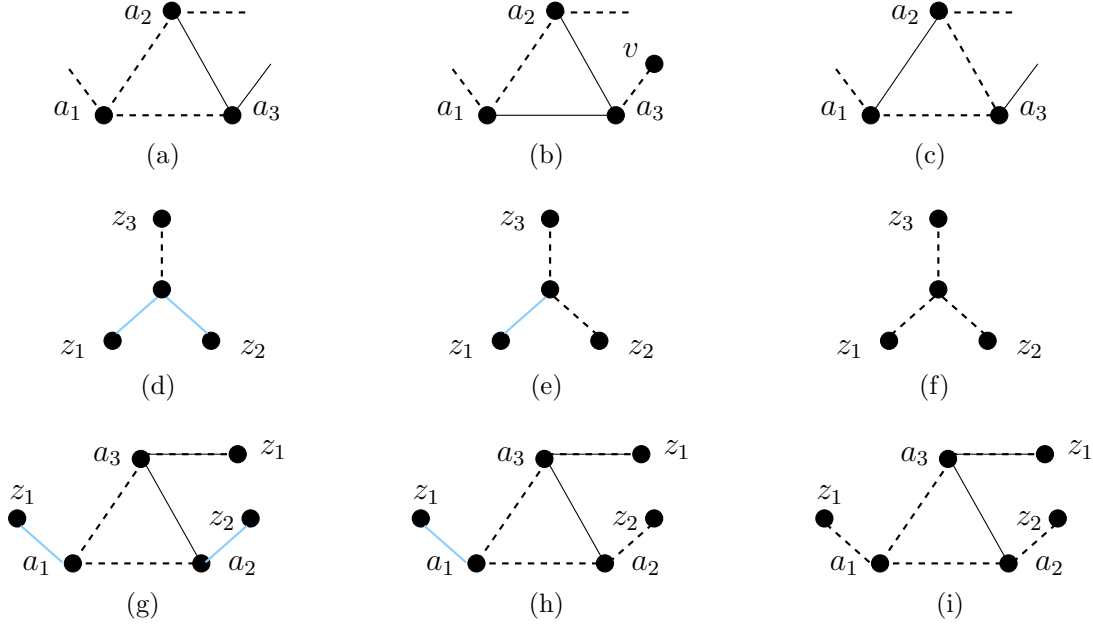


Figure 4.2.: Dotted edges belong to the tree T . Light edges maybe present or not.

If $x = a_1$ in Figure 4.2(i) then after the modifications in (\star) D' is a 2-Triangle obeying b). If, w.l.o.g., $x = a_3$ then after the modification in (\star) also exchange the edge $\{a_1, a_2\}$ by $\{a_3, a_2\}$ (i.e., delete $\{a_1, a_2\}$ from \hat{T} and add $\{a_3, a_2\}$ to \hat{T}). Then D' is a 2-triangle obeying b). \square

The next lemma shows the soundness of the **Triangle** rule.

Lemma 4.3.7: Let G' be the graph evolving from a **Triangle** application on a triangle D in a graph G .

1. G' has a spanning tree T'_o with at least k internal vertices $\Rightarrow G$ has a spanning tree T_o with at least $k + 2$ internal vertices.
2. G has an optimal spanning tree T_o with at least $k + 2$ internal vertices $\Rightarrow G'$ has a spanning tree T'_o with at least k internal vertices.

Proof. 1. Let r be the vertex in G' created by **Triangle** from a triangle D in G .

We will consider the three different cases for the number of T -edges neighboring r , which can be seen in Figures 4.2(d), 4.2(e) and 4.2(f), resp.. These situations can be transformed to the ones seen in Figures 4.2(g), 4.2(h) and 4.2(i), resp.. Thus, we obtain a spanning tree for G with a newly created triangle D .

Let us mention that it is possible that $|E(D) \cap E(T_o)| = 1$, i.e., when **Triangle** was applied in G to D already some edge of $E(D)$ was contained in T . ($|E(D) \cap E(T_o)| \geq 2$ impossible due to acyclicity and **Cycle**). This is not a problem for the transformations from Figure 4.2(d) to Figure 4.2(g) and from Figure 4.2(f) to Figure 4.2(i) as any edge pre-selection can be extended to the shown

situations. It only will lead to a problem when we transform from Figure 4.2(e) into Figure 4.2(h) and we are faced with the situation in Figure 4.2(a) after reconstructing the triangle. Then regardlessly transform to Figure 4.2(h) as the current partial solution never can be extended to an optimal one (Lemma 4.3.6 and Figure 4.2(a)).

2. D has to be an ℓ -triangle with $1 \leq \ell \leq 3$. Note that due to Lemma 4.3.6.a) in a 1-triangle exactly one 1-vertex appears and thus we have the situation in Figure 4.2(g). Due to Lemma 4.3.6.b) and c) 2- and 3-triangles contain no 1-vertices. Thus, we have the situations in Figures 4.2(h) and 4.2(i). Hence, these cases are exhaustive for any optimal solution. They can be transformed to the situations in Figures 4.2(d), 4.2(e) and 4.2(f), respectively, by contracting D . □

The **Triangle** rule appears to be quite powerful. Nevertheless, in our run time analysis of the forthcoming algorithm this rule is of no importance. No asymptotical run time improvement can be achieved. Regardless, we think this rule is quite effective for practical purposes. If used this rule should directly succeed **Special** in priority.

4.3.4. The Algorithm

The algorithm we describe here is recursive. It constructs a set F of edges which are selected to be in every spanning tree considered in the current recursive step. The algorithm chooses edges and considers all relevant choices for adding them to F or removing them from G . It selects these edges based on priorities chosen to optimize the running time analysis. Moreover, the set F of edges will always be the union of a tree T and a set of edges P that are not incident to the tree and have one end point of degree 1 in G (pt-edges). We do not explicitly write in the algorithm that edges move from P to T whenever an edge is added to F that is incident to both an edge of T and an edge of P . To maintain the connectivity of T , the algorithm explores edges in the set $\partial E(T)$ to grow T .

If $|V| > 2$ every spanning tree T must have a vertex v with $d_T(v) \geq 2$. Thus initially the algorithm creates an instance for every vertex v and every possibility that $d_T(v) \geq 2$. Due to the degree constraint there are no more than $4n$ instances. After this initial phase, the algorithm proceeds as described in Algorithm 1.

4.3.5. An Exact Analysis of the Algorithm

By a *Measure & Conquer* analysis taking into account the degrees of the vertices, their number of incident edges that are in F , and to some extent the degrees of their neighbors, we obtain the following result.

Theorem 4.3.8: MIST can be solved in time $\mathcal{O}(1.8669^n)$ on subcubic graphs.

Let $D_2 := \{v \in V \mid d_G(v) = 2, d_F(v) = 0\}$, $D_3^\ell := \{v \in V \mid d_G(v) = 3, d_F(v) = \ell\}$ and $D_3^{2*} := \{v \in D_3^2 \mid N_G(v) \setminus N_F(v) = \{u\} \text{ and } d_G(u) = 2\}$. Then the measure we use for

Algorithm 1: An Algorithm solving MAXIMUM INTERNAL SPANNING TREE

Data: A subcubic graph $G = (V, E)$ a tree $T \subseteq E$.

Result: A spanning tree T' with the maximum number of internal vertices such that $T' \supseteq T$.

- 1 Carry out each reduction rule exhaustively in the given order (until no rule applies).
 - 2 If $\partial E(T) = \emptyset$ and $V \neq V(T)$, then G is not connected and does not admit a spanning tree. Ignore this branch.
 - 3 If $\partial E(T) = \emptyset$ and $V = V(T)$, then return T .
 - 4 Select $\{a, b\} \in \partial E(T)$ with $a \in V(T)$ according to the following priorities (if such an edge exists):
 - 5 **case** a) *There is an edge $\{b, c\} \in \partial E(T)$.*
 - 6 b) $d_G(b) = 2$.
 - 7 c) b is incident to a *pt-edge*.
 - 8 d) $d_T(a) = 1$.
 - 9 | Recursively solve the two instances where $\{a, b\}$ is added to F or removed from G respectively, and return the spanning tree with most internal vertices of the two returned ones.
 - 10 **otherwise**
 - 11 | Select $\{a, b\} \in \partial E(T)$ with $a \in V(T)$. Let c, x be the other two neighbors of b . Recursively solve three instances where
 - 12 (i) $\{a, b\}$ is removed from G ,
 - 13 (ii) $\{a, b\}$ and $\{b, c\}$ are added to F and $\{b, x\}$ is removed from G , and
 - 14 (iii) $\{a, b\}$ and $\{b, x\}$ are added to F and $\{b, c\}$ is removed from G .
 - 15 | Return the spanning tree with most internal vertices of the three returned ones.
-

our running time bound is

$$\mu(G) = \omega_2 \cdot |D_2| + \omega_3^1 \cdot |D_3^1| + \omega_3^2 \cdot |D_3^2 \setminus D_3^{2*}| + |D_3^0| + \omega_3^{2*} \cdot |D_3^{2*}|$$

with the weights $\omega_2 = 0.3193$, $\omega_3^1 = 0.6234$, $\omega_3^2 = 0.3094$ and $\omega_3^{2*} = 0.4144$.

Let $\Delta_3^0 := \Delta_3^{0*} := 1 - \omega_3^1$, $\Delta_3^1 := \omega_3^1 - \omega_3^2$, $\Delta_3^{1*} := \omega_3^1 - \omega_3^{2*}$, $\Delta_3^2 := \omega_3^2$, $\Delta_3^{2*} := \omega_3^{2*}$ and $\Delta_2 = 1 - \omega_2$. We define $\Delta_3^i := \min\{\Delta_3^i, \Delta_3^{i*}\}$ for $1 \leq i \leq 2$, $\Delta_m^\ell = \min_{0 \leq j \leq \ell} \{\Delta_3^j\}$, and $\tilde{\Delta}_m^\ell = \min_{0 \leq j \leq \ell} \{\tilde{\Delta}_3^j\}$.

The proof of the theorem uses the following result.

Lemma 4.3.9: None of the reduction rules increase μ for the given weights.

Proof. **Bridge**, **Deg1**, **Deg2** and **Special** add edges to T . Due to the definitions of D_3^ℓ and D_3^{2*} and the choice of the weights it can be seen that μ only decreases. In case of **Pending** due to **Bridge** and **Deg1** note that for the vertex v from the reduction rule we have $d(v) - 1 \leq d_F(v)$, i.e., at most one edge incident with v is not in F . Thus, v has at least weight zero in μ . After the application of **Pending** it has weight zero. In **ConsDeg2** a degree 2 vertex is removed from the graph and hence μ decreases by ω_2 . It is also easy to see that the deletion of edges $\{u, v\} \notin T$ with $d_T(u) \geq 1$ and $d_T(v) = 0$ is safe with respect to u and v . The weight of u and v can only decrease due to this (as it can be seen in case of the rules **DoubleEdge**, **Cycle**, **Attach** and **Attach2**). Nevertheless, the rules which delete such kind of edges might cause that a $v \in D_3^2 \setminus D_3^{2*}$ will be in D_3^{2*} afterwards. Thus, we have to prove that in this case the overall reduction is enough. A sufficient criterion that the described scenario takes place is if a degree 2 vertex x is created with $x \notin \partial_V(T)$. **Cycle** may create vertices of degree 2, but all of them are in $\partial_V(T)$. The only further reduction rule which may create vertices of degree 2 not in $\partial_V(T)$ is **Attach** when $d(v) = 3$ (where v is mentioned in the rule definition). The minimum reduction then is $\omega_3^2 + \Delta_2 - 2 \cdot (\omega_3^{2*} - \omega_3^2) > 0$. It can be checked that no other reduction rule creates degree 2 vertices not contained in $\partial_V(T)$. \square

Proof. (of Theorem 4.3.8) As the algorithm deletes edges or moves edges from $E \setminus F$ to F , cases 1–3 do not contribute to the exponential function in the running time of the algorithm. It remains to analyze cases 4 and 5, which we do now. Note that after applying the reduction rules exhaustively, we have that for all $v \in \partial_V E(T)$, $d_G(v) = 3$ (**Deg2**) and for all $u \in V$, $d_P(u) \leq 1$ (**Pending**).

- 4.(a) Obviously, $\{a, b\}, \{b, c\} \in E \setminus T$, and there is a vertex d such that $\{c, d\} \in T$; see Figure 4.3(a). We have $d_T(a) = d_T(c) = 1$ due to the reduction rule **Attach**. We consider three cases.

$d_G(b) = 2$. When $\{a, b\}$ is added to F , **Cycle** deletes $\{b, c\}$. We get an amount of ω_2 and ω_3^1 as b drops out of D_2 and c out of D_3^1 (**Deg2**). Also a will be removed from D_3^1 and added to D_3^2 which amounts to a reduction of at least $\tilde{\Delta}_3^1$. When $\{a, b\}$ is deleted, $\{b, c\}$ is added to T (**Bridge**). By a symmetric argument we get a reduction of $\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1$ as well. In total this yields a $(\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1, \omega_2 + \omega_3^1 + \tilde{\Delta}_3^1)$ -branch.

$d_G(b) = 3$ and there is one pt-edge incident to b . Adding $\{a, b\}$ to F decreases the measure by $\tilde{\Delta}_3^1$ (from a) and $2\omega_3^1$ (deleting $\{b, c\}$, then **Deg2** on c). By Deleting $\{a, b\}$ we decrease μ by $2\omega_3^1$ and by $\tilde{\Delta}_3^1$ (from c). This amounts to a $(2\omega_3^1 + \tilde{\Delta}_3^1, 2\omega_3^1 + \tilde{\Delta}_3^1)$ -branch.

$d_G(b) = 3$ and no pt-edge is incident to b . Let $\{b, z\}$ be the third edge incident to b . In the first branch the measure drops by at least $\omega_3^1 + \tilde{\Delta}_3^1$ from c and a (**Deg2**), 1 from b (**Deg2**). In the second branch we get $\omega_3^1 + \Delta_2$. Observe that we also get an amount of at least $\tilde{\Delta}_m^1$ from $q \in N_T(a) \setminus \{b\}$ if $d_G(q) = 3$. If $d_G(q) = 2$ we get ω_2 . It results a $(\omega_3^1 + \tilde{\Delta}_3^1 + 1, \omega_3^1 + \Delta_2 + \min\{\omega_2, \tilde{\Delta}_m^1\})$ -branch.

Note that from this point on, for all $u, v \in V(T)$ there is no $z \in V \setminus V(T)$ with $\{u, z\}, \{z, v\} \in E \setminus T$.

- 4.(b) As the previous case does not apply, the other neighbor c of b has $d_T(c) = 0$, and $d_G(c) \geq 2$ (**Pending**). Additionally, observe that $d_G(c) = 3$ due to **ConsDeg2** and that $d_P(c) = 0$ due to **Special**, see Figure 4.3(b). We consider two subcases.

$d_T(a) = 1$. When we add $\{a, b\}$ to F , then $\{b, c\}$ is also added due to **Deg2**. The reduction is at least $\tilde{\Delta}_3^1$ from a , ω_2 from b and Δ_3^0 from c . When $\{a, b\}$ is deleted, $\{b, c\}$ becomes a pt-edge. There is $\{a, z\} \in E \setminus T$ with $z \neq b$, which is subject to a **Deg2** reduction rule. We get at least ω_3^1 from a , ω_2 from b , Δ_3^0 from c and $\min\{\omega_2, \tilde{\Delta}_m^1\}$ from z . This is a $(\tilde{\Delta}_3^1 + \Delta_3^0 + \omega_2, \omega_3^1 + \Delta_3^0 + \omega_2 + \min\{\omega_2, \tilde{\Delta}_m^1\})$ -branch.

$d_T(a) = 2$. Similarly, we obtain a $(\Delta_3^{2*} + \omega_2 + \Delta_3^0, \Delta_3^{2*} + \omega_2 + \Delta_3^0)$ -branch.

- 4.(c) In this case, $d_G(b) = 3$ and there is one pt-edge attached to b , see Figure 4.3(c). Note that $d_T(a) = 2$ can be ruled out due to **Attach2**. Thus, $d_T(a) = 1$. Let $z \neq b$ be such that $\{a, z\} \in E \setminus T$. Due to the priorities, $d_G(z) = 3$. We distinguish between the cases where c , the other neighbor of b , is incident to a pt-edge or not.

$d_P(c) = 0$. First suppose $d_G(c) = 3$. Adding $\{a, b\}$ to F allows a reduction of $2\tilde{\Delta}_3^1$ (due to case 4.(b) we can exclude Δ_3^{1*}). Deleting $\{a, b\}$ implies that we get a reduction from a and b of $2\omega_3^1$ (**Deg2** and **Pending**). As $\{a, z\}$ is added to F we reduce $\mu(G)$ by at least $\tilde{\Delta}_m^1$ as the state of z changes. Now due to **Pending** and **Deg1** we include $\{b, c\}$ and get Δ_3^0 from c . We have at least a $(2\tilde{\Delta}_3^1, 2\omega_3^1 + \tilde{\Delta}_m^1 + \Delta_3^0)$ -branch.

If $d_G(c) = 2$ we consider the two cases for z also. These are $d_P(z) = 1$ and $d_P(z) = 0$. The first entails $(\omega_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \tilde{\Delta}_3^1 + \omega_2 + \tilde{\Delta}_m^2)$. Note that when we add $\{a, b\}$ we trigger **Attach2** and by deleting $\{a, b\}$ all edges incident to c become bridges. The second is a $(\Delta_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \Delta_3^0 + \omega_2 + \tilde{\Delta}_m^2)$ -branch.

$d_P(c) = 1$. Let $d \neq b$ be the other neighbor of c that does not have degree 1. When $\{a, b\}$ is added to F , $\{b, c\}$ is deleted by **Attach2** and $\{c, d\}$ becomes a pt-edge (**Pending** and **Deg1**). The changes on a incur a measure decrease of Δ_3^{1*} and those on b, c a measure decrease of $2\omega_3^1$. When $\{a, b\}$ is deleted, $\{a, z\}$ is added to F (**Deg2**) and $\{c, d\}$ becomes a pt-edge by two applications

of the **Pending** and **Deg1** rules. Thus, the decrease of the measure is at least $3\omega_3^1$ in this branch. In total, we have a $(\Delta_3^{1*} + 2\omega_3^1, 3\omega_3^1)$ -branch here.

4.(d) Now, $d_G(b) = 3$, b is not incident to a pt-edge, and $d_T(a) = 1$. See Figure 4.3(c). There is also some $\{a, z\} \in E \setminus T$ such that $z \neq b$. Note that $d_T(z) = 0$, $d_G(z) = 3$ and $d_P(z) = 0$. Otherwise either **Cycle** or cases 4.(b) or 4.(c) would have been triggered. From the addition of $\{a, b\}$ to F we get $\Delta_3^1 + \Delta_3^0$ and from its deletion ω_3^1 (from a via **Deg2**), Δ_2 (from b) and at least Δ_3^0 from z and thus, a $(\Delta_3^1 + \Delta_3^0, \omega_3^1 + \Delta_2 + \Delta_3^0)$ -branch.

5. See Figure 4.3(d). The algorithm branches in the following way: 1) Delete $\{a, b\}$, 2) add $\{a, b\}, \{b, c\}$, and delete $\{b, x\}$, 3) add $\{a, b\}, \{b, x\}$ and delete $\{b, c\}$. Observe that these cases are sufficient to preserve an optimal solution. We can disregard the case when b is a leaf (i.e., $\{a, b\} \in T$ and $\{b, c\}, \{b, x\} \notin T$). Note that we can obtain a no worse solution $T^* := (T \setminus \{a, b\}) \cup \{b, x\}$ where $\{a, b\} \notin T$ (which refers to case a)). Observe that a is a 2-vertex in T^* and T^* is indeed a tree by $\{b, c\}, \{b, x\} \notin T$.

Due to Lemma 4.3.4 we also disregard the case when b is a 3-vertex as $\{a, b\}$ is not a bridge. Thus by branching in this manner we find at least one optimal solution. The reduction in the first branch is at least $\omega_3^2 + \Delta_2$. We get an additional amount of ω_2 if $d(x) = 2$ or $d(c) = 2$ from **ConsDeg2**. In the second branch we have to consider also the vertices c and x . We distinguish between three situations for $h \in \{c, x\}$:

$$\alpha) d_G(h) = 2.$$

$$\beta) d_G(h) = 3, d_P(h) = 0.$$

$$\gamma) d_G(h) = 3, d_P(h) = 1.$$

We will only analyze branch 2) as 3) is symmetric. We first get a reduction of $\omega_3^2 + 1$ from a and b . We reduce μ due to deleting $\{b, x\}$ by:

$$\alpha) \omega_2 + \tilde{\Delta}_m^2 \text{ by } \mathbf{Pending}.$$

$$\beta) \Delta_2.$$

$$\gamma) \omega_3^1 + \tilde{\Delta}_m^2 \text{ by } \mathbf{Pending} \text{ and } \mathbf{Deg1}.$$

Next we examine the amount by which μ will be decreased by adding $\{b, c\}$ to F . We distinguish between the cases α, β and γ :

$$\alpha) \omega_2 + \tilde{\Delta}_m^2.$$

$$\beta) \Delta_3^0.$$

$$\gamma) \tilde{\Delta}_3^1.$$

For $h \in \{c, x\}$ and $W \in \{\alpha, \beta, \gamma\}$ let 1_W^h be the indicator function which is set to one if we have situation W at vertex h . Otherwise it is zero. Now the branching vector can be stated the following way :

$$(\omega_3^2 + \Delta_2 + (1_\alpha^x + 1_\alpha^c) \cdot \omega_2,$$

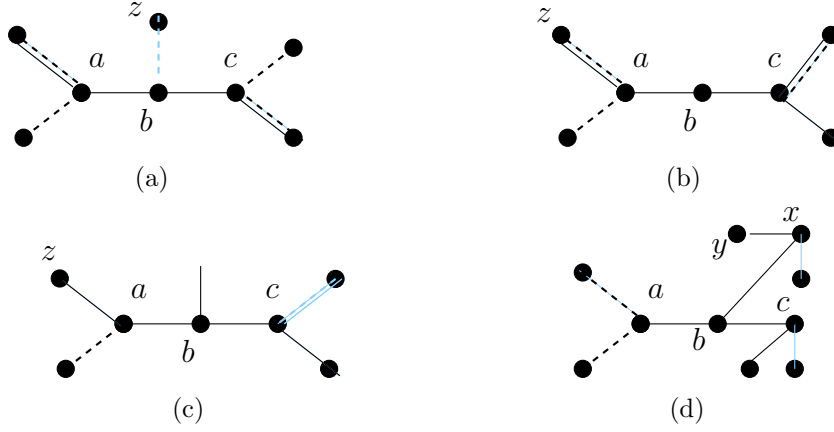


Figure 4.3.: Light edges may be not present. Double edges (dotted or solid, resp.) refer to edges which are either T -edges or not, resp. Edges attached to oblongs are pt-edges.

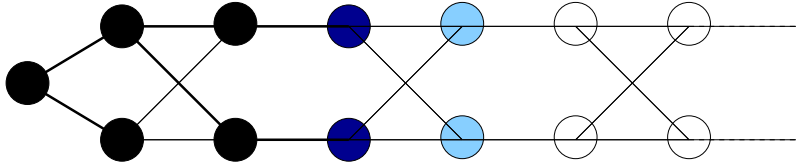


Figure 4.4.: A sketch of a lower bound example graph family for our MIST algorithm.

$$\omega_3^2 + 1 + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_2 + 1_\gamma^x \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_3^0 + 1_\gamma^c \cdot \tilde{\Delta}_3^1),$$

$$\omega_3^2 + 1 + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_2 + 1_\gamma^c \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_3^0 + 1_\gamma^x \cdot \tilde{\Delta}_3^1)$$

The amount of $(1_\alpha^x + 1_\alpha^c) \cdot \omega_2$ comes from possible applications of **ConsDeg2**.

Observe that every instance created by branching is smaller than the original instance in terms of μ . Together with Lemma 4.3.9 we see that every reduction step of the algorithm never increases μ . Now if we evaluate the upper bound for every given branching tuple for the given weights we can conclude that MIST can be solved in time $\mathcal{O}^*(1.8669^n)$ on subcubic graphs. \square

4.3.5.1. How Good is Our Run Time Estimate?

Clearly, such a question cannot be answered in an absolute fashion, since this immediately touches the famous \mathcal{P} - \mathcal{NP} problem. However, it might be possible to provide lower bound examples, i.e., graph (families) that require $\Omega(c^n)$ running time of *our algorithm* on graphs of order n .

Theorem 4.3.10: There is a family of graphs s.t. our algorithm proposed for solving MIST needs $\Omega(\sqrt[4]{2}^n)$ time, with $\sqrt[4]{2} \geq 1.189$.

Proof. Instead of giving a formal description of the graph family, consider Fig. 4.4.

Assume that the bold edges are already in the tree. Then, the dark blue vertices are the current leaves of the tree. W.l.o.g., the algorithm would consider to take one of the two edges connecting the upper dark blue vertex with light blue vertices into the tree. If the selected edge is not inserted into the tree, then the chosen dark blue vertex will become of degree two and hence the tree expands due to the degree two rule. In either case, the degree two rule will then integrate three more vertices into the tree. All in all, four vertices have been put into the tree in either branch, and we arrive at a situation that is basically the same as the one we started with, so that the argument repeats. Hence, when we start with a graph with $4n + 2$ vertices, then there will be one branching scenario that creates a binary search tree of height n . This shows the claim. \square

4.3.6. A Parameterized Analysis of the Algorithm

For general graphs, the smallest known kernel has size $3k$. This can be easily improved to $2k$ for subcubic graphs.

Lemma 4.3.11: MIST on subcubic graphs has a $2k$ -kernel.

Proof. Compute an arbitrary spanning tree T . If it has at least k internal vertices, answer YES. Otherwise, $t_3^T + t_2^T < k$. Then, by Proposition 4.3.1, $t_1^T < k + 2$. Thus, $|V| \leq 2k$. \square

Applying the algorithm of Theorem 4.3.8 on this kernel for subcubic graphs shows the following result.

Corollary 4.3.12: Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $3.4854^k n^{\mathcal{O}(1)}$.

However, we can achieve a faster parameterized running time by applying a *Measure&Conquer* analysis which is customized to the parameter k . We would like to put forward that our use of the technique of *Measure&Conquer* for a parameterized algorithm analysis goes beyond previous work as our measure is not restricted to differ from the parameter k by just a constant. We first demonstrate our idea with a simple analysis.

Theorem 4.3.13: Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $2.7321^k n^{\mathcal{O}(1)}$.

Proof. Note that the assumption that G has no Hamiltonian path can still be made due to the $2k$ -kernel of Lemma 4.3.11: the running time of the Hamiltonian path algorithm is $1.251^{2k} n^{\mathcal{O}(1)} = 1.5651^k n^{\mathcal{O}(1)}$. The running time analysis of our algorithm relies on the following measure:

$$\kappa := \kappa(G, F, k) := k - \omega \cdot |X| - |Y| - \tilde{k},$$

where $X := \{v \in V \mid d_G(v) = 3, d_T(v) = 2\}$, $Y := \{v \in V \mid d_G(v) = d_T(v) \geq 2\}$ and $0 \leq \omega \leq 1$. Let $U := V \setminus (X \cup Y)$ and note that k in the definition of κ never changes

in any recursive call of the algorithm. The variable \tilde{k} counts how many times the reduction rules **ConsDeg2** and **Pending** have been applied upon reaching the current search tree node. Note that by an **ConsDeg2** the number of internal vertices of the original instance goes up by one. If **Pending** is applied a vertex $v \in Y$ will be moved to U in the evolving instance G' even though v will be internal in the original instance G . This would increase κ if \tilde{k} would not balance this. Note that a vertex which has already been decided to be internal, but that still has an incident edge in $E \setminus T$, contributes a weight of $1 - \omega$ to the measure. Or equivalently, such a vertex has been only counted by ω . Consider the algorithm described earlier, with the only modification that that the algorithm keeps track of κ and that the algorithm stops and answers **YES** whenever $\kappa \leq 0$. None of the reduction and branching rules increases κ . The explicit proof for this will be skipped as it is subsumed by Lemma 4.3.16 which deals with a refined measure. We have that $0 \leq \kappa \leq k$ at any time of the execution of the algorithm.

In step 4, whenever the algorithm branches on an edge $\{a, b\}$ such that $d_T(a) = 1$ (w.l.o.g., we assume that $a \in V(T)$), the measure decreases by at least ω in one branch, and by at least 1 in the other branch. We speak of a $(\omega, 1)$ -branch. To see this, it suffices to look at vertex a . Due to **Deg2**, $d_G(a) = 3$. When $\{a, b\}$ is added to F , vertex a moves from the set U to the set X . When $\{a, b\}$ is removed from G , a subsequent application of the **Deg2** rule adds the other edge incident to a to F , and thus, a moves from U to Y .

Still in step 4, let us consider the case where $d_T(a) = 2$. Then condition (b) ($d_G(b) = 2$) of step 4 must hold, due to the preference of the reduction and branching rules: condition (a) is excluded due to reduction rule **Attach**, (c) is excluded due to **Attach2** and (d) is excluded due to its condition that $d_T(a) = 1$. When $\{a, b\}$ is added to F , the other edge incident to b is also added to F by a subsequent **Deg2** rule. Thus, a moves from X to Y and b from U to Y for a measure decrease of $(1 - \omega) + 1 = 2 - \omega$. When $\{a, b\}$ is removed from G , a moves from X to Y for a measure decrease of $1 - \omega$. Thus, we have a $(2 - \omega, 1 - \omega)$ -branch.

In step 5, $d_T(a) = 2$, $d_G(b) = 3$, and $d_F(b) = 0$. Vertex a moves from X to Y in each branch and b moves from U to Y in the two latter branches. In total we have a $(1 - \omega, 2 - \omega, 2 - \omega)$ -branch. By setting $\omega = 0.45346$ and evaluating the branching numbers, the proof follows. \square

This analysis can be improved by also measuring vertices of degree 2 and vertices incident to pt-edges differently.

Theorem 4.3.14: Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $2.1364^k n^{\mathcal{O}(1)}$.

The proof of this theorem follows the same lines as the previous one, except that we consider a more detailed measure:

$$\kappa := \kappa(G, F, k) := k - \omega_1 \cdot |X| - |Y| - \omega_2 |Z| - \omega_3 |W| - \tilde{k}, \text{ where}$$

- $X := \{v \in V \mid d_G(v) = 3, d_T(v) = 2\}$ is the set of vertices of degree 3 that are incident to exactly 2 edges of T ,

- $Y := \{v \in V \mid d_G(v) = d_T(v) \geq 2\}$ is the set of vertices of degree at least 2 that are incident to only edges of T ,
- $W := \{v \in V \setminus (X \cup Y) \mid d_G(v) \geq 2, \exists u \in N(v) \text{ st. } d_G(u) = d_F(u) = 1\}$ is the set of vertices of degree at least 2 that have an incident pt-edge, and
- $Z := \{v \in V \setminus W \mid d_G(v) = 2, N[v] \cap (X \cup Y) = \emptyset\}$ is the set of degree 2 vertices that do not have a vertex of $X \cup Y$ in their closed neighborhood, and are not incident to a pt-edge.

We immediately set $\omega_1 := 0.5485, \omega_2 := 0.4189$ and $\omega_3 := 0.7712$. Let $U := V \setminus (X \cup Y \cup Z \cup W)$. We first have to show that the algorithm can be stopped whenever the measure drops to 0 or less.

Lemma 4.3.15: Let $G = (V, E)$ be a connected graph, k be an integer and $F \subseteq E$ be a set of edges that can be partitioned into a tree T and a set of pending edges P . If none of the reduction rules applies to this instance and $\kappa(G, F, k) \leq 0$, then G has a spanning tree $T^* \supseteq F$ with at least k internal nodes.

Proof. Since the vertices in $X \cup Y$ are internal in any spanning tree containing F , it is sufficient to show that there exists a spanning tree $T^* \supseteq F$ that has at least $\omega_2|Z| + \omega_3|W|$ more internal vertices than T .

The spanning tree T^* is constructed as follows:

1. Greedily add a subset of edges $A \subseteq E \setminus F$ to F to obtain a spanning tree T' of G .
2. While there exists $v \in Z$ with neighbors u_1 and u_2 such that $d_{T'}(v) = d_{T'}(u_1) = 1$ and $d_{T'}(u_2) = 3$, then set $A := (A \setminus \{v, u_2\}) \cup \{u_1, v\}$ (*).

This procedure finishes in polynomial time as the number of internal vertices increases each time such a vertex is found. Call the resulting spanning tree T^* .

By connectivity of a spanning tree, we have:

Fact 1: If $v \in W$, then v is internal in T^* .

Note that $F \subseteq T^*$ as no vertex of Z is incident to an edge of F . By the construction of T^* , we have the following (*).

Fact 2: If u, v are two adjacent vertices in G but not in T^* , such that $v \in Z$ and u, v are leafs in T^* , then v 's other neighbor has T^* -degree 2.

Let $Z_\ell \subseteq Z$ be the subset of vertices of Z that are leafs in T^* and let $Z_i := Z \setminus Z_\ell$. As $F \subseteq T^*$ and by Fact 1, all vertices of $X \cup Y \cup W \cup Z_i$ are internal in T^* . Let P denote the subset of vertices of $N(Z_\ell)$ that are internal in T^* . As P might intersect with W and for $u, v \in Z_\ell$, $N(u)$ and $N(v)$ might intersect (but $u \notin N(v)$ because of **ConsDeg2**), we assign an initial potential of 1 to vertices of P . By definition, $P \cap (X \cup Y) = \emptyset$. Thus the number of internal vertices in T^* is at least $|X| + |Y| + |Z_i| + |P \cup W|$. To finish the proof of the claim, we show that $|P \cup W| = |W \setminus P| + |P \cap W| + |P \setminus W| \geq \omega_2|Z_i| + \omega_3|W|$.

4.3. Subcubic Maximum Internal Spanning Tree

Decrease the potential of each vertex in $P \cap W$ by ω_3 . Then, for each vertex $v \in Z_\ell$, decrease the potential of each vertex in $P_v = N(v) \cap P$ by $\omega_2/|P_v|$. We show that the potential of each vertex in P remains positive. Let $u \in P$ and $v_1 \in Z_\ell$ be a neighbor of u . Note that $d_{T^*}(v_1) = 1$. We distinguish two cases based on u 's tree-degree in T^* .

$$d_{T^*}(u) = 2$$

$u \in W$: Then by connectivity $\{u, v_1\} \notin T^*$ and u is incident to only one vertex out of Z_ℓ , namely v_1 . Again by connectivity $h \in N(v_1) \setminus \{u\}$ is a internal vertex. Thus, the potential is $1 - \omega_3 - \omega_2/2 \geq 0$.

$u \notin W$: u is incident to at most 2 vertices of Z_ℓ (by connectivity of T^*), its potential remains thus positive as $1 - 2\omega_2 \geq 0$.

$$d_{T^*}(u) = 3$$

$u \in W$: Because $u \in W$ is incident to a pt-edge, it has one neighbor in Z_ℓ (connectivity of T^*), which has only internal neighbors (by Fact 2). The potential of u is thus $1 - \omega_3 - \omega_2/2 \geq 0$.

$u \notin W$: u has at most two neighbors in Z_ℓ , and both of them have only inner neighbors due to Fact 2. As $1 - 2\omega_2/2 \geq 0$, u 's potential remains positive.

Thus, we have shown that $|P \cap W| + |P \setminus W| \geq \omega_2|Z_\ell| + \omega_3|W|$ from which the claim follows. \square

We also show that reducing an instance does not increase its measure.

Lemma 4.3.16: Let (G', F', k') be an instance resulting from the exhaustive application of the reduction rules to an instance (G, F, k) . Then, $\kappa(G', F', k') \leq \kappa(G, F, k)$.

Proof. **Cycle:** If the reduction rule **Cycle** is applied to (G, F, k) , then an edge in $\partial E(T)$ is removed from the graph. Then, the parameter k stays the same, and either each vertex remains in the same set among X, Y, Z, W, U , or one or two vertices move from X to Y , which we denote shortly by the status change of a vertex u : $\{X\} \rightarrow \{Y\}(\omega_1 - 1)$. The value of this status change is $(-1) - (-\omega_1) \leq 0$. As the value of the status change is non-positive, it does not increase the measure.

DoubleEdge Suppose between u and v is a double edge. Then as mentioned before at most one of them belongs to T . The possible transitions for u (and v) are $\{X\} \rightarrow \{Y\}(\omega_1 - 1)$ if $d_T(u) = 2$ and $d_G(u) = 3$, $\{U\} \rightarrow \{U\}(0)$ if $d_T(u) = 1$ and $d_G(u) = 3$, $\{U\} \rightarrow \{Z\}(-\omega_2)$ if $d_G(u) = 3$ and $d_T(u) = 0$, $\{U\} \rightarrow \{U\}(0)$ if $d_G(u) = 2$ and $d_T(u) = 1$, $\{Z\} \rightarrow \{U\}(\omega_2)$ if $d_G(u) = 2$ and $d_T(u) = 0$. Now in the last case we must have $d_G(v) = 3$ and $d_T(v) = 0$ or $d_G(v) = 3$ and $d_T(v) = 1$. Thus, in the first case the combined status change is $\{Z, U\} \rightarrow \{U, Z\}(0)$. In the second case immediately afterwards **Bridge** will be applied and the status change is $\{Z, U\} \rightarrow \{U, Y\}(\omega_2 - 1)$

Bridge: If **Bridge** is applied, then let $e = \{u, v\}$ with $u \in \partial_V E(T)$. Vertex u is either in U or in X , and $v \in U \cup Z \cup W$. If $v \in U$, then $v \in U$ after the application of **Bridge**, as v is not incident to an edge of T (otherwise reduction rule **Cycle** would have applied). In this case, it is sufficient to check how the status of u can change, which is $\{U\} \rightarrow \{Y\}(-1)$ if u has degree 2, $\{U\} \rightarrow \{X\}(-\omega_1)$ if $d_G(u) = 3$ and $d_T(u) = 1$, and $\{X\} \rightarrow \{Y\}(\omega_1 - 1)$ if $d_G(u) = 3$ and $d_T(u) = 2$. If $v \in Z$, then v moves to U as u necessarily ends up in $X \cup Y$. The possible status changes are $\{U, Z\} \rightarrow \{Y, U\}(\omega_2 - 1)$ if $d_G(u) = 2$, $\{U, Z\} \rightarrow \{X, U\}(\omega_2 - \omega_1)$, if $d_G(u) = 3$ and $d_T(u) = 1$, and $\{X, Z\} \rightarrow \{Y, U\}(\omega_1 + \omega_2 - 1)$ if $d_G(u) = 3$ and $d_T(u) = 2$. If $v \in W$, v ends up in X or Y , depending on whether it is incident to one or two pt-edges. The possible status changes are then $\{U, W\} \rightarrow \{Y, X\}(\omega_3 - 1 - \omega_1)$, $\{U, W\} \rightarrow \{Y, Y\}(\omega_3 - 2)$, $\{U, W\} \rightarrow \{X, X\}(\omega_3 - 2 \cdot \omega_1)$, $\{U, W\} \rightarrow \{X, Y\}(\omega_3 - \omega_1 - 1)$, $\{X, W\} \rightarrow \{Y, X\}(\omega_1 + \omega_3 - 1 - \omega_1)$, and $\{X, W\} \rightarrow \{Y, Y\}(\omega_1 + \omega_3 - 2)$.

Deg1: If **Deg1** applies, the possible status changes are $\{U\} \rightarrow \{W\}(-\omega_3)$ and $\{Z\} \rightarrow \{W\}(\omega_2 - \omega_3)$. Note that **Bridge** is applied before.

Pending: In **Pending**, the only possible status change $\{W\} \rightarrow \{U\}$ has positive value, but the measure κ still decreases as \tilde{k} also increases by 1.

ConsDeg2: Similarly, in **ConsDeg2**, a vertex in $Z \cup U$ disappears, but \tilde{k} increases by 1.

Deg2: In **Deg2**, the possible status changes with respect to u, v are $\{U\} \rightarrow \{Y\}(-\omega_2)$, $\{U, Z\} \rightarrow \{Y, U\}(\omega_2 - 1)$, and $\{U, W\} \rightarrow \{Y, X\}(\omega_3 - 1 - \omega_1)$.

Special: In **Special**, the possible status changes with respect to u, v are $\{U, Z\} \rightarrow \{X, U\}(\omega_2 - \omega_1)$ and $\{X\} \rightarrow \{Y\}(\omega_1 - 1)$.

Attach. In **Attach**, u moves from X to Y . Thus the status change for u is $\{X\} \rightarrow \{Y\}(\omega_1 - 1)$. Taking into account the status of $v \in N_{V \setminus T}(u)$ another status change is $\{X, U\} \rightarrow \{Y, Z\}(\omega_1 - 1 - \omega_2)$ in case $d_G(v) = 3$ and $d_F(v) = 0$. Observe that $v \in Z$ is not possible as $u \in X$.

Attach2 The only status change happens for u : $\{X\} \rightarrow \{Y\}$. □

Proof. (of Theorem 4.3.14) Table 4.2 outlines how vertices a, b , and their neighbors move between U, X, Y, Z , and W in the branches where an edge is added to F or deleted from G in the different cases of the algorithm. For each case, the worst branching vector is given.

	add	delete	branching tuple
Case 4.(a), $d_G(b) = 2$			

4.3. Subcubic Maximum Internal Spanning Tree

	$a : U \rightarrow X$ $b : Z \rightarrow U$ $c : U \rightarrow Y$	$a : U \rightarrow Y$ $b : Z \rightarrow U$ $c : U \rightarrow X$	$(1 + \omega_1 - \omega_2, 1 + \omega_1 - \omega_2)$
Case 4.(a), $d_G(b) = 3$, b is incident to a pt-edge			
	$a : U \rightarrow X$ $b : W \rightarrow Y$ $c : U \rightarrow Y$	$a : U \rightarrow Y$ $b : W \rightarrow Y$ $c : U \rightarrow X$	$(2 + \omega_1 - \omega_3, 2 + \omega_1 - \omega_3)$
Case 4.(a), $d_G(b) = 3$, b is not incident to a pt-edge			
	$a : U \rightarrow X$ $b : U \rightarrow Y$ $c : U \rightarrow Y$	$a : U \rightarrow Y$ $b : U \rightarrow Z$	$(2 + \omega_1, 1 + \omega_2)$
Case 4.(b), $d_T(a) = 1$			
	$a : X \rightarrow Y$ $b : Z \rightarrow Y$	$a : X \rightarrow Y$ $b : Z \rightarrow U$ $c : U \rightarrow W$	$(1 + \omega_1 - \omega_2, 1 + \omega_3 - \omega_2)$
Case 4.(b), $d_T(a) = 2$			
	$a : X \rightarrow Y$ $b : Z \rightarrow Y$	$a : X \rightarrow Y$ $b : Z \rightarrow U$ $c : U \rightarrow W$	$(2 - \omega_1 - \omega_2, 1 - \omega_1 - \omega_2 + \omega_3)$
Case 4.(c)			
	$a : U \rightarrow X$ $b : W \rightarrow X$	$a : U \rightarrow Y$ $b : W \rightarrow Y$ $c : U \rightarrow W$	$(2\omega_1 - \omega_3, 2)$
Case 4.(d)			
	$a : U \rightarrow X$	$a : U \rightarrow Y$ $b : U \rightarrow Z$	$(\omega_1, 1 + \omega_2)$
Case 5, $d_G(x) = d_G(c) = 3$ and there is $q \in (X \cap (N(x) \cup N(c)))$, w.l.o.g. $q \in N(c)$			
	$a : X \rightarrow Y$ $b : U \rightarrow Y$ $q : X \rightarrow Y$	$a : X \rightarrow Y$ $b : U \rightarrow Z$	$(2 - \omega_1, 3 - 2\omega_1, 1 - \omega_1 + \omega_2)$
Case 5, $d_G(x) = d_G(c) = 3$			
	$a : X \rightarrow Y$ $b : U \rightarrow Y$	$a : X \rightarrow Y$ $b : U \rightarrow Z$	$(2 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2, 1 - \omega_1 + \omega_2,)$

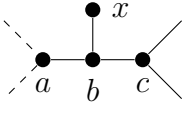
$c/x : U \rightarrow Z$	
There are 3 branches; 2 of them (add) are symmetric.	
Case 5, $d_G(x) = 2$ or $d_G(c) = 2$ and	
	$a : X \rightarrow Y \quad a : X \rightarrow Y$ $b : U \rightarrow Y \quad b : U \rightarrow Z \quad (2 - \omega_1, 2 - \omega_1, 2 - \omega_1)$ When $\{a, b\}$ is deleted, ConsDeg2 additionally decreases k by 1 and removes a vertex of Z .

Table 4.2.: Analysis of the branching for the running time of Theorem 4.3.14

The tight branching numbers are found for cases 4.(b) with $d_T(a) = 2$, 4.(c), 4.(d), and 5. with all of b 's neighbors having degree 3. The respective branching numbers are $(2 - \omega_1 - \omega_2, 1 - \omega_1 - \omega_2 + \omega_3)$, $(2\omega_1 - \omega_3, 2)$, $(\omega_1, 1 + \omega_2)$, and $(1 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2)$. They all equal 2.1364. \square

4.4. Conclusion & Future Research

We have shown that MAX INTERNAL SPANNING TREE can be solved in time $\mathcal{O}^*(3^n)$. This result has been out-dated by Nederlof [123] by providing a $\mathcal{O}^*(2^n)$ polynomial-space algorithm for MIST which is based on the principle of Inclusion-Exclusion and on a new concept called “branching walks”.

We focused on algorithms for MIST that work for the degree-bounded case, in particular, for subcubic graphs. The main novelty is a *Measure&Conquer* approach to analyze our algorithm from a parameterized perspective (parameterizing by the solution size). We are not aware of many examples where this was successfully done without cashing the obtained gain at an early stage, see M. Wahlström [155]. More examples in this direction would be interesting to see. Further improvements on the running times of our algorithms pose another natural challenge.

A related problem worth investigating is the generalization to directed graphs: Find a directed tree, which consist of directed paths form the root to the leaves with as few leaves as possible. Which results can be carried over to the directed case?

Chapter 5.

A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem

5.1. Introduction

We investigate the following problem DIRECTED MAXIMUM LEAF SPANNING TREE (DMLST)

Given: A directed graph $G(V, A)$.

Task: Find a directed spanning tree for G with the maximum number of leaves.

Alternatively, we can find an out-branching with the maximum number of leaves. Here an out-branching in a directed graph is a spanning tree T in the underlying undirected graph, but the arcs are directed from the root to the leaves, which are the vertices of out-degree zero with respect to T . The terms out-branching and directed spanning tree are equivalent.

5.1.1. Known Results.

The undirected version of the problem already has been widely studied with regard to its approximability. There is a 2-approximation running in polynomial time by R. Solis-Oba [145]. In almost linear time H.-I. Lu and R. Ravi [113] provide a 3-approximation. P.S. Bonsma and F. Zickfeld [15] could show that the problem is $\frac{3}{2}$ -approximable when the input is restricted to cubic graphs. J. Daligault and S. Thomassé [33] described a 92-approximation algorithm together with an $\mathcal{O}(k^2)$ -kernel for the DIRECTED MAXIMUM LEAF SPANNING TREE problem.

This problem has also drawn notable attention in the field of parameterized algorithms. Here the problem is known as *directed k -leaf spanning tree* where k is a lower bound on the number of leaves in the directed spanning tree. The algorithm of J. Kneis, A. Langer and P. Rossmanith [98] solves this problem in time $\mathcal{O}^*(4^k)$. Moreover, in J. Daligault *et al.* [32] an upper-bound of $\mathcal{O}^*(3.72^k)$ is achieved. The same authors could also analyze their algorithm with respect to the input size n . This implies a run time upper bound of $\mathcal{O}^*(1.9973^n)$. D. Raible and H. Fernau [135] improved this run time to $\mathcal{O}^*(3.4575^k)$ in the more special case of undirected graphs.

F.V. Fomin, F. Grandoni and D. Kratsch [73] gave an exact, non-parameterized algorithm with run time $\mathcal{O}^*(1.9407^n)$ for the undirected version. H. Fernau *et al.* [51] improved this upper bound to $\mathcal{O}^*(1.8966^n)$. I. Koutis and R. Williams [103] could derive a randomized $\mathcal{O}^*(2^k)$ -algorithm for the undirected version. Using an observation of V. Raman and S. Saurabh [137] this implies a randomized algorithm with run time $\mathcal{O}^*(1.7088^n)$.

5.1.2. Our Achievements.

The main result in this paper improves the current best upper of $\mathcal{O}^*(1.9973^n)$ by [32]. We can achieve a new bound of $\mathcal{O}^*(1.9044^n)$. Our algorithm is inspired by the one of [51]. However, this algorithm cannot be simply transferred to the directed version. Starting from an initial root the algorithm grows a tree T . The branching process takes place by deciding whether the vertices neighbored to the tree will become final leaves or internal vertices. A crucial ingredient of the algorithm was also to create *floating leaves*, i.e., vertices which are final leaves in the future solution but still have to be attached to the T , the tree which is grown. This concept has been already used in [51] and partly by [32]. In the undirected case we guarantee that in the bottleneck case we can generate at least two such leaves. In the directed version there is a situation where only one can be created. Especially for this problem we had to find a workaround.

5.1.3. Preliminaries, Terminology & Notation

A subgraph $H(V_H, A_H)$ of G is called a *directed tree* if there is a unique root $r \in V_H$ such that there is a unique directed path P from r to every $v \in V_H \setminus \{r\}$ under the restriction that its arc set obeys $A(P) \subseteq A_H$. Speaking figuratively, in a directed tree the arcs are directed from the parent to the child. If for a directed tree $H = (V_H, A_H)$ that is a subgraph of $G(V, A)$ we have $V = V_H$ we call it *spanning directed tree* of G . The terms *out-tree* and *out-branching* are sometimes used for directed tree and spanning directed tree, respectively. The *leaves* of a directed tree $H = (V_H, A_H)$ are the vertices u such that $d_{V_H}^-(u) = d_{V_H}(u) = 1$. In $leaves(H)$ all leaves of a tree H are comprised and $internal(H) := V(H) \setminus leaves(H)$. The unique vertex v such that $N_{V_H}^-(u) = \{v\}$ for a tree-vertex will be called *parent* of u . A vertex $v \in V_H$ such that $d_{V_H}(v) \geq 2$ will be called *internal*. Let $T(V_T, A_T)$ and $T'(V_{T'}, A_{T'})$ be two trees. T' *extends* T , written $T' \succeq T$, iff $V_T \subseteq V_{T'}$, $A_T \subseteq A_{T'}$. Simplistically, we will consider a tree T also as a set of arcs $T \subseteq A$ such that $G(T)$ is a directed tree. The notions of \succeq and $leaves(T)$ carry over canonically.

An *arc-cut set* is a set of arcs $B \subset A$ such that $G(A \setminus B)$ is a digraph which is not weak connected. We suppose that $|V| \geq 2$. The function $\chi()$ returns 1 if its argument evaluates to true and 0 otherwise.

5.1.4. Basic Idea of the Algorithm

First we formally re-define our problem:

ROOTED DIRECTED MAXIMUM LEAF SPANNING TREE (RDMLST)

Given: A directed graph $G(V, A)$ and a vertex $r \in V$.

Task: Find a spanning directed tree $T' \subseteq A$ such that $|leaves(T')|$ is maximum and $d_{T'}^-(r) = 0$.

Once we have an algorithm for RDMLST it is easy to see that it can be used to solve DMLST. As a initial step we will consider every vertex as a possible root r of the final solution. This yields a total of n cases.

Then in the course of the algorithm for RDMLST we will gradually extend an out-tree $T \subseteq A$, which is predetermined to be a subgraph in the final out-branching. Let $V_T := V(T)$ and $\bar{V}_T := V \setminus V_T$. We will also maintain a mapping $lab : V \rightarrow \{\text{free}, \text{IN}, \text{LN}, \text{BN}, \text{FL}\} =: D$, which assigns different roles to the vertices. If $lab(v) = \text{IN}$ then v is already fixed to be internal, if $lab(v) = \text{LN}$ then it will be a leaf. If $lab(v) = \text{BN}$ then v already has a parent in T , but can be leaf or internal in the final solution. Such vertices are called *branching nodes*. In general we will decide this by branching on such BN-vertices. If $lab(v) = \text{FL}$ then v is constrained to be a leaf but has not yet been attached to the tree T . Such vertices are called *floating leaves*. If $lab(v) = \text{free}$ then $v \notin V_T$ and nothing has been fixed for v yet. For a *label* $Z \in D$ and $v \in V$ we will often write $v \in Z$ when we mean $lab(v) = Z$. Vertices in IN or LN will also be called *internal nodes* or *leaf nodes*, respectively. A given tree T' defines a labeling $V_{T'} \rightarrow D$ to which we refer by $lab_{T'}$: Let $\text{IN}_{T'} := \{v \in V_{T'} \mid d_{T'}^+(v) \geq 1\}$, $\text{LN}_{T'} := \{v \in V_{T'} \mid d_G^+(v) = 0\}$ and $\text{BN}_{T'} = V_{T'} \setminus (\text{IN}_{T'} \cup \text{LN}_{T'})$. Then for any $ID \in D \setminus \{\text{FL}, \text{free}\}$ we have $ID_{T'} = lab^{-1}(ID)$. We always assure that lab_T and lab are the same on V_T . The subscript might be hence suppressed if $T' = T$. If $T' \succ T$, then we assume that $\text{IN}_T \subseteq \text{IN}_{T'}$ and $\text{LN}_T \subseteq \text{LN}_{T'}$. So, the labels IN and LN remain once they are fixed. For the remaining labels we have the following possible transitions: $\text{FL} \rightarrow \text{LN}$, $\text{BN} \rightarrow \{\text{LN}, \text{IN}\}$ and $\text{free} \rightarrow D \setminus \{\text{free}\}$. Let $\text{BN}_i = \{v \in \text{BN} \mid d^+(v) = i\}$, $\text{free}_i = \{v \in \text{free} \mid d^-(v) = i\}$ for $i \geq 1$, $\text{BN}_{\geq \ell} := \cup_{j=\ell}^n \text{BN}_j$ and $\text{free}_{\geq \ell} := \cup_{j=\ell}^n \text{free}_j$.

5.2. The Polynomial Part

5.2.1. Halting Rules

First we specify halting rules. If one of these rules applies the algorithm halts. Then it either returns a solution or answers that none can be built in the according branch of the search tree.

(H1) If there exists a $v \in \text{free} \cup \text{FL}$ with $d^-(v) = 0$. Halt and answer NO.

(H2) If $\text{BN} = \emptyset$. Halt. A spanning tree has been constructed if $\text{free} \cup \text{FL} = \emptyset$. If so return $|\text{LN}|$.

(H3) If there is a bridge $e := (u, v) \in A \setminus T$ which splits the graph in at least two connected components of size at least two and $v \in \text{FL}$. Halt and answer **NO**.

5.2.2. Reduction rules

We state a set of six reduction rules in the following. Similar reduction rules for the undirected version have already appeared in [51, 135]. We assume that the halting rules are already checked exhaustively

- (R1)** Let $v \in V$. If $\text{lab}(v) = \text{FL}$ then remove $N_A^+(v)$. If $\text{lab}(v) = \text{BN}$ then remove $N_A^-(v) \setminus T$.
- (R2)** If there exists a vertex $v \in \text{BN}$ with $d^+(v) = 0$ then set $\text{lab}(v) := \text{LN}$.
- (R3)** If there exists a vertex $v \in \text{free}$ with $d(v) = 1$ then set $\text{lab}(v) := \text{FL}$.
- (R4)** If $v \in \text{LN}$ then remove $N_A(v) \setminus T$.
- (R5)** Let $u \in \text{BN}$ such that $N_A^+(u)$ is a an arc-cut set. Then $\text{lab}(u) := \text{IN}$ and for all $x \in N^+(u) \cap \text{FL}$ set $\text{lab}(x) := \text{LN}$, and for all $x \in N^+(u) \cap \text{free}$ set $\text{lab}(x) := \text{BN}$.
- (R6)** If there is an arc $(a, b) \in A$ with $a, b \in \text{free}$ and $G(A \setminus \{a, b\})$ consist of two strongly connected components of vertex-size at least two. Then contract (a, b) such that the new vertex is free. both vertices.

Proposition 5.2.1: The reduction rules are sound.

Proof. **(R1)** A floating leaf v cannot be a parent anymore. Thus, it is valid to remove $N_A^+(v)$. If $v \in \text{BN}$ then v already has a parent in T . Thus, no arc in $N^-(v) \setminus T$ will ever be part of a tree $T' \succeq T$.

- (R2)** The vertex v cannot be a parent anymore. Thus, setting $\text{lab}(v) := \text{LN}$ is sound.
- (R3)** The vertex v must be a leaf in any tree $T' \succeq T$.
- (R4)** The only arcs present in any tree $T' \succeq T$ will be $N_A(v) \cap T$. Thus, $N_A(v) \setminus T$ can be removed.
- (R5)** As $N_A^+(v)$ is an arc-cut set, setting $v \in \text{LN}$ would cut off a component which cannot be reached from the root r . Thus, $v \in \text{IN}$ is constrained.
- (R6)** Let G^* be the graph after contracting (a, b) . If G^* has a spanning tree with k leaves, then also G . On the other hand note that in every spanning tree $T' \succeq T$ for G we have that $a, b \in \text{IN}$ and $(a, b) \in T'$. Hence, the tree $T^\#$ evolved by contracting (a, b) in T' is a spanning tree with k leaves in G^* .

□

5.3. The Exponential Part

5.3.1. Branching rules

If $N^+(\text{internal}(T)) \subseteq \text{internal}(T) \cup \text{leaves}(T)$, we call T an *inner-maximal* directed tree. We make use of the following fact:

Lemma 5.3.1 ([98] Lemma 4.2): If there is a tree T' with $\text{leaves}(T') \geq k$ such that $T' \succeq T$ and $x \in \text{internal}(T')$ then there is a tree T'' with $\text{leaves}(T'') \geq k$ such that $T'' \succeq T$, $x \in \text{internal}(T'')$ and $\{(x, u) \in A\} \subseteq T''$

See the Algorithm 2 which describes the branching rules. As mentioned before, the search tree evolves by branching on BN-vertices. For some $v \in \text{BN}$ we will set either $\text{lab}(v) = \text{LN}$ or $\text{lab}(v) = \text{IN}$. In the second case we adjoin the vertices $N_A^+(v) \setminus T$ as BN-nodes to the partial spanning tree T . This is justified by Lemma 5.3.1. Thus, during the whole algorithm we only consider inner-maximal trees. Right in the beginning we therefore have $A(\{r\} \cup N^+(r))$ as a initial tree where r is the vertex chosen as the root.

We also introduce an abbreviating notation for the different cases generated by branching: $\langle v \in \text{LN}; v \in \text{IN} \rangle$ means that we recursively consider the two cases were v becomes a leaf node and an internal node. The semicolon works as a delimiter between the different cases. Of course, more complicated expression like $\langle v \in \text{BN}, x \in \text{BN}; v \in \text{IN}, x \in \text{LN}; v \in \text{LN} \rangle$ are possible, which generalize straight-forward.

5.3.2. Correctness of the algorithm

In the following we are going to prove a lemma which is crucial for the correctness and the run time.

Lemma 5.3.2: Let $T \subseteq A$ be a given tree such that $v \in \text{BN}_T$ and $N^+(v) = \{x_1, x_2\}$. Let $T', T^* \subseteq A$ be optimal solutions with $T', T^* \succeq T$ under the restriction that $\text{lab}_{T'}(v) = \text{LN}$, and $\text{lab}_{T^*}(v) = \text{IN}$ and $\text{lab}_{T^*}(x_1) = \text{lab}_{T^*}(x_2) = \text{LN}$.

1. If there is a vertex $u \neq v$ with $N^+(u) = \{x_1, x_2\}$. Then $|\text{leaves}(T')| \geq |\text{leaves}(T^*)|$.
2. Assume that $d^-(x_i) \geq 2$ ($i = 1, 2$). Assume that there exists some $u \in (N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$ such that $\text{lab}_{T^*}(u) = \text{IN}$. Then $|\text{leaves}(T')| \geq |\text{leaves}(T^*)|$.

Proof. 1. Let $T^+ := (T^* \setminus \{(v, x_1), (v, x_2)\}) \cup \{(u, x_1), (u, x_2)\}$. We have $\text{lab}_{T^+}(v) = \text{LN}$ and u is the only vertex besides v where $\text{lab}_{T^*}(u) \neq \text{lab}_{T^+}(u)$ is possible. Hence, u is the only vertex where we could have $\text{lab}_{T^*}(u) = \text{LN}$ such that $\text{lab}_{T^+}(u) = \text{IN}$. Thus, we can conclude $|\text{leaves}(T^+)| \geq |\text{leaves}(T^*)|$. As T' is optimal under the restriction that $v \in \text{LN}$ it follows $|\text{leaves}(T')| \geq |\text{leaves}(T^+)| \geq |\text{leaves}(T^*)|$.

2. W.l.o.g. we have $u \in N^-(x_1) \setminus \{v, x_2\}$. Let $q \in N^-(x_2) \setminus \{v\}$ and $T^+ := (T^* \setminus \{(v, x_1), (v, x_2)\}) \cup \{(u, x_1), (q, x_2)\}$. We have $\text{lab}_{T^+}(v) = \text{LN}$, $\text{lab}_{T^+}(u) = \text{lab}_{T^*}(u) = \text{IN}$ and q is the only vertex besides v where we could have $\text{lab}_{T^*}(q) \neq$

Algorithm 2: An Algorithm for solving RDMLST

Data: A directed graph $G = (V, A)$ and a directed tree $T \subseteq A$.

Result: A spanning directed tree T' with the maximum number of leaves such
 $T' \succeq T$

1 Check if a halting rule applies.

2 Apply the reduction rules exhaustively.

3 **if** $BN_1 \neq \emptyset$ **then**

4 Choose some $v \in BN_1$.

5 Let $P = \{v_0, v_1, \dots, v_k\}$ be a path of maximum length s.t. (1) $v_0 = v$, (2) for all $1 \leq i \leq k - 1$ $d_{P_{i-1}}^+(v_i) = 1$ (where $P_{i-1} = \{v_0, \dots, v_{i-1}\}$) and (3)

$P \setminus \text{free} \subseteq \{v_0, v_k\}$

6 **if** $d_{P_{i-1}}^+(v_k) = 0$ **then**

7 └ Put $v \in \text{LN}$ (B1)

8 **else**

9 └ $\langle v \in \text{IN}, v_1, \dots, v_k \in \text{IN}; v \in \text{LN} \rangle$ (B2)

10 **else**

11 Choose a vertex $v \in \text{BN}$ with maximum out-degree.

12 **if** $a) d^+(v) \geq 3$ **or** $b) (N^+(v) = \{x_1, x_2\} \text{ and } N^+(v) \subseteq \text{FL})$ **then**

13 └ $\langle v \in \text{IN}; v \in \text{LN} \rangle$

14 └ & in case $b)$ apply `makeleaves`(x_1, x_2) in the 1st branch. (B3)

15 **else if** $N^+(v) = \{x_1, x_2\}$ **then**

16 └ **if** for $z \in (\{x_1, x_2\} \cap \text{free})$ we have $|N^+(z) \setminus N^+(v)| = 0$ ($B4.1$) **or**

17 └ $N_A^+(z)$ is an arc-cut set ($B4.2$) **or**

18 └ $N^+(z) \setminus N^+(v) = \{v_1\}$. ($B4.3$) **then**

19 └ $\langle v \in \text{IN}; v \in \text{LN} \rangle$ (B4)

20 **else if** $N^+(v) = \{x_1, x_2\}$, $x_1 \in \text{free}$, $x_2 \in \text{FL}$ **then**

21 └ $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}; v \in \text{LN} \rangle$

22 └ & apply `makeleaves`(x_1, x_2) in the 2nd branch. (B5)

23 **else if** $N^+(v) = \{x_1, x_2\}$, $x_1, x_2 \in \text{free}$, $\exists z \in (N^-(x_1) \cap N^-(x_2)) \setminus \{v\}$ **then**

24 └ $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{LN} \rangle$ (B6)

25 **else if** $N^+(v) = \{x_1, x_2\}$, $x_1, x_2 \in \text{free}$, $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| \geq 2$ **then**

26 └ $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{LN}; v \in \text{LN} \rangle$

27 └ & apply `makeleaves`(x_1, x_2) in the 3rd branch. (B7)

28 **else**

29 └ $\langle v \in \text{IN}; v \in \text{LN} \rangle$ (B8)

Procedure `makeleaves`(x_1, x_2)

1 **begin**2 $\forall u \in [(N^-(x_1) \cup N^-(x_2)) \setminus \{x_1, x_2, v\}] \cap \text{free set } u \in \text{FL};$ 3 $\forall u \in [(N^-(x_1) \cup N^-(x_2)) \setminus \{x_1, x_2, v\}] \cap \text{BN set } u \in \text{LN};$ 4 **end**

$lab_{T^+}(q)$ (i.e., it is possible that $lab_{T^*}(q) = \text{LN}$ and $lab_{T^+}(q) = \text{IN}$). Therefore $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$. \square

5.3.2.1. Correctness of the Different Branching Cases

First note that **(H2)** takes care of the case that indeed an out-branching has been built. If so the number of its leaves is returned.

Below we will argue that each branching case in Algorithm 2 is correct in a way that it preserves at least one optimal solution. Cases **(B4)** and **(B8)** do not have to be considered in detail as these are simple binary and exhaustive branchings.

(B1) Suppose there is an optimal extension $T' \succeq T$ such that $lab_{T'}(v) = lab_{T'}(v_0) = \text{IN}$. Due to the structure of P there must be an i , $0 < i \leq k$ such that $(v_j, v_{j-1}) \in T'$ for $0 < j \leq i$, i.e., $v, v_1, \dots, v_{i-1} \in \text{IN}$ and $v_i \in \text{LN}$. W.l.o.g., we choose T' in a way that i is minimum but T' is still optimal (\clubsuit). By **(R5)** there must be a vertex v_z , $0 < z \leq i$, such that there is an arc (q, v_z) with $q \notin P$ and $q \in V(T')$. Now consider $T'' = (T' \setminus \{(v_{z-1}, v_z)\}) \cup \{q, v_z\}$. In T'' the vertex v_{z-1} is a leaf and therefore $|leaves(T'')| \geq |leaves(T')|$. Additionally, we have that $z - 1 < i$ which is a contradiction to the choice of T' (\clubsuit).

(B2) Note that $lab(v_k) \in \{\text{BN}, \text{FL}\}$ is not possible due to **(R1)** and, thus, $lab(v_k) = \text{free}$. By the above arguments from **(B1)** we can exclude the case that $v, v_1, \dots, v_{i-1} \in \text{IN}$ and $v_i \in \text{LN}$ ($i \leq k$). Thus, under the restriction that we set $v \in \text{IN}$, the only remaining possibility is also to set $v_1, \dots, v_k \in \text{IN}$.

(B3) *b)* When we set $v \in \text{IN}$ then the two vertices in $N^+(v)$ will become leaf nodes (i.e., become part of LN). Thus, Lemma 5.3.2.2 applies (Note that **(R5)** does not apply and therefore $(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\} \neq \emptyset$ as well as $d(x_i) \geq 2$ ($i = 1, 2$)). This means that every vertex in $(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$ can be assumed to be leaf node in the final solution. This justifies to apply `makeleaves`(x_1, x_2).

(B5) The branching is exhaustively with respect to v and x_1 . Nevertheless, in the second branch `makeleaves`(x_1, x_2) is carried out. This is justified by Lemma 5.3.2.2 similarly as in **(B3)***b)*. By setting $v \in \text{IN}$ and $x_1 \in \text{LN}$, x_2 will be attached to v as a LN-node and **(R5)** does not apply.

(B6) In this case we neglect the possibility that $v \in \text{IN}, x_1, x_2 \in \text{LN}$. But due to Lemma 5.3.2.1 a no worse solution can be found in the recursively considered case where we set $v \in \text{LN}$. This shows that the considered cases are sufficient.

(B7) Similarly, as in case (B3) we can justify by Lemma 5.3.2.2 the application of $\text{makeleaves}(x_1, x_2)$ in the third branch.

Further branching cases will not be considered as their correctness is clear due to exhaustive branching.

5.3.3. Analysis of the Run Time

5.3.3.1. The Measure

To analyze the run time we follow the *Measure&Conquer*-approach (see [74]) and use the following measure:

$$\mu(G) = \sum_{i=1}^n \epsilon_i^{\text{BN}} |\text{BN}_i| + \sum_{i=1}^n \epsilon_i^{\text{free}} |\text{free}_i| + \epsilon^{\text{FL}} |\text{FL}|$$

The concrete values are $\epsilon^{\text{FL}} = 0.2251$, $\epsilon_1^{\text{BN}} = 0.6668$, $\epsilon_i^{\text{BN}} = 0.7749$ for $i \geq 2$, $\epsilon_1^{\text{free}} = 0.9762$ and $\epsilon_2^{\text{free}} = 0.9935$. Also let $\epsilon_j^{\text{free}} = 1$ for $j \geq 3$ and $\eta = \min\{\epsilon^{\text{FL}}, (1 - \epsilon_1^{\text{BN}}), (1 - \epsilon_2^{\text{BN}}), (\epsilon_2^{\text{free}} - \epsilon_1^{\text{BN}}), (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}), (\epsilon_1^{\text{free}} - \epsilon_1^{\text{BN}}), (\epsilon_1^{\text{free}} - \epsilon_2^{\text{BN}})\} = \epsilon_1^{\text{free}} - \epsilon_2^{\text{BN}} = 0.2013$.

For $i \geq 2$ let $\Delta_i^{\text{free}} = \epsilon_i^{\text{free}} - \epsilon_{i-1}^{\text{free}}$ and $\Delta_1^{\text{free}} = \epsilon_1^{\text{free}}$. Thus, $\Delta_{i+1}^{\text{free}} \leq \Delta_i^{\text{free}}$ with $\Delta_s^{\text{free}} = 0$ for $s \geq 4$.

5.3.3.2. Run Time Analysis of the Different Branching Cases

In the following we state for every branching case by how much μ will be reduced. Especially, Δ_i states the amount by which the i -th branch decreases μ . If v is the vertex chosen by Algorithm 2 then it is true that for all $x \in N^+(v)$ we have $d^-(x) \geq 2$ by **(R5)** (\clubsuit).

(B2) $\langle v \in \text{IN}, v_1, \dots, v_k \in \text{IN}, v \in \text{LN} \rangle$

Recall that $d_{P_{k-1}}^+(v_k) \geq 2$ and $v_k \in \text{free}$ by **(R1)**. Then we must have that $v_1 \in \text{free}_{\geq 2}$ by **(R5)**.

1. v becomes IN-node; v_1, \dots, v_k become IN-nodes; the free vertices in $N^+(v_k)$ become BN-nodes, the floating leaves in $N^+(v_k)$ become LN-nodes:
 $\Delta_1 \geq \epsilon_1^{\text{BN}} + \sum_{i=2}^k \epsilon_1^{\text{free}} + \chi(v_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \chi(v_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + 2 \cdot \eta$
2. v becomes LN-node; the degree of v_1 is reduced:
 $\Delta_2 \geq \epsilon_1^{\text{BN}} + \sum_{i=2}^3 \chi(v_1 \in \text{free}_i) \cdot \Delta_i^{\text{free}}$

The greatest branching number 1.7542 evolves from the case where $k = 1$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$ and for all $u \in N^+(v_1)$ we have $u \in \text{free}_1$.

(B3) $\langle v \in \text{IN}; v \in \text{LN} \rangle$.

Case a)

1. v becomes IN-node; the free out-neighbors of v become BN-nodes; the FL out-neighbors of v becomes LN-nodes:

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + \sum_{x \in N^+(v) \cap \text{free}_{\geq 3}} (1 - \epsilon_2^{\text{BN}}) + \sum_{x \in N^+(v) \cap \text{free}_2} (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \sum_{y \in N^+(v) \cap \text{FL}} \epsilon^{\text{FL}}$$

2. v becomes LN-node; the in-degree of the free out-neighbors of v is decreased;

$$\Delta_2 \geq \epsilon_2^{\text{BN}} + \sum_{i=2}^3 |N^+(v) \cap \text{free}_i| \cdot \Delta_i^{\text{free}}$$

The greatest branching number 1.9044 evolves from the cases where $d^+(v) = 3$ and $|N^+(v) \cap \text{FL}| + |N^+(v) \cap \text{free}_{\geq 4}| = 3$.

Case b)

Recall that v is a BN of maximum out-degree, thus $d^+(z) \leq d^+(v) = 2$ for all $z \in \text{BN}$. On the other hand $\text{BN}_1 = \emptyset$ which implies $\text{BN} = \text{BN}_2$ from this point on. Hence, we have $N^+(v) = \{x_1, x_2\}$, $d^-(x_i) \geq 2$, ($i = 1, 2$) and $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| \geq 1$ by \clubsuit , in the following branching cases. Therefore the additional amount of $\min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$ in the first branch is justified by the application of `makeleaves`(x_1, x_2). Note that by \clubsuit at least one free-node becomes a FL-node, or one BN-node becomes a LN-node. Also due to **(R1)** we have that $N^+(x_i) \cap \text{BN} = \emptyset$. From this case the branching number 1.719 is derived.

1. v becomes IN-node; the FL out-neighbors of v become LN-nodes; the vertices in $[N^-(x_1) \cup N^-(x_2) \setminus \{v, x_1, x_2\}] \cap \text{BN}$ become LN-nodes; the vertices in $[N^-(x_1) \cup N^-(x_2) \setminus \{v, x_1, x_2\}] \cap \text{free}$ become FL-nodes.

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + 2 \cdot \epsilon^{\text{FL}} + \min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$$

2. v becomes LN; $\Delta_2 \geq \epsilon_2^{\text{BN}}$.

(B4) $\langle v \in \text{IN}; v \in \text{LN} \rangle$.

- (B4.1):
1. v becomes IN-node; z becomes LN-node by **(R1)**, **(R2)** or both; The vertex $q \in \{x_1, x_2\} \setminus \{z\}$ becomes LN-node or BN-node (depending on $q \in \text{FL}$ or $q \in \text{free}$)

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \min\{\epsilon^{\text{FL}}, (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})\}$$

2. v becomes LN-node;

$$\Delta_2 \geq \epsilon_2^{\text{BN}}$$

- (B4.2):
1. v becomes IN-node; $N_A^+(z)$ is an arc-cut. Thus, z becomes IN-node as **(R5)** applies; The vertex $q \in \{x_1, x_2\} \setminus \{z\}$ becomes LN-node or BN-node (depending on $q \in \text{FL}$ or $q \in \text{free}$)

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \min\{\epsilon^{\text{FL}}, (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})\}$$

2. v becomes LN-node;

$$\Delta_2 \geq \epsilon_2^{\text{BN}}$$

Cases (B4.1)/(B4.2) provide a branching number of 1.717. Note that in all following branching cases we have $N^+(x_i) \cap \text{free}_1 = \emptyset$ ($i = 1, 2$) by this case.

(B4.3): We have $|N^+(z) \setminus N^+(v)| = 1$. Thus, in the next recursive call after the first branch and the exhaustive application of **(R1)**, either **(R6)**, case (B2) or (B1) applies. **(R5)** does not apply due to (B4.2) being ranked higher. Note that the application of any other reduction rule does not change the situation. If (B2) applies we can analyze the current case together with its succeeding one. If (B2) applies in the case we set $v \in \text{IN}$ we deduce that $v_0, v_1, \dots, v_k \in \text{free}$ where $z = v_0 = x_1$ (w.l.o.g., we assumed $z = x_1$). Observe that $v_1 \in \text{free}_{\geq 2}$ as (B4.2) does not apply.

1. v becomes IN-node; x_1 becomes LN-node; x_2 becomes LN- or BN-node (depending on whether $x_2 \in \text{free}$ or $x_2 \in \text{FL}$); the degree of v_1 drops:

$$\begin{aligned} \Delta_{11} &\geq \epsilon_2^{\text{BN}} + \chi(x_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \chi(x_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \\ &\chi(x_2 \in \text{free}_{\geq 3}) \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{free}_2) \cdot \\ &(\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{FL}) \cdot \epsilon^{\text{FL}} + \sum_{i=2}^3 \chi(v_1 \in \text{free}_i) \cdot \Delta_i^{\text{free}} \end{aligned}$$

2. v becomes IN-node, $x_1, v_1 \in \text{IN}, \dots, v_k$ become IN-nodes; the free vertices in $N^+(v_k)$ become BN-nodes, the floating leaves in $N^+(v_k)$ become LN-nodes:

$$\begin{aligned} \Delta_{12} &\geq \epsilon_2^{\text{BN}} + \chi(x_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \chi(x_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \\ &\chi(x_2 \in \text{free}_{\geq 3}) \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{free}_2) \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \\ &\chi(x_2 \in \text{FL}) \cdot \epsilon^{\text{FL}} + \chi(v_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \chi(v_1 \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \sum_{i=2}^k \epsilon_1^{\text{free}} \\ &+ 2\eta \end{aligned}$$

3. v becomes LN-node: the degrees of x_1 and x_2 drop:

$$\Delta_2 \geq \epsilon_2^{\text{BN}} + \sum_{\ell=2}^{\max_{h \in \{1,2\}} d^-(x_h)} \sum_{j=1}^2 \chi(x_j \in \text{free}_\ell) \cdot \Delta_\ell^{\text{free}}$$

The worst case branching number from above is 1.897. It is created by two cases with $k = 1$: 1. $x_2 \in \text{free}_{\geq 4}$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$, $d^-(x_1) \geq 4$ and 2. $x_2 \in \text{FL}$, $d^-(v_1) \geq 4$, $d^+(v_1) = 2$, $d^-(x_1) \geq 4$

If case (B1) applies to v_1 the reduction in both branches is as least as great as in (B4.1)/(B4.2).

If **(R6)** applies after the first branch (somewhere in the graph) we get $\Delta_1 \geq \epsilon_2^{\text{BN}} + (\epsilon_2^{\text{free}} - \epsilon_1^{\text{BN}}) + \epsilon_1^{\text{free}} + \min\{\epsilon^{\text{FL}}, (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})\}$ and $\Delta_2 \geq \epsilon_2^{\text{BN}}$. Here the amount of ϵ_1^{free} in Δ_1 originates from an **(R6)** application. The corresponding branching number is 1.644.

(B5) $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}; v \in \text{LN} \rangle$

1. v and x_1 become IN-nodes; x_2 becomes a LN-node; the vertices in $N^+(x_1) \cap \text{free}$ become BN-nodes; the vertices in $N^+(x_1) \cap \text{FL}$ become LN-nodes;

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \epsilon^{\text{FL}} + \sum_{x \in N^+(x_1) \cap \text{free}} (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \sum_{x \in N^+(x_1) \cap \text{FL}} \epsilon^{\text{FL}}$$

2. v becomes IN-node; x_1 becomes LN-node; x_2 becomes LN-node; after applying **makeleaves**(x_1, x_2) the vertices in $[N^-(x_1) \cup N^-(x_2) \setminus \{v, x_1, x_2\}] \cap \text{BN}$ become LN-nodes and the vertices in $[N^-(x_1) \cup N^-(x_2) \setminus \{v, x_1, x_2\}] \cap \text{free}$ become FL-nodes:

$$\Delta_2 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \epsilon^{\text{FL}} + \min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$$

3. v becomes LN: $\Delta_3 \geq \epsilon_2^{\text{BN}}$

The amount of $\min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$ in the second branch is due to \clubsuit and the application of **makeleaves**(x_1, x_1). The greatest branching number 1.8871 evolves from the case where $N^+(x_1) \subseteq \text{free}$, $d^+(x_1) = 2$, $d^-(x_1) = 2$ and $N^+(x_1) \setminus \{v\} \subseteq \text{free}_1$.

(B6) $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{LN} \rangle$ The branching vector can be derived by considering items 1,2 and 4 of (B7) and the reductions Δ_1, Δ_2 and Δ_4 in μ obtained in each item.

(B7) $\langle v \in \text{IN}, x_1 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{IN}; v \in \text{IN}, x_1 \in \text{LN}, x_2 \in \text{LN}; v \in \text{LN} \rangle$

Note that if $N_A^+(x_1)$ or $N_A^+(x_2)$ is an arc-cut set then (B4.2) applies. Thus, all the branching cases must be applicable.

Moreover due to the previous branching case (B4.3) we have $|N^+(x_1) \setminus N^+(v)| = |N^+(x_1) \setminus \{x_2\}| \geq 2$ and $|N^+(x_2) \setminus N^+(v)| = |N^+(x_2) \setminus \{x_1\}| \geq 2$ (\star).

Note that $N^-(x_1) \cap N^-(x_2) = \{v\}$ due to (B6).

For $i \in \{1, 2\}$ let $fl_i = |\{x \in N^+(x_i) \setminus N^+(v) \mid x \in \text{FL}\}|$, $fr_i^{\geq 3} = |\{u \in N^+(x_i) \setminus N^+(v) \mid u \in \text{free}_{\geq 3}\}|$ and $fr_i^2 = |\{u \in N^+(x_i) \setminus N^+(v) \mid u \in \text{free}_2\}|$.

Observe that for $i \in \{1, 2\}$ we have $(fl_i + fr_i^{\geq 3} + fr_i^2) \geq 2$ due to (\star).

1. v becomes IN; x_1 becomes IN; x_2 becomes BN; the free out-neighbors of x_1 become BN; the FL out-neighbors of x_1 become LN;

$$\begin{aligned} \Delta_1 \geq & \epsilon_2^{\text{BN}} + \chi(x_1 \in \text{free}_{\geq 3}) + \chi(x_1 \in \text{free}_2) \cdot \epsilon_2^{\text{free}} + \\ & \chi(x_2 \in \text{free}_{\geq 3}) \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + \chi(x_2 \in \text{free}_2) \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}}) + \\ & (fl_1 \cdot \epsilon^{\text{FL}} + fr_1^{\geq 3} \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + fr_1^2 \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})) \end{aligned}$$

2. v becomes IN; x_1 becomes LN; x_2 becomes IN; the free out-neighbors of x_2 becomes BN; the FL out-neighbors of x_2 become LN;

$$\begin{aligned} \Delta_2 \geq & \epsilon_2^{\text{BN}} + \left(\sum_{i=1}^2 [\chi(x_i \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \chi(x_i \in \text{free}_2) \cdot \epsilon_2^{\text{free}}] \right) + \\ & (fl_2 \cdot \epsilon^{\text{FL}} + fr_2^{\geq 3} \cdot (\epsilon_3^{\text{free}} - \epsilon_2^{\text{BN}}) + fr_2^2 \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})) \end{aligned}$$

3. v becomes IN; x_1 becomes LN; x_2 becomes LN; the free in-neighbors of x_1 become FL; the BN in-neighbors of x_1 become LN; the free in-neighbors of x_2 become FL; the BN in-neighbors of x_2 become LN:

$$\begin{aligned} \Delta_3 \geq & \epsilon_2^{\text{BN}} + \left[\sum_{i=1}^2 (\chi(x_i \in \text{free}_{\geq 3}) \cdot \epsilon_3^{\text{free}} + \chi(x_i \in \text{free}_2) \cdot \epsilon_2^{\text{free}}) \right] + \\ & \max\{2, (d^-(x_1) + d^-(x_2) - 4)\} \cdot \min\{\epsilon_1^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\} \end{aligned}$$

Note that the additional amount of $\max\{2, (d^-(x_1) + d^-(x_2) - 4)\} \cdot \{\epsilon_2^{\text{free}} - \epsilon^{\text{FL}}, \epsilon_2^{\text{BN}}\}$ is justified by Lemma 5.3.2.2 and by the fact that $d^-(x_i) \geq 2$ and $N^-(x_1) \cap N^-(x_2) = \{v\}$ due to (B6). Thus, we have $|N^-(x_1) \cup N^-(x_2) \setminus \{x_1, x_2, v\}| \geq \max\{2, (d^-(x_1) + d^-(x_2) - 4)\}$.

4. v becomes LN; the degrees of x_1 and x_2 drop:

$$\Delta_4 \geq \epsilon_2^{\text{BN}} + \sum_{j=2}^{\max_{\ell \in \{1,2\}} \{d^-(x_\ell)\}} \sum_{i=1}^2 (\chi(d^-(x_i) = j) \cdot \Delta_j^{\text{free}})$$

The following cases determine branching numbers between 1.9043 and 1.9044:

1. $d^-(x_1) = d^+(x_1) = 2$, $d^-(x_2) = 4$, $d^+(x_2) = 2$, $fr_1^2 = fr_2^2 = 2$
2. $d^-(x_1) = d^-(x_2) = 3$, $d^+(x_1) = d^+(x_2) = 2$, $fr_1^2 = fr_2^2 = 2$
3. $d^-(x_2) = d^+(x_2) = 2$, $d^-(x_1) = 4$, $d^+(x_1) = 2$, $fr_1^2 = fr_2^2 = 2$

(B8) Observe that in the second branch we can apply **(R6)**. Due to the non-applicability of **(R5)** and the fact that (B7) is ranked higher in priority we have $|(N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}| = 1$. Especially, (B6) cannot be applied by which we derive that $N^-(x_1) \cap N^-(x_2) = \{v\}$. Thus, due to this we have the situation in Figure 5.1.

So, w.l.o.g, there are arcs $(q, x_1), (x_1, x_2) \in A$ (and possibly also $(x_2, x_1) \in A$), where $\{q\} = (N^-(x_1) \cup N^-(x_2)) \setminus \{v, x_1, x_2\}$, because we can rely on $d^-(x_i) \geq 2$ ($i = 1, 2$) by **(♣)**.

1. Firstly, assume that $q \in \text{free}$.

a) v becomes IN; x_1 and x_2 becomes BN:

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + 2 \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})$$

b) The arc (q, x_1) will be contracted by **(R6)** when we v becomes LN, as x_1 and x_2 only can be reached by using (q, x_1) :

$$\Delta_2 \geq \epsilon_2^{\text{BN}} + \epsilon_1^{\text{free}}.$$

The branching number here is 1.606.

2. Secondly, assume $q \in \text{BN}$. Then $q \in \text{BN}_2$ due to the branching priorities.

a) v becomes IN; x_1 and x_2 become BN:

$$\Delta_1 \geq \epsilon_2^{\text{BN}} + 2 \cdot (\epsilon_2^{\text{free}} - \epsilon_2^{\text{BN}})$$

b) Then after setting $v \in \text{LN}$, rule **(R5)** will make q internal and subsequently also x_1 :

$$\Delta_2 \geq \epsilon_2^{\text{BN}} + \epsilon_2^{\text{free}} + \epsilon_2^{\text{BN}}.$$

This amount is justified by the changing roles of the vertices in $N^+(q) \cup \{q\}$.

1.499 is the right corresponding branching number.

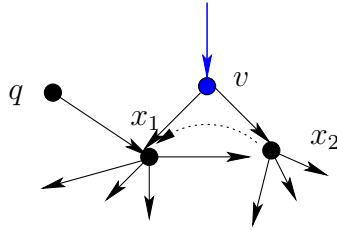


Figure 5.1.: The only situation which can occur in branching case (B8). The blue arc is contained in T , the dotted one maybe present or not.

By the above case analysis we are able to conclude:

Theorem 5.3.3: DIRECTED MAXIMUM LEAF SPANNING TREE can be solved in $\mathcal{O}^*(1.9044^n)$ steps.

The proven run time bound admits only a small gap to the bound of $\mathcal{O}^*(1.8966^n)$ for the undirected version. It seems that we can benefit from degree two vertices only on a small scale in contrast to the undirected problem version. Speaking loosely if $v \in \text{BN}_2$ and $x \in N(v) \setminus T$ we can follow a WIN/WIN approach in the undirected version. Either $d(x)$ is quite big then we will add many vertices to BN or FL when v and subsequently x become internal. If $d(x)$ is small, say two, then by setting $v \in \text{LN}$ the vertex x becomes a FL-node. This implies also an extra reduction of the measure. We point out that in the directed case the in- and out-degree of a vertex generally is not related. Thus, the approach described for the undirected problem remains barred for the directed version.

5.4. Conclusions

5.4.1. An Approach Using Exponential Space

The algorithm of J. Kneis *et al.* [98] can also be read in an exact non-parameterized way. It is not hard to see that it yields a run time of $\mathcal{O}^*(2^n)$. Alternatively, keep the cases (B1) and (B2) of Algorithm 2 and substitute all following cases by a simple branch on some BN-node. Using n as a measure we see that $\mathcal{O}^*(2^n)$ is an upper bound.

We are going to use the technique of memoization to obtain an improved run time. Let $SG^\alpha := \{G(V') \mid V' \subseteq V, |V'| \leq \alpha \cdot n\}$ where $\alpha = 0.141$. Then we aim to create the following table L indexed by some $G' \in SG^\alpha$ and some $V_{\text{BN}} \subseteq V(G')$ where the entries are from

$$\mathcal{L} = \{\tilde{T} \mid \tilde{T} \text{ is directed spanning tree for } G'_{\text{BN}} \text{ with root } r'\}$$

(where $G'_{\text{BN}} = (V(G') \cup \{r', y\}, A(G') \cup (\{(r', y)\} \cup_{u \in V_{\text{BN}}} (r', u)))$ and r', y are new vertices):

$$L[G', V_{\text{BN}}] = T' \text{ such that } |\text{leaves}(T')| = \min_{\tilde{T} \in \mathcal{L}} |\text{leaves}(\tilde{T})| .$$

Entries where such a directed spanning tree \tilde{T} does not exist (e.g. if $V_{\text{BN}} = \emptyset$) get the value \emptyset . This table can be filled up in time $\mathcal{O}^*\left(\binom{n}{\alpha \cdot n} \cdot 2^{\alpha n} \cdot 1.9044^{\alpha n}\right) \subseteq \mathcal{O}^*(1.8139^n)$. This run time is composed of enumerating SG^α , then by cycling through all possibilities for V_{BN} and finally solving the problem on instance G'_{BN} with Algorithm 2.

Theorem 5.4.1: DIRECTED MAXIMUM LEAF SPANNING TREE can be solved in time $\mathcal{O}^*(1.8139^n)$ consuming $\mathcal{O}^*(1.6563^n)$ space.

Proof. Run the above mentioned $\mathcal{O}^*(2^n)$ -algorithm until $|G^r| \leq \alpha \cdot n$ with $G^r := V \setminus \text{internal}(T)$. Then let $T^e = L[G^r, V(G^r) \cap \text{BN}_T]$. Note that the vertex $r \in V(T^e)$ must be internal and $y \in \text{leaves}(T^e)$. By Lemma 5.3.1 we can assume that $A(\{r\} \cup N^+(r)) \subseteq T^e$. Now identify the vertices $\text{BN}_T \cap V(T^e)$ with $V(G^r) \cap \text{BN}_T$ and delete r and y to a directed spanning tree \hat{T} for the original graph G . Or more formally let $\hat{T} := T \cup (T^e \setminus A(\{r\} \cup N^+(r)))$. Observe that \hat{T} extends T to optimality. \square

Note that in the first phase we cannot substitute the $\mathcal{O}^*(2^n)$ -algorithm by Algorithm 2. It might be the case that **(R6)** generates graphs which are not vertex-induced subgraphs of G .

5.4.2. Résumé

The paper at hand presented an algorithm which solves the DIRECTED MAXIMUM LEAF SPANNING TREE problem in time $\mathcal{O}^*(1.9044^n)$. Although this algorithm follows the same line of attack as the one of [51] the algorithm itself differs notably. The approach of [51] does not simply carry over. To achieve our run time bound we had to develop new algorithmic ideas. This is reflected by the greater number of branching cases.

Chapter 6.

Parameterized Measure&Conquer for k -Leaf Spanning Tree

6.1. Introduction.

We address the following problem in graphs:

k -LEAF SPANNING TREE

Given: An undirected graph $G(V, E)$, and the parameter k .

We ask: Is there a spanning tree for G with at least k leaves?

The problem has a notable applicability in the design of ad-hoc sensor networks [11, 150](J. Blum *et al.*, M. Thai *et al.*). In this area it might be referred to as CONNECTED DOMINATING SET. A spanning tree with k leaves is equivalent to a connected dominating set with $n - k$ vertices. The k -LEAF SPANNING TREE problem already has been widely studied with regard to its approximability. R. Solis-Oba [145] obtained a 2-approximation running in polynomial time. In almost linear time H.-I. Lu and R. Ravi [113] provided a 3-approximation. P.S. Bonsma and F. Zickfeld [15] could show that the problem is $\frac{3}{2}$ -approximable when the input is restricted to cubic graphs.

Concerning parameterized algorithms, a sequence of papers culminated in the one of J. Kneis, A. Langer and P. Rossmanith [98]. This fairly simple branching algorithm achieves a run time of $\mathcal{O}^*(4^k)$. Prior to this there were run time achievements by P.S. Bonsma *et al.* [14] of $\mathcal{O}^*(9.49^k)$, by V. Estivill-Castro *et al.* [40] of $\mathcal{O}^*(8.12^k)$ and by P.S. Bonsma and F. Zickfeld [16] of $\mathcal{O}^*(6.75^k)$. These bounds all have been obtained by using combinatorial arguments. The best kernelization result is due to [40] where they exhibited a kernel size of $3.75k$. I. Koutis and R. Williams [103] could derive a randomized $\mathcal{O}^*(2^k)$ -algorithm for the undirected version. H. Fernau *et al.* [51] gave an algorithm with run time $\mathcal{O}^*(1.8966^n)$ for undirected graphs.

There is also a directed version of the problem: Find an out-branching with k leaves. Here an out-branching in a directed graph is a tree in the underlying undirected graph. But the arcs are directed from the root to the leaves, which are the vertices of out-degree zero. The algorithm of J. Kneis, A. Langer and P. Rossmanith [98] solves also this problem in time $\mathcal{O}^*(4^k)$. Moreover, in Daligault *et al.* [32] an upper-bound of $\mathcal{O}^*(3.72^k)$ is stated. We are in the unusual situation that the run time for search tree algorithms for the directed case is no worse than the one obtained in the undirected case. By using rules that are specific for the undirected case, we are able to derive improved run times

valid for the undirected case only.

6.1.1. Our Contributions.

We developed the simple and elegant algorithm of [98] further. The run time improvement of $\mathcal{O}^*(3.4581^k)$ is due to two reasons: 1. We could improve the bottleneck case by new branching rules. 2. Due using amortized analysis, we were able to prove a tighter upper-bound on the run time. For this we use a non-standard measure which in its form is quite related to the *Measure&Conquer*-approach in exact, non-parameterized algorithmics, see Fomin, Grandoni and Kratsch [74]. Notice however that there are only few examples for using *Measure&Conquer* in parameterized algorithmics. In addition, we analyze our algorithm with respect to the number of vertices and obtain also small improvements for the MINIMUM CONNECTED DOMINATING SET problem. This seems to be the first attempt to analyze the same algorithm both with respect to the standard parameter k and with respect to the number of vertices n . We mention that the approaches of [32, 51] is going to some extent into the same direction. The basic scheme of the algorithms is similar. Nevertheless, our run time shows that our results are different. Moreover, the first paper does not make use of *Measure&Conquer* techniques and the second follows a non-parameterized route.

6.1.2. Terminology.

An edge $\{x, y\}$ might also be abbreviated as xy . An *edge cut set* is a subset $\hat{E} \subset E$ such that $G[E \setminus \hat{E}]$ is not connected. A *tree* is a subset of edges $T \subseteq E$ such that $G[T]$ is connected and cycle-free. A *spanning tree* is a tree such that $\bigcup_{e \in T} e = V$. A tree T' extends another tree T if $T \subseteq T'$. We will write $T' \succ T$. A *bridge* $e \in E$ leaves $G[E \setminus \{e\}]$ disconnected. For any $E' \subseteq E$ let $leaves(E') := \{v \in V \mid d_{E'}(v) = 1\}$ and $internal(E') := \{v \in V \mid d_{E'} \geq 2\}$.

6.1.3. Overall Strategy.

In the rest of the chapter, we address the following annotated version of our problem:

ROOTED k -LEAF SPANNING TREE

Given: An undirected graph $G(V, E)$ a root vertex $r \in V$, and the parameter k .

We ask: Is there a spanning tree T for G with $|leaves(T)| \geq k$ with $d_T(r) \geq 2$?

An algorithm solving this problem will also solve k -LEAF SPANNING TREE (with a polynomial delay) by considering every $v \in V$ as the root. All throughout the algorithm we will maintain a tree $T \subseteq E$ whose vertices are $V_T := \bigcup_{e \in T} e$. Let $\bar{V}_T := V \setminus V_T$. T will be seen as predetermined to be part of the solution. During the course of the algorithm, T will have two types of leaves. Namely, *leaf nodes (LN)* and *branching nodes (BN)*. The first mentioned will also appear as leaves in the solution. The latter ones can be leaves or internal vertices. Generally, we decide this by branching as far as reduction

rules do not enforce exactly one possibility. *Internal nodes* (IN) are already determined to be non-leaves in T .

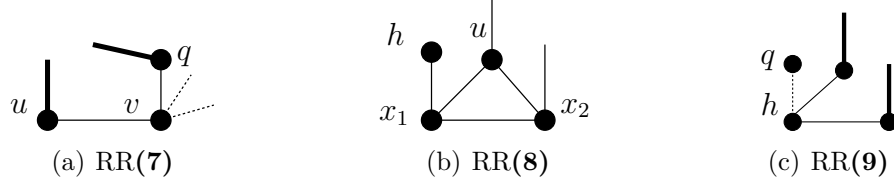
The algorithm will also produce a third kind of leaves: *floating leaves* (FL). These are vertices from \overline{V}_T which are already determined to be leaves, but are not attached to the tree T yet. If a vertex is neither a branching node nor a leaf node nor a floating leaf nor an internal node we call it *free*. We will refer to the different possible roles of a vertex by a labeling function $lab : V \rightarrow \{\text{IN}, \text{FL}, \text{BN}, \text{LN}, \text{free}\} := D$. A given tree T' defines a labeling $V_{T'} \rightarrow D$ to which we refer by $lab_{T'}$. Let $\text{IN}_{T'} := \{v \in V_{T'} \mid d_{T'}(v) \geq 2\}$, $\text{LN}_{T'} := \{v \in V_{T'} \mid d_{\overline{V}_{T'}}(v) = 0, d(v) = 1\}$ and $\text{BN}_{T'} = V_{T'} \setminus (\text{IN} \cup \text{LN})$. Then for any $ID \in D \setminus \{\text{FL}, \text{free}\}$ we have $ID_{T'} = lab^{-1}(ID)$. We always ensure that lab_T and lab are the same on V_T . The subscript might be suppressed if $T' = T$. If $T' \succ T$, then we assume that $\text{IN}_T \subseteq \text{IN}_{T'}$ and $\text{LN}_T \subseteq \text{LN}_{T'}$. So, the labels IN and LN remain once they are fixed. For the other labels, we have the following possible transitions: $\text{FL} \rightarrow \text{LN}$, $\text{BN} \rightarrow \{\text{LN}, \text{IN}\}$ and $\text{free} \rightarrow D \setminus \{\text{free}\}$. Subsequently, we assume $|V| > 4$.

6.2. Reduction Rules & Observations.

6.2.1. Reduction Rules.

We assume that reduction rule (i) is applied before (i+1). The rules (1)-(3) also appeared in previous work [51].

- (1) If there is an edge $e \in E \setminus T$ with $e \subseteq V_T$, then delete e .
- (2) Every $u \in \text{BN}$ with $d(u) = 1$ becomes a leaf node and every $u \in \text{free}$ with $d(u) = 1$ becomes a floating leaf.
- (3) Let $u \in \text{BN}$. If the removal of $E_{\overline{V}_T}(u)$ in $G[V \setminus \text{FL}]$ or $G[V]$ creates two components then u becomes internal.
- (4) Let u, v be free and assume that there is a bridge $\{u, v\} \in E \setminus T$ in $G[V]$, where C_1, C_2 are the two components created by deleting $\{u, v\}$. If $|V(C_1)| > 1$ and $|V(C_2)| > 1$ then contract $\{u, v\}$. The new vertex is also free.
- (5) Delete $\{u, v\} \in E$ if u and v are floating leaves.
- (6) Delete $\{u, v\} \in E \setminus T$ if $d_V(u) = 2$, $u \in \text{BN}$ and a) $d_V(v) = 2$, or b) $v \in \text{FL}$.
- (7) Delete $\{u, v\}$ if $u \in \text{BN}$ with $d(u) = 2$, $N_{\overline{V}_T}(u) = \{v\}$ and $d_{V_T}(v) \geq 2$, see Figure 6.1(a).
- (8) If u, x_1, x_2 form a triangle, x_1 is free and $\{h\} = N(x_1) \setminus \{x_2, u\}$ such that $d(h) = 1$, see Figure 6.1(b). Then x_1 becomes a floating leaf and h will be deleted.
- (9) Let $h \in \overline{V}_T$ be a free vertex such that a) $N_{\overline{V}_T}(h) = \{q\}$ and $d(q) = 1$ or b) $d_{\overline{V}_T}(h) = 0$, see Figure 6.1(c). Then h becomes a floating leaf and q is deleted in case a).


 Figure 6.1.: Bold edges are from T . Dotted edges may be present or not.

Lemma 6.2.1: The reduction rules are sound.

Proof. Let T' be a spanning tree with $T' \succ T$ such that $|\text{leaves}(T')| \geq k$.

- (1) Any edge $e \in E \setminus T$ with $e \subseteq V_T$ added to T would introduce a cycle.
- (2) In this case v must be a leaf node due to its degree constraint.
- (3) If $E_{\overline{V}_T}(u)$ is an edge cut set in $G[V]$, then u must be internal as we are looking for a spanning tree. Assume $E_{\overline{V}_T}(u)$ is a edge cut set in $G[V \setminus \text{FL}]$ but not in $G[V]$. If $\text{lab}_{T'}(u) = \text{LN}$ then there must be a $z \in \text{FL}$ with $d_{T'}(z) \geq 2$. Thus, $z \in \text{IN}_{T'}$, a contradiction.
- (4) Let G' be the graph which emerges by contracting $\{u, v\}$. Then we have $d_{T'}(u) \geq 2$ and $d_{T'}(v) \geq 2$ due to reasons of connectivity. By contracting $\{u, v\}$ in T' , we get a solution for G' . If G' has a spanning tree $T^* \succ T$ with k leaves, then clearly the same is true for G .
- (5) If $\{u, v\}$ is part of a solution T' then $T' \cong K_2$. This contradicts $|V| > 2$.
- (6.a) Suppose $e := \{u, v\} \in T'$ and u and v are not leaves. By removing e , u and v become leaves and T' is split in two components T'_1 and T'_2 . As e is not a bridge there is some $e' := \{a, b\} \in E \setminus T$ such that $T^* = T'_1 \cup T'_2 \cup \{e'\}$ is connected. Due to adjoining e' at most two leaves will become internal. Hence, $|\text{leaves}(T^*)| \geq |\text{leaves}(T')|$. It is possible that $\text{lab}(a) = \text{FL}$. But if it was true that for every such edge e' one of its endpoints is a floating leaf, then we could apply (3) as $E_{\overline{V}_T}(v)$ would be an edge cut set in $G[V \setminus \text{FL}]$. If $e := \{u, v\} \in T'$ and u or v is a leaf then the proof of (6.b) applies.
- (6.b) Suppose $e := \{u, v\} \in T'$. As e is not a bridge by (3) and (4) there is an $e' = \{v, x\} \in E \setminus T'$ with $x \neq u$ and $x \notin \text{FL}$. Let $T^* := (T' \setminus \{e\}) \cup \{e'\}$. Note that $|\text{leaves}(T^*)| \geq |\text{leaves}(T')|$ as u is a leaf in T^* and x is internal.
- (7) Let $q \in N_{V_T}(v) \setminus \{u\}$ and assume $e := \{u, v\} \in T'$. Then $e' := \{v, q\} \notin T'$ as $q \in V_T$. Let $T^* = (T' \setminus \{e\}) \cup \{e'\}$. Then $|\text{leaves}(T^*)| \geq |\text{leaves}(T')|$ as $\text{lab}_{T^*}(u) = \text{LN}$.
- (8) Let G^* be the reduced graph. In T' x_1 must be internal. Observe that we can assume that $d_{T'}(x_1) = 2$ (*). Otherwise, $\{u, x_1\}, \{x_1, x_2\}, \{x_1, h\} \in T'$, $\{u, x_2\} \notin T'$ and,

w.l.o.g., u is internal as $|V| > 4$. Then simply delete $\{x_1, x_2\}$ from T' and adjoin $\{u, x_2\}$. This way we can ensure (\otimes) . Due to (\otimes) we have that G^* has a spanning tree with k leaves iff G has one.

(9.a) Note that we have $d_{T'}(h) = 2$ since otherwise T' contains a cycle. Let G^* be the reduced graph. Analogously as in **(8)**, we can show that G^* has a spanning tree with k leaves iff G has one.

(9.b) If $d_{T'}(h) > 1$, then T' would possess a cycle. □

Lemma 6.2.2: Reduction rule **(1)** does not create bridges in $E \setminus T$.

Proof. Suppose an edge $e = \{u, v\} \notin T$ is deleted by **(1)** and a second $e' = \{x, y\} \in E \setminus T$ becomes a bridge in $G[E \setminus \{e\}]$. Then $G' := G[E \setminus \{e, e'\}]$ consists of two components G_1 and G_2 such that, w.l.o.g., $r, x \in V(G_1)$ and $y \in V(G_2)$. Thus, there is a simple path $P = rh_1 \dots h_\ell y$ in G such that $h_i \neq x, y$ ($1 \leq i \leq \ell$) and there is a $1 \leq j \leq \ell$ with, w.l.o.g., $h_j = u$ and $h_{j+1} = v$. As $u, v \in V_T$ there is a simple path P' in $G_1[T]$ from u to v such that $e, e' \notin E(P')$. Let $\hat{P} = rh_1 \dots h_{j-1} P' h_{j+2} \dots h_\ell y$. \hat{P} is a path in G' which connects r and y omitting e and e' . Thus, e' is not a bridge in $G[E \setminus \{e\}]$. □

From now on we assume that G is reduced due to the given reduction rules.

6.2.2. Observations.

If $N(\text{internal}(T)) \subseteq \text{internal}(T) \cup \text{leaves}(T)$, we call T an *inner-maximal* tree.

Lemma 6.2.3 ([98] Lemma 4.2): If there is a tree T' with $\text{leaves}(T') \geq k$ such that $T' \succeq T$ and $x \in \text{internal}(T')$ then there is a tree T'' with $\text{leaves}(T'') \geq k$ such that $T'' \succeq T$, $x \in \text{internal}(T'')$ and $\{\{x, u\} \in E\} \subseteq T''$

By the above lemma we can restrict our attention to inner-maximal spanning trees. And in fact the forthcoming algorithm will only construct such trees. Then for a $v \in \text{internal}(T)$ we have that $E_V(v) \subseteq T$ as by Lemma 6.2.3 we can assume that T is inner-maximal. Thus, in the very beginning we have $T = E_V(r)$.

Lemma 6.2.4: Let $v \in \text{BN}_T$ and $N_{\overline{V}_T}(v) = \{u\}$. Then u is free and $d_{\overline{V}_T}(u) \geq 2$.

Proof. Note that $d_V(v) = 2$. Moreover, $u \notin \text{IN} \cup \text{BN} \cup \text{FL}$ due to T 's inner-maximality, **(1)** and **(6.b)**. Thus, u is free. If $d_{\overline{V}_T}(u) = 0$ then **(9.b)** could be applied. If $d_{\overline{V}_T}(u) = 1$ then either we can apply **(3)** (if $\{u, v\}$ is a bridge) or **(6.a)** or **(7)** depending on whether $d_{V_T}(u) \geq 2$ or not. □

We are now going to define some function $co : \text{BN} \rightarrow V$. For $v \in \text{BN}$, let

$$co(v) = \begin{cases} v : d_{\overline{V}_T}(v) \geq 2 \\ u : N_{\overline{V}_T}(v) = \{u\} \end{cases}$$

Note that co is well defined on a reduced instance as we have $d_{\overline{V}_T}(v) \geq 1$ (otherwise it becomes a leaf node **(2)**). Note that we have $d_{\overline{V}_T}(co(v)) \geq 2$. Either $d_{\overline{V}_T}(v) \geq 2$

or $d_{\overline{V}_T}(v) = 1$. In the latter case, $N_{\text{free}}(v) = N_{\overline{V}_T}(v) = \{u\}$, such that $d_{\overline{V}_T}(u) \geq 2$ by Lemma 6.2.4. This property will be used frequently.

The next lemma has been shown by [98] and is presented using the introduced definitions of this chapter.

Lemma 6.2.5 ([98] Lemma 5): Let $v \in \text{BN}_T$ such that $N_{\overline{V}_T}(v) = \{u\}$. If there is no spanning tree $T' \succ T$ with k leaves and $\text{lab}_{T'}(v) = \text{LN}$, then there is also no spanning tree $T'' \succ T$ with k leaves, $\text{lab}_{T''}(v) = \text{IN}$ and $\text{lab}_{T''}(u) = \text{LN}$.

Observe that for a vertex v with $\text{co}(v) \neq v$ once we set $\text{lab}(v) = \text{IN}$ then it is also valid to set $\text{lab}(\text{co}(v)) = \text{IN}$. By Lemma 6.2.5 we must only ensure that we also consider the possibility $\text{lab}(v) = \text{LN}$.

A Further Reduction Rule With the assertion of Lemma 6.2.4 we state another reduction rule:

- (10)** Let $w \in \text{BN}$ with $x_1 \in N_{\text{free}}(w)$ such that a free degree one vertex q is attached to x_1 . Further, if
- a) there exists $v \in \text{BN}$ with $N_{\overline{V}_T}(v) = \{x_1, x_2\}$ and $\{x_1, x_2\} \in E$, see Figure 6.2(a), or
 - b) there exists $v \in \text{BN}$ with $\text{co}(v) \neq v$ and $N_{\overline{V}_T}(\text{co}(v)) = \{x_1, x_2\}$, see Figure 6.2(c), then set $\text{lab}(v) = \text{LN}$.

Lemma 6.2.6: Rule **(10)** is sound.

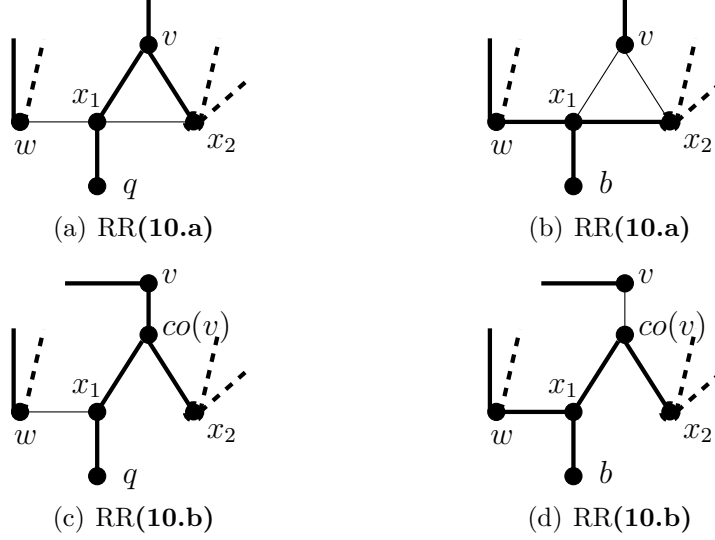
Proof. a). Let $T' \succ T$ be a spanning tree with $\text{lab}_{T'}(v) = \text{lab}_{T'}(\text{co}(v)) = \text{IN}$. Consider $T^* := (T' \setminus \{x_1 v, x_2 v\}) \cup \{w x_1, x_1 x_2\}$, see Figures 6.2(a) and 6.2(b). As $\text{lab}_{T'}(x_1) = \text{lab}_{T^*}(x_1) = \text{IN}$ and $\text{lab}_{T^*}(v) = \text{LN}$ we have $|\text{leaves}(T^*)| \geq |\text{leaves}(T')|$. Hence, we do not have to consider $\text{lab}(v) = \text{lab}(\text{co}(v)) = \text{IN}$.

b) We can skip the possibility $\text{lab}(v) = \text{lab}(\text{co}(v)) = \text{IN}$. Assume the contrary. In that case consider $T^* := (T' \setminus \{v \text{co}(v)\}) \cup \{x_1 w\}$, see Figures 6.2(c) and 6.2(d), with $|\text{leaves}(T^*)| \geq |\text{leaves}(T')|$ as x_1 must be internal. Note that $\text{lab}_{T^*}(v) = \text{LN}$. □

The next lemmas refer to the case where there is a $v \in \text{BN}_T$ with $d_{\overline{V}_T}(\text{co}(v)) = 2$. In the following we use the abbreviation $\mathcal{N} := \{\text{co}(v), x_1, x_2\}$.

Lemma 6.2.7: Let $T \subseteq E$ be a given tree such that $v \in \text{BN}_T$ and $N(\text{co}(v)) = \{x_1, x_2\}$. Let T', T^* be optimal spanning trees under the restrictions $T' \succ T$, $T^* \succ T$, $\text{lab}_{T'}(v) = \text{LN}$, $\text{lab}_{T^*}(v) = \text{lab}_{T^*}(\text{co}(v)) = \text{IN}$ and $\text{lab}_{T^*}(x_1) = \text{lab}_{T^*}(x_2) = \text{LN}$.

1. If there is a $z \in ((N(x_1) \cap N(x_2)) \setminus \mathcal{N})$, then $|\text{leaves}(T')| \geq |\text{leaves}(T^*)|$.
2. If $\text{co}(v) = v$, $y \in N(x_2) \setminus \mathcal{N}$, $z \in N(x_1) \setminus \mathcal{N}$ with $\text{lab}_{T^*}(z) = \text{IN}$, then $|\text{leaves}(T')| \geq |\text{leaves}(T^*)|$.

Figure 6.2.: Bold edges are from T . Dotted edges may be present or not.

3. If $co(v) \neq v$ and if there is a $z \in ((N(x_1) \cup N(x_2)) \setminus \mathcal{N})$ with $lab_{T^*}(z) = \text{IN}$, then $|leaves(T')| \geq |leaves(T^*)|$.

Proof. 1. Firstly, suppose $co(v) = v$. Consider $T^+ := (T^* \setminus \{v x_1, v x_2\}) \cup \{z x_1, z x_2\}$, see Figures 6.3(a) and 6.3(b). We have $lab_{T^+}(v) = \text{LN}$ and z can be the only vertex besides v where $lab_{T^+}(z) \neq lab_{T^*}(z)$. Thus, z could be the only vertex with $lab_{T^+}(z) = \text{IN}$ and $lab_{T^*}(z) = \text{LN}$. Therefore, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$. Secondly, if $co(v) \neq v$ then consider $T^\# := (T^* \setminus \{v co(v), co(v) x_2\}) \cup \{z x_1, z x_2\}$ instead of T^+ .

2. Consider $T^+ := (T^* \setminus \{v x_1, v x_2\}) \cup \{z x_1, y x_2\}$, see Figures 6.3(c) and 6.3(d). We have $lab_{T^+}(v) = \text{LN}$ and at most for y we could have $lab_{T^*}(y) = \text{LN}$ and $lab_{T^+}(y) = \text{IN}$. Hence, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$.
3. W.l.o.g., $z \in N(x_1) \setminus \mathcal{N}$. Consider $T^\# := (T^* \setminus \{v co(v)\}) \cup \{z x_1\}$. We have $lab_{T^\#}(v) = \text{LN}$ and therefore $|leaves(T')| \geq |leaves(T^\#)| \geq |leaves(T^*)|$. \square

Lemma 6.2.8: Let $T \subseteq E$ be a given tree such that $v \in \text{BN}_T$ and $N(co(v)) = \{x_1, x_2\}$. Let T', T^* be optimal spanning trees under the restrictions $T' \succ T$, $T^* \succ T$, $lab_{T'}(v) = \text{LN}$, $lab_{T^*}(v) = lab_{T^*}(co(v)) = \text{IN}$ and $lab_{T^*}(x_1) = \text{LN}$.

1. If $co(v) = v$, $\{x_1, x_2\} \in E$, $N(x_2) \setminus \text{FL} = \{v, x_1\}$ and if there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) = \text{IN}$, then $|leaves(T')| \geq |leaves(T^*)|$.
2. If $co(v) \neq v$, $N(x_2) \setminus \text{FL} \subseteq \{co(v), x_1\}$ and if there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) = \text{IN}$, then $|leaves(T')| \geq |leaves(T^*)|$.
3. If $lab_{T^*}(x_2) = \text{IN}$, $d_{T^*}(x_2) = 2$, $E_{\overline{v}T}(v)$ is not a edge cut-set in G and if there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) = \text{IN}$, then $|leaves(T')| \geq |leaves(T^*)|$.

Proof. 1. Consider $T^+ := (T^* \setminus \{v x_1, v x_2\}) \cup \{x_1 x_2, z x_1\}$. Note that $lab_{T^+}(v) = \text{LN}$ and x_1 is the only vertex where we have $lab_{T^*}(x_1) = \text{LN}$ and $lab_{T^+}(x_1) = \text{IN}$. Observe that T^+ is indeed a spanning tree. It is impossible that we have created a cycle, because x_1 is the only non-FL neighbor of x_2 in T^+ . Thus, $|leaves(T^+)| \geq |leaves(T^*)|$.

2. Consider $T^\# := (T^* \setminus \{v co(v)\}) \cup \{z x_1\}$ instead of T^+ from item 1.

3. First assume $co(v) = v$. Consider $T^+ := (T^* \setminus \{v x_1, v x_2\}) \cup \{z x_1\}$, see Figures 6.3(e) and 6.3(f). T^+ is a forest consisting of two trees T_1^+ and T_2^+ , where v and x_2 have become leaf nodes. Thus $|leaves(T^+)| - 2 = |leaves(T^*)|$. As $E_{\overline{V}_T}(v)$ is not a edge cut-set there is some $e \in E \setminus (T^+ \cup E_V(v))$ such that $T^{++} := T^+ \cup \{e\}$ is connected. Furthermore, T^+ has at most two leaf nodes more than T^{++} as the addition of e might turn at most two leaf nodes into internal nodes. Thus as $lab_{T^{++}}(v) = \text{LN}$, $|leaves(T^+)| \geq |leaves(T^{++})| \geq |leaves(T^*)|$.

If $co(v) \neq v$ then consider $T^\# := (T^* \setminus \{v co(v), co(v) x_2\}) \cup \{z x_1\}$ instead of T^+ . As $E_{\overline{V}_T}(v)$ is not an edge cut-set, $\{v, co(v)\}$ is not a bridge. Thus, there is an $e \in E \setminus (T^\# \cup \{\{v, co(v)\}\})$ such that $|leaves(T^\# \cup \{e\})| \geq |leaves(T^*)|$ and $T^\# \cup \{e\}$ is a spanning tree where v is LN-node. \square

In [98] the bottleneck case was when branching on a vertex $v \in BN$ with at most two non-tree neighbors, that is $d_{\overline{V}_T}(v) \leq 2$. The last two lemmas dealt with this case. If the bottleneck case also matches the conditions of Lemma 6.2.7 or 6.2.8 we either can skip some recursive call or decrease the yet to be defined measure by an extra amount. Otherwise we show that the branching behavior is more beneficial. This is a substantial ingredient for achieving a better run time upper bound.

6.3. The Algorithm.

We are now ready to present Algorithm 4. We mention that if the answer YES is returned a k -leaf spanning tree can be constructed easily. This will be guaranteed by Lemma 6.3.1. For the sake of a short presentation of the different branchings, we introduce the following notation $\langle b_1; b_2; \dots; b_n \rangle$ called a *branching*. Here the entries b_i are separated by a semicolon and stand for the different *parts of the branching*. They will express how the label of some vertices change. For example: $\langle v \in \text{LN}; v, co(v) \in \text{IN} \rangle$. This stands for a binary branching where in the first part we set $lab(v) = \text{LN}$ and in the second $lab(v) = lab(co(v)) = \text{IN}$.

6.3.1. Correctness.

6.3.1.1. Branchings

When we set $lab(v) = \text{IN}$ then we also set $T \leftarrow T \cup \{\{u, v\} \in E \mid u \notin V_T\}$. This is justified by Lemma 6.2.3. If we set $lab(v) = \text{LN}$, then we delete $\{\{u, v\} \in E \mid \{u, v\} \notin T\}$ as these edges will never appear in any solution.

Algorithm 4: An algorithm solving k -LEAF SPANNING TREE**Data:** A graph $G = (V, E)$, k and a tree $T \subseteq E$.**Result:** YES if there is a spanning tree with at least k leaves and NO otherwise.

```

1  if  $\kappa(G) \leq 0$  or  $|BN| + |LN| \geq k$  then
2  |   return YES
3  else if  $G[V \setminus FL]$  is not connected or  $BN = \emptyset$  then
4  |   return NO
5  else
6  |   Apply the reduction rules exhaustively
7  |   Choose a vertex  $v \in BN$  of maximum degree
8  |   if  $d_{\overline{V}_T}(co(v)) \geq 3$  then
9  |   |    $\langle v \in LN; v, co(v) \in IN \rangle$  (B1)
10 |   else if  $N_{\overline{V}_T}(co(v)) = \{x_1, x_2\}$  then
11 |   |   Choose  $v$  according to the following priorities:
12 |   |   case  $(\{x_1, x_2\} \subseteq FL)$  or (B2.a)
13 |   |    $(x_1 \text{ free} \ \& \ d_{\overline{V}_T \setminus \mathcal{N}}(x_1) = 0)$  or (B2.b)
14 |   |    $(x_1 \text{ free} \ \& \ N_{\overline{V}_T \setminus \mathcal{N}}(x_1) = \{z\} \ \& \ (d_{\overline{V}_T \setminus \mathcal{N}}(z) \leq 1 \text{ or } z \in FL))$  (B2.c)
15 |   |   |    $\langle v \in LN; v, co(v) \in IN \rangle$  (B2)
16 |   |   case  $x_1 \text{ free}, x_2 \in FL$  or (B3.a)
17 |   |    $x_1, x_2 \text{ free}, N_{\overline{FL}}(x_2) \subseteq \{x_1, co(v)\}$  or (B3.b)
18 |   |    $x_1, x_2 \text{ free} \ \& \ d_{\overline{V}_T \setminus \mathcal{N}}(x_2) = 1$  (B3.c)
19 |   |   |    $\langle v \in LN; v, co(v) \in IN, x_1 \in LN; v, co(v), x_1, co(x_1) \in IN \rangle$  (B3)
20 |   |   |   and apply makeleaves $(x_1, x_1)$  in the 2nd branch
21 |   |   case  $x_1, x_2 \text{ free} \ \& \ \exists z \in \bigcap_{i=1,2} N_{\overline{FL} \setminus \mathcal{N}}(x_i)$ 
22 |   |   |    $\langle v \in LN; v, co(v), x_2, co(x_2) \in IN, x_1 \in LN; v, co(v), x_1, co(x_1) \in IN \rangle$ 
23 |   |   |   (B4)
24 |   |   otherwise
25 |   |   |    $\langle v \in LN; v, co(v) \in IN, x_1, x_2 \in LN; v, co(v), x_2, co(x_2) \in IN, x_1 \in$ 
26 |   |   |    $LN; v, co(v), x_1, co(x_1) \in IN \rangle$  (B5)
26 |   |   |   and apply makeleaves $(x_1, x_2)$  in the 2nd branch

```

Procedure **makeleaves (x_1, x_2)**

```

1  begin
2  |    $\forall u \in [(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}] \cap \text{free set } u \in FL;$ 
3  |    $\forall u \in [(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}] \cap \text{BN set } u \in LN;$ 
4  end

```

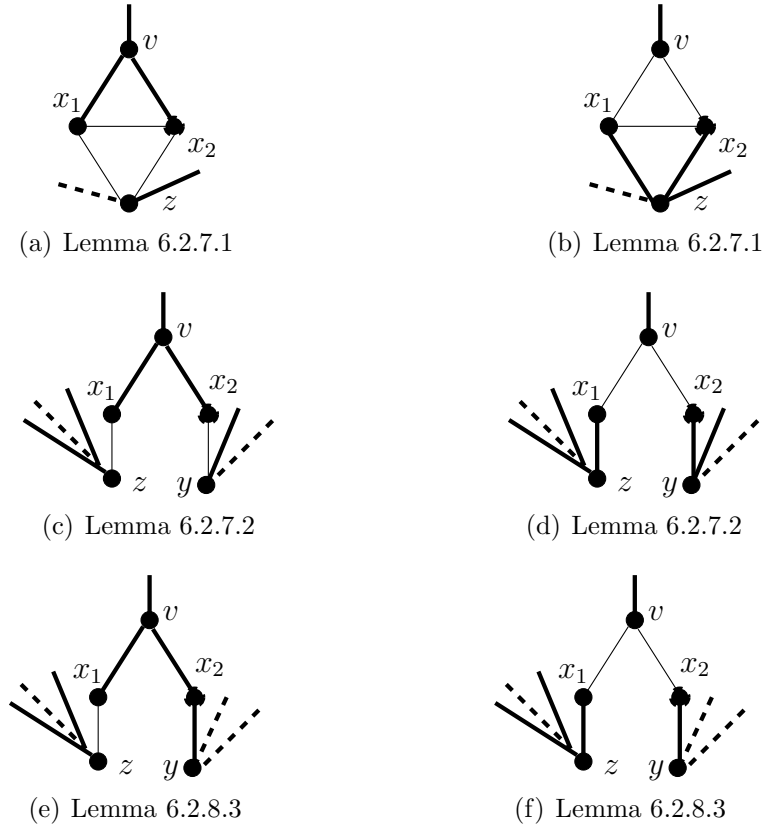


Figure 6.3.: Bold edges are from T . Dotted edges may be present or not.

In every branching of our algorithm, the possibility that $lab(v) = \text{LN}$ is considered. This recursive call must be possible as otherwise **(3)** would have been triggered before. Now consider the case $co(v) \neq v$. If the recursive call for $lab(v) = \text{LN}$ does not succeed, then we consider $lab(v) = \text{IN}$. Due to Lemma 6.2.5 we immediately can also set $lab(co(v)) = \text{IN}$. This fact is used throughout the branchings (B1)-(B5). Nonetheless, in the branchings (B1), (B2), (B3) and (B5) every possibility for v , x_1 and x_2 is considered in one part of the branching, i.e., these are exhaustive branchings. But in (B3) and (B5) the procedure `makeleaves()` is invoked and (B4) is not exhaustive. We will go through each subcase and argue that this is correct in a way that at least one optimal solution is preserved.

B3.a) In the second part of the branch, every vertex in $(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}$ can be assumed to be a leaf node in the solution. Otherwise, due to Lemmas 6.2.7.2 and 6.2.7.3, a solution, which is no worse than the neglected one, can be found in the first part of the branch when we set $lab(v) = \text{LN}$. Therefore the application of `makeleaves(x_1, x_1)` is correct.

B3.b) There is a $z_1 \in N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_1)$ (by **(3)**). If $co(v) = v$ then we must have $\{x_1, x_2\} \in E$ due to **(3)**. Thus, either Lemma 6.2.8.1 or 6.2.8.2 apply (depending whether

$co(v) = v$ or not). As in the previous item it follows that the application of $\text{makeleaves}(x_1, x_1)$ is correct.

B3.c) Let $z_1 \in N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_1)$ and $\tilde{T} \succ T$ be an optimal spanning tree solution extending T such that $lab_{\tilde{T}}(v) = lab_{\tilde{T}}(co(v)) = \text{IN}$ and $lab_{\tilde{T}}(x_1) = \text{LN}$. If $lab_{\tilde{T}}(x_2) = \text{LN}$ then Lemmas 6.2.7.2 and 6.2.7.3 apply. This means in the branch setting $v, co(v) \in \text{IN}, x_1 \in \text{LN}$, we can assume that vertices in $(N(x_1 \cup N(x_2)) \setminus \mathcal{N})$ are leaves, i.e., we can adjoin them to FL or LN. If $lab_{\tilde{T}}(x_2) = \text{IN}$ then we must have $d_{\tilde{T}}(x_2) = 2$. Thus, Lemma 6.2.8.3 applies. This can be seen as follows.

Let us recall the situation: $lab_{\tilde{T}}(v) = lab_{\tilde{T}}(co(v)) = \text{IN}$, $lab_{\tilde{T}}(x_1) = \text{LN}$ and $lab_{\tilde{T}}(x_2) = \text{IN}$. Thus, $d_{\tilde{T}}(x_2) = 2$. $E_{\overline{V}_T}(v)$ is not an edge cut set in $G[V]$ nor in $G[V \setminus \text{FL}]$ due to **(3)**.

Let us reconsider the proof of Lemma 6.2.8.3, especially how T^{++} is created in case $co(v) = v$. We join two subtrees T_1^+ and T_2^+ by an edge $e := \{x, y\} \in E \setminus \{T^+, E_V(v)\}$. In our algorithm we additionally have a labeling. So, it might be that $lab(x) = \text{FL}$. As our algorithm never turns a floating leaf into an internal node, it never will add e . Thus, we would not find T^{++} . But also observe that if for every such edge joining T_1^+ and T_2^+ this holds then $E_{\overline{V}_T}(v)$ is an edge cut set in $G[V \setminus \text{FL}]$.

If $co(v) \neq v$ we also look for an edge $e := \{x, y\} \in E \setminus \{T^+, E_V(v)\}$ to join two trees. If this is not possible this also would imply that $\{v, co(v)\}$ is an edge cut set in $G[V \setminus \text{FL}]$.

In both cases this cannot happen by **(3)**. Thus, we can choose e in a way that $lab(x) \neq \text{FL}$ and $lab(y) \neq \text{FL}$. This shows that Lemma 6.2.8.3 indeed can be applied. Thus, in the second part of the branch we can assume that every vertex in $N(x_1) \setminus \mathcal{N}$ is a leaf. Any solution which violates this assumption can be substituted by a no worse one where $v \in \text{LN}$ which is assured by Lemma 6.2.8.3. This justifies to call $\text{makeleaves}(x_1, x_1)$.

(B4) does not consider the possibility that $lab(v) = \text{IN}$ and $lab(x_1) = lab(x_2) = \text{LN}$. Here we refer to Lemma 6.2.7.1, which states that a no worse solution can be found when we set $lab(v) = \text{LN}$.

(B5) If $co(v) = v$ then by **(3)** $N(x_i) \setminus \{v\} \neq \emptyset$, if $co(v) \neq v$ then analogously $\bigcup_{i=1}^2 N(x_i) \setminus \mathcal{N} \neq \emptyset$. Then due to Lemmas 6.2.7.2 and 6.2.7.3 an application of $\text{makeleaves}(x_1, x_2)$ is valid. Note that Lemmas 6.2.7.2 and 6.2.7.3 can also be read with exchanged roles between x_1 and x_2 .

6.3.1.2. The Measure

To derive an upper-bound on the run time for our algorithm, we use the measure

$$\kappa(G) := k - \omega_f \cdot |\text{FL}| - \omega_b \cdot |\text{BN}| - |\text{LN}| \text{ with } \omega_b = 0.5130 \text{ and } \omega_f = 0.4117.$$

$\kappa(G)$ is defined by a tree T and a labeling (which both are to be built up by our algorithm). Thus, we use a subscript when we are referring to this, i.e., $\kappa(G)_T$.

When our algorithm returns YES, then T might still not be a spanning tree. We first have to attach all the floating leaves to T . It is possible that a branching node turns into an internal node and thus $\kappa(G)$ increases. The next important lemma shows if we take all the floating leaves and branching nodes into account then $\kappa(G)$ decreases.

Lemma 6.3.1: If for a given a labeling $\kappa(G) \leq 0$, then a spanning tree \hat{T} with $|\text{leaves}(\hat{T})| \geq k$ can be constructed in polynomial time.

Proof. Delete the vertices in FL and compute a depth-first spanning tree DT for the remaining graph starting from T . Then simply attach the vertices from FL to one of its neighbors in DT . This way we obtain a spanning tree $\hat{T} \succ T$. Let $\text{LBN} = \text{BN}_T \cap \text{LN}_{\hat{T}}$ and $\text{IBN} = \text{BN}_T \setminus \text{LBN}$. For $c \in \text{IBN}$ let T_c be the subtree rooted at c in \hat{T} . Clearly, $|\text{leaves}(T_c)| \geq 1$ (\star). Observe that every vertex $v \in \text{FL} \cup \text{LBN}$ now has weight zero. Thus, $\kappa(G)_{\hat{T}}$ was decreased by $(1 - \omega_f)$ or $(1 - \omega_b)$, resp., with respect to $\kappa(G)_T$ due to making v a leaf node. Due to the next inequality we have $|\text{leaves}(\hat{T})| \geq k$.

$$\begin{aligned} k - |\text{LN}_{\hat{T}}| = \kappa(G)_{\hat{T}} &\leq \kappa(G)_T - |\text{LBN}| \cdot (1 - \omega_b) + |\text{IBN}| \cdot \omega_b - |\text{FL}| \cdot (1 - \omega_f) \\ &\leq \kappa(G)_T + |\text{IBN}| \cdot \omega_b - \sum_{c \in \text{IBN}} |\text{leaves}(T_c)| \cdot (1 - \omega_f) \\ &\leq \kappa(G)_T + |\text{IBN}| \cdot (\omega_b + \omega_f - 1) \leq \kappa(G)_T \leq 0 \end{aligned} \quad (\text{by } (\star))$$

□

Next we consider the interaction of the reduction rules with the measure.

Lemma 6.3.2: An exhaustive application of the reduction rules never increases $\kappa(G)$.

Proof. Rule (2) decreases $\kappa(G)$ by $1 - \omega_b$ or ω_f . The deletion of an edge from $E \setminus T$ can cause a change in $\kappa(G)$ only in one way: a branching node can become a leaf node. Then $\kappa(G)$ is decreased by $1 - \omega_b$. If one of (1), (5), (6) and (7) is applied then exactly such an edge is deleted. If (10) is applied then a vertex $v \in \text{BN}$ becomes a leaf node. Thus, one or more edges are deleted and therefore $\kappa(G)$ is not increasing. In (3) there is a vertex $u \in \text{BN}$ which becomes internal. Thus, $\kappa(G)$ increases by ω_b . Moreover, there is a second vertex $q \in N_{\overline{V}_T}(u)$. If q is free then it becomes a branching node and $\kappa(G)$ decreases by ω_b . Thus, $\kappa(G)$ does not change if we sum up both amounts. If q is a floating leaf then it becomes a leaf node. Thus $\kappa(G)$ decreases by $1 - \omega_f - \omega_b > 0$. As in (4) an edge with with two free vertices is contracted such that the resulting vertex is also free, $\kappa(G)$ remains the same. In (8) a free vertex becomes a floating leaf and a vertex h with degree one is deleted. As we have $h \in \text{FL}$, $\kappa(G)$ does not change. Rule (9.a) can be analyzed analogously. In (9.b) a free vertex turns in a floating leaf and $\kappa(G)$ decreases by ω_f . □

6.3.2. Run Time Analysis.

In our algorithm the changes of $\kappa(G)$ are due to reduction rules or branching. In the first case Lemma 6.3.2 ensures that $\kappa(G)$ will never increase. In the second case we have reductions of $\kappa(G)$ of the following type. When a vertex $v \in \text{BN}$ is made a leaf node (i.e., we set $\text{lab}(v) = \text{LN}$) then $\kappa(G)$ will drop by an amount of $(1 - \omega_b)$. On the other hand when v becomes an internal node (i.e., we set $\text{lab}(v) = \text{IN}$) then $\kappa(G)$ will increase by ω_b . This is due to v not becoming a leaf. Moreover, the free neighbors of v become branching nodes and the floating leaves become leaf nodes, due to Lemma 6.2.3. Therefore $\kappa(G)$ will be decreased by ω_b and $1 - \omega_f$, respectively. We point out that the weights ω_b and ω_f have to be chosen such that $\kappa(G)$ will not increase in any part of a branching of our algorithm.

Analyzing the Different Branching Cases

(B1) Let $i := |N_{\overline{V}_T}(\text{co}(v)) \cap \text{FL}|$ and $j := |N_{\overline{V}_T}(\text{co}(v)) \cap \text{free}|$. Note that $i + j \geq 3$. Then the branching vector is: $(1 - \omega_b, i \cdot (1 - \omega_f) + j \cdot \omega_b - \omega_b)$.

(B2) a) The branching vector is $(1 - \omega_b, 2 \cdot (1 - \omega_f) - \omega_b)$.

b) When we set $\text{lab}(v) = \text{lab}(\text{co}(v)) = \text{IN}$ the vertex x_1 will become a leaf node due to **(1)**. The branching vector is $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$. The vertex x_2 contributes an amount of $\min\{1 - \omega_f, \omega_b\}$ depending on whether $x_2 \in \text{FL}$ or $x_2 \in \text{BN}$.

c) Firstly, suppose that $z \in \text{FL}$.

1. $d(z) = 1$:

a) $\text{co}(v) = v$: If $\{x_1, x_2\} \notin E$ then either **(9.a)** or **(3)** applies (depending whether $d_{V_T}(x_1) > 1$). If $\{x_1, x_2\} \in E$ then there is $z_1 \in N_{V_T}(x_1) \setminus \{v\}$ as otherwise **(8)** applies. But then **(10.a)** applies.

b) $\text{co}(v) \neq v$: If $d_{V_T}(x_1) = 0$ then either **(8)** or **(4)** applies (depending whether $\{x_1, x_2\} \in E$). If $d_{V_T}(x_1) > 0$ then **(10.b)** applies.

2. $d(z) \geq 2$: After setting $\text{lab}(v) = \text{lab}(\text{co}(v)) = \text{IN}$ and applying **(1)** exhaustively we have $d(x_1) = 2$ and $x_1, x_2 \in \text{BN}$ afterwards. Observe that adding an edge to T does not create a bridge. The same holds for rule **(1)** (Lemma 6.2.2). Thus, rules **(3)** or **(4)** are not triggered before the rules with lower priority. As **(2)** and **(5)** do not change the local setting with respect to x_1 and z **(6.b)** will delete $\{x_1, z\}$, leading to a $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$ branch.

The case that $z \in \text{free}$ can be seen by similar arguments. Observe that $d_G(z) \geq 2$ by **(2)**. In the part where we set $v, \text{co}(v) \in \text{IN}$ we obtain a tree $T' := T \cup E_{\overline{V}_T}(\text{co}(v))$. Then rule **(1)** will delete edges incident to x_1 such that $d_V(x_1) = 2$ and $x_1 \in \text{BN}_{T'}$.

1. If $d_{T'}(z) \geq 2$ then **(7)** deletes $\{x_1, z\}$.

2. If $d_{T'}(z) = 1$ then by $d_G(z) \geq 2$ and $d_{\overline{V}_T \setminus \mathcal{N}}(z) \leq 1$ we deduce that $d_G(z) = 2$ before setting $v, \text{co}(v) \in \text{IN}$. Note that $x_2 \notin N(x)$ as $x_2 \in V_{T'}$. Hence, **(6.a)**

deletes $\{x_1, z\}$ afterwards.

Hence we have a $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$ branch. We point out that it is guaranteed that no reduction rule triggered after **(1)** of lower priority than **(6)** and **(7)** will change the local situation with respect to x_1 and z (note Lemma 6.2.2).

Remark 6.3.3: From this point on, w.l.o.g., for a free vertex x_i , $i = 1, 2$, we have:

1. $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) \geq 2$ or
2. $N_{\overline{V}_T \setminus \mathcal{N}}(x_i) = \{z_i\}$ such that $d_{\overline{V}_T \setminus \mathcal{N}}(z_i) \geq 2$ and $z_i \notin \text{FL}$.

If $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) = 0$ then (B2.b) would apply. If $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) = 1$ and 2. would fail, then case (B2.c) applied.

Remark 6.3.4: Note that if 2. applies to x_i , then when we set $\text{lab}(v) = \text{lab}(\text{co}(v)) = \text{IN}$ we have that $\text{co}(x_i) = z_i$ after the application of the reduction rules. In this sense (with a slight abuse of notation) we set $\text{co}(x_i) = x_i$ if case 1. applies and $\text{co}(x_i) = z_i$ if case 2. applies.

(B3) If x_i is free let $fl_i := |(N(\text{co}(x_i)) \setminus \mathcal{N}) \cap \text{FL}|$ and $fr_i := |(N(\text{co}(x_i)) \setminus \mathcal{N}) \cap \text{free}|$.

Due to Remark 6.3.3 we have $fl_i + fr_i \geq 2$.

a) Note that we must have that $S := N_{\overline{\text{FL}}}(x_1) \setminus \mathcal{N} \neq \emptyset$ due to **(3)**. If $\text{co}(v) = v$, then also $N(x_i) \setminus \{v\} \neq \emptyset$ ($i = 1, 2$) due to **(3)**. Hence, in the second branch for every $q \in S$ we get ω_f if $\text{lab}(q) = \text{free}$ as we set $\text{lab}(q) = \text{FL}$ in $\text{makeleaves}(x_1, x_1)$. If $\text{lab}(q) = \text{BN}$, we set $\text{lab}(q) = \text{LN}$ in $\text{makeleaves}(x_1, x_1)$ and receive $1 - \omega_b$. We have the following reduction in $\kappa(G)$ for the different parts of the branching.

$$v \in \text{LN}: 1 - \omega_b.$$

$$v \in \text{IN}, \text{co}(v) \in \text{IN}, x_1 \in \text{LN}: 1 + \min\{\omega_f, 1 - \omega_b\} + 1 - \omega_f - \omega_b$$

$$v, \text{co}(v), x_1, \text{co}(x_1) \in \text{IN}: 1 - \omega_f + fl_1(1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b.$$

Remark 6.3.5: Note that from this point we have that x_1 and x_2 are free.

b) Note that we must have that $S := N_{\overline{\text{FL}}}(x_1) \setminus \mathcal{N} \neq \emptyset$ due to **(3)**. Analogously as in a) we obtain $\min\{\omega_f, 1 - \omega_b\}$ in addition from $N(x_1) \setminus \mathcal{N}$ in the second part of the branch. Thus, we have the branching vector

$$(1 - \omega_b, 1 + \min\{\omega_f, 1 - \omega_b\} + \omega_b - \omega_b, \omega_b + fl_1(1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b)(\diamond)$$

Remark 6.3.6: Observe that from now on there is always a vertex $z_i \in N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_i)$ ($i = 1, 2$) due to the previous case.

c) This entails the same branch as in (\diamond) .

(B4) Due to Lemma 6.2.7.1 a

$(1 - \omega_b, 1 + fr_2 \cdot \omega_b + fl_2 \cdot (1 - \omega_f) - \omega_b, \omega_b + fr_1 \cdot \omega_b + fl_1 \cdot (1 - \omega_f) - \omega_b)$ -branch can be derived.

(B5) In this case $\bigcap_{i=1,2} N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_i) = \emptyset$ is true as otherwise (B4) applies. This means for $i = 1, 2$ there are two different vertices $z_i \in N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_i)$ (Remark 6.3.6). Due to (B3.c) we have $d_{\overline{\text{V}}_T \setminus \mathcal{N}}(x_i) \geq 2$. Thus, in the second part we additionally get $(fr_1 + fr_2) \cdot \omega_f$. If $fr_i = 0$ we get an amount of $1 - \omega_b$ by Remark 6.3.6.

$v \in \text{LN}$: $1 - \omega_b$.

$v, co(v) \in \text{IN}, x_1, x_2 \in \text{LN}$: $2 + (fr_1 + fr_2) \cdot \omega_f + (\max\{0, 1 - fr_1\} + \max\{0, 1 - fr_2\}) \cdot (1 - \omega_b) - \omega_b$

$v, co(v), x_2, co(x_2) \in \text{IN}, x_1 \in \text{LN}$: $1 + fl_2 \cdot (1 - \omega_f) + fr_2 \cdot \omega_b - \omega_b$

$v, co(v), x_1, co(x_1) \in \text{IN}$: $\omega_b + fl_1 \cdot (1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b$.

We have calculated the branching number for every mentioned recursion such that $2 \leq fr_i + fl_i \leq 5$, $i = 1, 2$, with respect to ω_b and ω_f . The branching number of any other recursion is upper-bounded by one of these. We mention the bottleneck cases which attain the given run time:

Cases (B3.b)/(B3.c) and Case (B5) such that $d_{\overline{\text{V}}_T \setminus \mathcal{N}}(x_i) = 2$ and a) $fl_1 = fl_2 = 0$, $fr_1 = fr_2 = 2$, b) $fr_1 = 2$, $fl_1 = 0$, $fr_2 = fl_2 = 1$; Compared to [32] case (B5) has been improved. We can find at least two vertices which are turned from free vertices to floating leaves or from branching nodes to leaf nodes. In [32] only one vertex with this property can be found in the worst case. Thus, due to the previous case analysis and the fact that the reduction rules can be executed in polynomial time, we can state our main result:

Theorem 6.3.7: k -LEAF SPANNING TREE can be solved in time $\mathcal{O}^*(3.4581^k)$.

6.4. An Exact Exponential Time Analysis

We stated the run time of Alg 4 in terms of k where k is the number of leaves in the spanning tree.

Exponential Time Analysis. Fomin, Grandoni and Kratsch [73] gave an exact exponential-time algorithm with run time $\mathcal{O}^*(1.9407^n)$. Based on and re-analyzing the parameterized algorithm of Kneis, Langer and Rossmanith [98], this was improved to $\mathcal{O}^*(1.8966^n)$, see [51]. We can show further (slight) improvements by re-analyzing our parameterized algorithm. Therefore, we define a new measure:

$$\tau := n - \tilde{\omega}_f \cdot |\text{FL}| - \tilde{\omega}_b \cdot |\text{BN}| - |\text{LN}| - |\text{IN}| \text{ with } \tilde{\omega}_b = 0.2726 \text{ and } \tilde{\omega}_f = 0.5571.$$

The remaining task is quite easy. We only have to adjust the branching vectors we derived with respect to $\kappa(G)$ to τ .

$$\begin{aligned}
 (B1) \quad & : \quad (1 - \tilde{\omega}_b, i \cdot (1 - \tilde{\omega}_f) + j \cdot \tilde{\omega}_b + 1 - \tilde{\omega}_b). \\
 (B2) \ a) \quad & : \quad (1 - \tilde{\omega}_b, 2 \cdot (1 - \tilde{\omega}_f) + 1 - \tilde{\omega}_b). \\
 (B2) \ b)/c) \quad & : \quad (1 - \tilde{\omega}_b, 1 + \min\{1 - \tilde{\omega}_f, \tilde{\omega}_b\} + 1 - \tilde{\omega}_b). \\
 (B3) \ a) \quad & : \quad (1 - \tilde{\omega}_b, 1 + \min\{\tilde{\omega}_f, 1 - \tilde{\omega}_b\} + 1 - \tilde{\omega}_f + 1 - \tilde{\omega}_b, \\
 & \quad fl_1 \cdot (1 - \tilde{\omega}_f) + fr_1 \cdot \tilde{\omega}_b + 1 - \tilde{\omega}_b + 1 - \tilde{\omega}_f + 1). \\
 (B3) \ b)/c) \quad & : \quad (1 - \tilde{\omega}_b, 1 + \tilde{\omega}_f + \tilde{\omega}_b + 1 - \tilde{\omega}_b, \\
 co(x_1) \neq x_1 \quad & \tilde{\omega}_b + fl_1 \cdot (1 - \tilde{\omega}_f) + fr_1 \tilde{\omega}_b + 1 - \tilde{\omega}_b + 2). \\
 (B3) \ b)/c) \quad & : \quad (1 - \tilde{\omega}_b, 1 + fr_1 \cdot \tilde{\omega}_f + \max\{0, 1 - fr_1\} \cdot (1 - \tilde{\omega}_b) + \tilde{\omega}_b + 1 - \tilde{\omega}_b, \\
 co(x_1) = x_1 \quad & \tilde{\omega}_b + fl_1 \cdot (1 - \tilde{\omega}_f) + fr_1 \cdot \tilde{\omega}_b + 1 - \tilde{\omega}_b + 1). \\
 (B4) \quad & : \quad (1 - \tilde{\omega}_b, 1 + fr_2 \cdot \tilde{\omega}_b + fl_2 \cdot (1 - \tilde{\omega}_f) + 1 - \tilde{\omega}_b + 1, \\
 & \quad \tilde{\omega}_b + fl_1 \cdot (1 - \tilde{\omega}_f) + fr_1 \cdot \tilde{\omega}_b + 1 + 1 - \tilde{\omega}_b). \\
 (B5) \quad & : \quad (1 - \tilde{\omega}_b, 2 + 1 - \tilde{\omega}_b + (fr_1 + fr_2) \cdot \tilde{\omega}_f + \\
 & \quad (\max\{0, 1 - fr_1\} + \max\{0, 1 - fr_2\}) \cdot (1 - \tilde{\omega}_b), \\
 & \quad 2 + 1 - \tilde{\omega}_b + fl_2 \cdot (1 - \tilde{\omega}_f) + fr_2 \cdot \tilde{\omega}_b, \\
 & \quad \tilde{\omega}_b + 1 - \tilde{\omega}_b + 1 + fl_1 \cdot (1 - \tilde{\omega}_f) + fr_1 \cdot \tilde{\omega}_b)
 \end{aligned}$$

Note that in cases (B3) b) and (B3) c) we were making a case distinction based upon $co(x_1)$. If $co(x_1) \neq x_1$ then in the branch where we set $v, co(v), x_1, co(x_1) \in \text{IN}$ we can decrease τ by one more as $co(x_1)$ can be counted additionally. If $co(x_1) = x_1$ then $fr_1 \cdot \tilde{\omega}_f + \max\{0, 1 - fr_1\} \cdot (1 - \tilde{\omega}_b)$ is the least amount by which τ is reduced due to the application of $\text{makeleaves}(x_1, x_1)$.

It can be checked that every branching number of the above recursions is upper bounded by $\mathcal{O}^*(1.8961^\tau)$.

Theorem 6.4.1: MAXIMUM LEAF SPANNING TREE can be solved in $\mathcal{O}^*(1.8961^n)$.

6.5. Conclusions.

Parameterized Measure & Conquer. Amortized search tree analysis, also known as *Measure & Conquer*, is a big issue in exact, non-parameterized algorithmics. Although search trees play an important role in exact parameterized algorithmics, this kind of analysis has been rather seldom applicable. Good examples are the papers of Fernau and Raible [54], which deals with MAXIMUM ACYCLIC SUBGRAPH, the analysis of 3-HITTING-SET in Wahlström's PhD Thesis [155] and the amortized analysis of cubic vertex cover by Chen, Kanj and Xia [22]. This result contributes to this topic. Let us emphasize the difference to the, say, non-parameterized *Measure&Conquer* and to this case. Usually if a measure μ , which is used to derive an upper bound of the form $c^{|\mathcal{V}|}$, is decreased to zero then we immediately have a solution. Almost all time this is quite

clear because then the instance is polynomial-time solvable. Now if the parameterized measure $\kappa(G)$ is smaller than zero then in general a hard sub-instance remains. Also $\kappa(G)$ has been decreased due to producing floating leaves, which are not attached to the tree yet. Thus, it is crucial to have Lemma 6.3.1, which ensures that a k -leaf spanning tree can be indeed constructed. Beyond that, it is harder to show that no reduction rules ever increase $\kappa(G)$. As vertices which have been counted already partly (e.g., because they belong to $FL \cup BN$) can be deleted, $\kappa(G)$ can even increase temporarily. Concerning the traditional approach this is a straight-forward task and is hardly ever mentioned. It is a challenge to find further parameterized problems where this, say, parameterized Measure & Conquer paradigm can be applied.

As there is a linear kernel [40] we can first kernelize the input graph in polynomial time. The kernel size is no more than $3.75k$. Thus, run our algorithm on the kernel yields a run time upper-bound of $\mathcal{O}(3.4581^k + poly(n))$. Notice that the right choice for ω_f and ω_b is quite crucial. For example, setting $\omega_f = \omega_b = 0.5$ only shows a run time upper-bound of $\mathcal{O}^*(3.57^k)$. To find these beneficial values, a local search procedure was executed on a computer. It is worth pointing out that our algorithm is quite explicit. This means that its statement to some extent lengthy but on the other hand easier to implement. The algorithm does not use compact mathematical expressions which might lead to ambiguities in the implementation process.

Chapter 7.

Breaking the 2^n -Barrier for irredundance

7.1. Introduction

A set $I \subseteq V$ is called an *irredundant set* of a graph $G = (V, E)$ if each $v \in I$ is either isolated in $G[I]$, the subgraph induced by I , or there is at least one vertex $u \in V \setminus I$ with $N(u) \cap I = \{v\}$, called a *private neighbor* of v . An irredundant set I is *maximal* if no proper superset of I is an irredundant set. The lower irredundance number $\text{ir}(G)$ equals the minimum cardinality taken over all maximal irredundant sets of G ; similarly, the upper irredundance number $\text{IR}(G)$ equals the maximum cardinality taken over all such sets.

In graph theory, the irredundance numbers have been extensively studied due to their relation to numerous other graph parameters. An estimated 100 research papers [43] have been published on the properties of irredundant sets in graphs, e.g., [2, 13, 27, 42, 41, 45, 90, 12, 108, 25]. For example, if $D \subseteq V$ is an (inclusion-wise) minimal dominating set, then for every $v \in D$ there is some minimality witness, i.e., a vertex that is only dominated by v . In fact, a set is minimal dominating if and only if it is irredundant and dominating [26]. Since each independent set is also an irredundant set, the well-known *domination chain* $\text{ir}(G) \leq \gamma(G) \leq \alpha(G) \leq \text{IR}(G)$ is a simple observation. Here, as usual, $\gamma(G)$ denotes the size of a minimum dominating set, and $\alpha(G)$ denotes the size of a maximum independent set in G . It is known that $\gamma(G)/2 < \text{ir}(G) \leq \gamma(G) \leq 2 \cdot \text{ir}(G) - 1$, see [89].

Determining the irredundance numbers is NP-hard even for bipartite graphs [90]. They can be computed in linear time on graphs of bounded treewidth [7], but the fastest currently known exact algorithm for general graphs is the simple $\mathcal{O}^*(2^n)$ brute-force approach enumerating all subsets.

Since there has been no progress in the exact exponential time area, it is tempting to study these problems from a parameterized complexity viewpoint. The hope is that the additional notion of a *parameter*, e.g., the size k of the irredundant set, allows for a more fine-grained analysis of the running time, maybe even a running time polynomial in n and exponential only in k : It has been known for a while (see, e.g., [137]) that it is possible to break the so-called 2^n -barrier for (some) vertex-selection problems by designing parameterized algorithms that run in time $\mathcal{O}^*(c^k)$ for some $c < 4$ by a “win-

win” approach: either the parameter is “small” ($k < n/2 + \epsilon$ for an appropriate $\epsilon > 0$) and we use the parameterized algorithm, or we enumerate all $\binom{n}{n/2-\epsilon} < 2^n$ subsets.

Unfortunately, the problem of finding an irredundant set of size k is $W[1]$ -complete when parameterized in k as shown by Downey et al. [36], which implies that algorithms with a running time of $\mathcal{O}(f(k)\text{poly}(n))$ are unlikely. However, they also proved that the parameterized dual, where the parameter is $k' := n - k$, admits a problem kernel of size $3k'^2$ and is therefore in \mathcal{FPT} (but the running time has a superexponential dependency on the parameter). What’s more, in order to break the 2^n -barrier for the unparameterized problems, we can also use the dual parameter. Therefore in this paper we study the parameterized problem (following the notation of [36]) CO-MAXIMUM IRREDUNDANT SET (CO-MAXIR):

CO-MAXIMUM IRREDUNDANT SET (CO-MAXIR)

Given: A graph $G(V, E)$, and the parameter k .

We ask: Is there a set $N \subseteq V$ such that $|N| \leq k$ and $V \setminus N$ is an irredundant set..

Our contribution. Our first contribution is a kernel with $3k$ vertices for CO-MAXIR. In particular, this improves the kernel with $3k^2$ vertices and the corresponding running time of $O^*(8^{k^2})$ of [36].

Secondly, we present an algorithm which trades the generality for improved running time and solves CO-MAXIR in time $\mathcal{O}^*(3.069^k)$. Although the algorithm surprisingly simple, a major effort is required to prove the running time using a non-standard measure and a *Measure&Conquer*-approach. While nowadays *Measure&Conquer* is a standard technique for the analysis of moderately exponential time algorithms (see, e.g., [69]), it is still seldom used in parameterized algorithmics.

Finally, as a direct consequence of the above algorithms, we obtain the first exact exponential time algorithm breaking the 2^n -barrier for computing the irredundance numbers on arbitrary graphs with n vertices, a well-known open question (see, e.g., [71]).

7.2. Preliminaries and a Linear Kernel

The following alternative definition of *irredundance* is more descriptive and eases understanding the results in this paper: The vertices in an irredundant set can be thought of as kings, where each such king ought to have his very own private garden that no other king can see (where “seeing” means adjacency). Each king has exactly one cultivated garden, and all the other private neighbors degenerate to wilderness. It is also possible that the garden is already built into the king’s own castle. One can easily verify that this alternate definition is equivalent to the formal one given above.

Definition 7.2.1. Let $G = (V, E)$ be a graph and $I \subseteq V$ an irredundant set. We call the vertices in I *kings*, the set consisting of exactly one private neighbor for each king we call *gardens*, and all remaining vertices *wilderness*. If a king has more than one private neighbor, we fix one of these vertices as a unique garden and the other vertices

as wilderness. If a vertex $v \in I$ has no neighbors in I , we (w.l.o.g.) say v has an *internal* garden, otherwise the garden is *external*. We denote the corresponding sets as $\mathcal{K}, \mathcal{G}, \mathcal{W}$. Note that \mathcal{K} and \mathcal{G} are not necessarily disjoint, since there might be kings with internal gardens. Kings with external gardens are denoted by \mathcal{K}_e and kings with internal garden by \mathcal{K}_i . Similarly, the set of external gardens is $\mathcal{G}_e := \mathcal{G} \setminus \mathcal{K}$. In what follows these sets are also referred to as “labels”.

7.2.1. A Linear Kernel

By using a crown reduction, see [24, 44], we can show:

Theorem 7.2.1: CO-MAXIR admits a kernel with at most $3k$ vertices.

Proof. Let $G = (V, E)$ be a graph and let I be an irredundant set of size at least $n - k$ in a graph G .

We use a crown reduction, see [24, 44]. A crown is a subgraph $G' = (C, H, E') = G[C \cup H]$ of G such that C is an independent set in G , H contains all neighbors of C in G (i.e., H separates C from $V \setminus (H \cup C)$), and such that there is a matching M of size $|H|$ between C and H .

We first show that if G contains a crown (C, H, E') , then G contains a maximum irredundant set I such that $I \supseteq C$ and $H \subseteq V \setminus I$. Assume that this is wrong. So, we have a solution I and let $I = \mathcal{K}_i \cup \mathcal{K}_e$ be an arbitrary partition of I into internal and external kings. Let $\mathcal{G}_e \in V \setminus I$ be an arbitrary set that can serve as a set of gardens for \mathcal{K}_e . Let $\mathcal{W} = V \setminus (I \cup \mathcal{G}_e)$. Let $\mathcal{K}_i^C = \mathcal{K}_i \cap C$. We find partners of \mathcal{K}_i^C in H by the matching M , formally by considering $\mathcal{K}_{iM}^C = \mathcal{K}_i^C \cap V(M)$, to which the matching associates a set $H_i^M \subseteq H$ with $|\mathcal{K}_{iM}^C| = |H_i^M|$.

Let $I^H = H \cap I = H \cap (\mathcal{K}_i \cup \mathcal{K}_e)$. These are the kings in the so-called head H . Let $G^H = H \cap \mathcal{G}_e$. These are the external gardens in the head. The corresponding kings (which are in $N[H]$) are denoted by K_H .

Clearly, $(I^H \cup G^H) \cap H_i^M = \emptyset$, as well as $I^H \cap G^H = \emptyset$. $K(H) := \mathcal{K}_i^C \dot{\cup} I^H \dot{\cup} K_H$ comprise the kings that interfere with $H \cup C$. Interfere means that either a king is a vertex in the head, has its garden (private neighbor) in the head or the internal kings situated in the crown C .

Moreover, $|K(H)| = |\mathcal{K}_i^C \dot{\cup} I^H \dot{\cup} K_H| = |\mathcal{K}_{iM}^C \dot{\cup} I^H \dot{\cup} K_H| + |\mathcal{K}_i^C \setminus \mathcal{K}_{iM}^C| \leq |H_i^M| + |I^H \dot{\cup} K_H| + |\mathcal{K}_i^C \setminus \mathcal{K}_{iM}^C| \leq |H| + |\mathcal{K}_i^C \setminus \mathcal{K}_{iM}^C| \leq |C|$. Observe that every vertex in $v \in \mathcal{K}_{iM}^C$ has its distinct partner in $u \in H$ such that also $u \in \mathcal{W}$. The partner u can be found via the matching M . Now note that $(I \setminus K(H)) \cup C$ gives another irredundant set not smaller than I .

It remains to show that we can always find a large crown in $G = (V, E)$ if $|V| > 3k$.

Let L be a maximal matching in G . We claim that if $|L| > k$, then we can safely answer NO. Assume that I is a maximum irredundant set in G . Let $I = \mathcal{K}_i \cup \mathcal{K}_e$ be an arbitrary partition of I into internal and external kings. Let $\mathcal{G}_e \in V \setminus I$ be an arbitrary set that can serve as a set of gardens for \mathcal{K}_e . Let $\mathcal{W} = V \setminus (I \cup \mathcal{G}_e)$. In general, $\mathcal{W} \cap V(L)$

causes no trouble for the following counting argument. More formally, let us assign a weight $w(x) = 1$ to such wilderness vertices $x \in \mathcal{W}$.

If $x \in \mathcal{K}_i$, then let $w(x) := 0$. Observe that the vertex y matched to x by L lies in \mathcal{W} . So, consider $x \in (\mathcal{K}_e \cup \mathcal{G}_e)$. Let us assign a weight of $\frac{1}{2}$ to each such vertex. Notice that $|V| - |I| = \sum_{x \in V} w(x)$. Moreover, $|L| \leq \sum_{x \in V(L)} w(x)$ according to the fact for every $\{u, v\} \in L$ we have $w(u) + w(v) \geq 1$. Hence, if $|L| > k$, then $|V| - |I| > k$, so that we can answer **NO** as claimed.

Hence, L contains at most k edges if G contains an irredundant set of size at least $n - k$. This reasoning also holds for maximal matchings that contain no L -augmenting path of length three, a technical notion introduced in [24]. The demonstration given in [24, Theorem 3] shows the claimed kernel bound. \square

The above theorem already shows that CO-MAXIR allows fixed-parameter tractable algorithms with a running time exponential in k , namely $\mathcal{O}^*(8^k)$, which is a new contribution.

7.2.2. Basic Facts

Our algorithm for the irredundance number recursively branches on the vertices of the graph and assigns each vertex one of the four possible labels $\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W}$, until a labeling that forms a solution has been found (if one exists). If I is an irredundant set of size at least $n - k$, then it is easy to see that $|\mathcal{G} \setminus \mathcal{K}| + |\mathcal{W}| \leq k$ and $|\mathcal{K}_e| = |\mathcal{K} \setminus \mathcal{G}| \leq k$, which indicates a first termination condition. Furthermore, one can easily observe that for any irredundant set $I \subseteq V$ the following simple properties hold for all $v \in V$: (1) if $|N(v) \cap \mathcal{K}| \geq 2$ then $v \in \mathcal{K} \cup \mathcal{W}$; (2) if $|N(v) \cap \mathcal{G}| \geq 2$ then $v \in \mathcal{G} \cup \mathcal{W}$; (3) if $|N(v) \cap \mathcal{K}| \geq 2$ and $|N(v) \cap \mathcal{G}| \geq 2$ then $v \in \mathcal{W}$. Additionally, for all $v \in \mathcal{K}_i$, we have $N(v) \subseteq \mathcal{W}$.

This gives us a couple of conditions the labeling has to satisfy in order to yield an irredundant set: each external garden is connected to exactly one external king and vice versa. Once the algorithm constructs a labeling that cannot yield an irredundant set anymore the current branch can be terminated.

Definition 7.2.2. Let $G = (V, E)$ be a graph and let $\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W} \subseteq V$ be a labeling of V . Let $\bar{V} = V \setminus (\mathcal{K}_i \cup \mathcal{K}_e \cup \mathcal{G}_e \cup \mathcal{W})$. We call $(\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W})$ *valid* if the following conditions hold, and *invalid* otherwise.

1. $\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W}$ are pairwise disjoint,
2. for each $v \in \mathcal{K}_i$, $N(v) \subseteq \mathcal{W}$,
3. for each $v \in \mathcal{K}_e$, $N(v) \cap (\mathcal{G}_e \cup \bar{V}) \neq \emptyset$,
4. for each $v \in \mathcal{K}_e$, $|N(v) \cap \mathcal{G}_e| \leq 1$,
5. for each $v \in \mathcal{G}_e$, $N(v) \cap (\mathcal{K}_e \cup \bar{V}) \neq \emptyset$, and
6. for each $v \in \mathcal{G}_e$, $|N(v) \cap \mathcal{K}_e| \leq 1$.

As a direct consequence, we can define a set of vertices that can no longer become external gardens or kings without invalidating the current labeling:

$$\begin{aligned} \text{Not}\mathcal{G} &:= \{v \in \overline{V} \mid \text{the labeling } (\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e \cup \{v\}, \mathcal{W}) \text{ is invalid}\} \\ \text{Not}\mathcal{K} &:= \{v \in \overline{V} \mid \text{the labeling } (\mathcal{K}_i, \mathcal{K}_e \cup \{v\}, \mathcal{G}_e, \mathcal{W}) \text{ is invalid}\} \end{aligned}$$

It is easy to see that $\text{Not}\mathcal{K}$ and $\text{Not}\mathcal{G}$ can be computed in polynomial time, and since vertices in $\text{Not}\mathcal{G} \cap \text{Not}\mathcal{K}$ can only be wilderness, we can also assume that $\text{Not}\mathcal{G} \cap \text{Not}\mathcal{K} = \emptyset$.

7.3. Measure & Conquer Tailored To The Problem

We use a more precise annotation of vertices: In the course of the algorithm, they will be either unlabeled \mathcal{U} , kings with internal gardens \mathcal{K}_i , kings with external gardens \mathcal{K}_e , (external) gardens \mathcal{G}_e , wilderness \mathcal{W} , not being kings $\text{Not}\mathcal{K}$, or not being gardens $\text{Not}\mathcal{G}$. We furthermore partition the set of vertices V into *active* vertices

$$V_a = \mathcal{U} \cup \text{Not}\mathcal{G} \cup \text{Not}\mathcal{K} \cup \{v \in \mathcal{K}_e \mid N(v) \cap \mathcal{G}_e = \emptyset\} \cup \{v \in \mathcal{G}_e \mid N(v) \cap \mathcal{K}_e = \emptyset\}$$

that have to be reconsidered, and *inactive* vertices $V_i = V \setminus V_a$. This means that the inactive vertices are either from \mathcal{W} , \mathcal{K}_i or paired-up external kings and gardens. Define $\mathcal{K}_{ea} := \mathcal{K}_e \cap V_a$ and $\mathcal{K}_{ei} := \mathcal{K}_e \cap V_i$ (and analogously $\mathcal{G}_{ea} := \mathcal{G}_e \cap V_a$ and $\mathcal{G}_{ei} := \mathcal{G}_e \cap V_i$).

We use the next non-standard measure

$$\varphi(k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a) = k - |\mathcal{W}| - |\mathcal{G}_{ei}| - \omega_\ell(|\mathcal{K}_{ea}| + |\mathcal{G}_{ea}|) - \omega_n(|\text{Not}\mathcal{G}| + |\text{Not}\mathcal{K}|),$$

where $\text{Not}\mathcal{G}$ and $\text{Not}\mathcal{K}$ are taken into account. We will later determine the weights ω_ℓ and ω_n to optimize the analysis, where $0 \leq \omega_n \leq 0.5 \leq \omega_\ell \leq 1$ and $\omega_n + \omega_\ell \leq 1$. If the measure changes we will describe in words how the measure changes in each case.

Although we are looking for a maximal irredundant set, we can likewise look for a *complete labeling* $L = (\mathcal{K}_i^L, \mathcal{G}_{ei}^L, \mathcal{K}_{ei}^L, \mathcal{W}^L)$ that partitions the whole vertex set $V = \mathcal{K}_i^L \uplus \mathcal{G}_{ei}^L \uplus \mathcal{K}_{ei}^L \uplus \mathcal{W}^L$ into internal kings, external kings and gardens, as well as wilderness. Having determined L , $I_L = \mathcal{K}_{ei}^L \uplus \mathcal{K}_i^L$ should be an irredundant set, and conversely, to a given irredundant set I , one can compute in polynomial time a corresponding complete labeling.

7.3.1. Reduction Rules

Let us first present the reduction rules that we employ in Table 7.1. The lower the number the higher the priority of the rule, i.e. lower numbered rules are applied before higher numbered ones.

Lemma 7.3.1: The rules listed in Table 7.1 are sound and do not increase the measure.

1. If V contains a vertex $x \in \mathcal{K}_i$ and a neighbor $u \in (\mathcal{G}_e \cup \mathcal{K}_e \cup \mathcal{K}_i) \cap N(x)$, then return NO.
2. a) If V contains a vertex x with two neighbors u, v where $x \in \mathcal{K}_e$ and $u, v \in \mathcal{G}_e$, then return NO.
b) If V contains a vertex x with two neighbors u, v where $x \in \mathcal{G}_e$ and $u, v \in \mathcal{K}_e$, then return NO.
3. If V contains an isolated vertex $v \in (\mathcal{G}_e \cup \mathcal{K}_e)$, then return NO.
4. If V contains an isolated vertex $v \in (\text{Not}\mathcal{K} \cup \text{Not}\mathcal{G})$, then put v into \mathcal{W} , decreasing the measure by $1 - \omega_n$.
5. If V contains an isolated vertex $u \in \mathcal{U}$, then put u into \mathcal{K}_i and set $V_a = V_a \setminus \{u\}$.
6. Delete an edge between two external kings or two external gardens.
7. a.) Delete an edge between a \mathcal{K}_e - and a $\text{Not}\mathcal{G}$ -vertex.
b.) Delete an edge between a \mathcal{G}_e - and a $\text{Not}\mathcal{K}$ -vertex.
8. Remove any edges incident to vertices in \mathcal{W} .
9. a) Delete an edge between two $\text{Not}\mathcal{K}$ -vertices.
b) Delete an edge between two $\text{Not}\mathcal{G}$ -vertices.
10. If $u \in \mathcal{U}$ such that $N(u) = \{v\}$ for some $v \in \mathcal{U}$, then put u into \mathcal{K}_i and set $V_a = V_a \setminus \{u\}$.
11. If $u \in \mathcal{K}_i$, then put its neighbors $N(u)$ into \mathcal{W} and set $V_a = V_a \setminus N(u)$; this decreases the measure by $|N(u)|$.
12. a) If V contains two neighbors u, v such that $u \in \mathcal{G}_{ea}$ and $v \in \mathcal{U} \cup \text{Not}\mathcal{G}$ with either $d(u) = 1$ or $d(v) = 1$, then put v into \mathcal{K}_e , and make u, v inactive;
b) If V contains two neighbors u, v such that $u \in \mathcal{K}_{ea}$ and $v \in \mathcal{U} \cup \text{Not}\mathcal{K}$ with either $d(u) = 1$ or $d(v) = 1$, then put v into \mathcal{G}_e , and make u, v inactive; this decreases the measure by $1 - \omega_\ell$ if $v \in \mathcal{U}$ and otherwise by $1 - \omega_\ell - \omega_n$.
13. a) If V contains a vertex v with two neighboring gardens such that $v \in \mathcal{U}$ (i.e., $|N(v) \cap \mathcal{G}_e| \geq 2$), then set $v \in \text{Not}\mathcal{K}$; if $v \in \text{Not}\mathcal{G}$, then set $v \in \mathcal{W}$.
b) If V contains a vertex v with two neighboring kings such that $v \in \mathcal{U}$ (i.e., $|N(v) \cap \mathcal{K}_e| \geq 2$), then set $v \in \text{Not}\mathcal{G}$; if $v \in \text{Not}\mathcal{K}$, then set $v \in \mathcal{W}$. This decreases the measure by ω_n or $(1 - \omega_n)$, respectively.
14. Assume that V contains two inactive neighbors u, v where $u \in \mathcal{K}_e$ and $v \in \mathcal{G}_e$, then put all $x \in (N(u) \cap \mathcal{U})$ into $\text{Not}\mathcal{G}$, all $x \in (N(u) \cap \text{Not}\mathcal{K})$ into \mathcal{W} , all $x \in (N(v) \cap \mathcal{U})$ into $\text{Not}\mathcal{K}$ and all $x \in (N(v) \cap \text{Not}\mathcal{G})$ into \mathcal{W} . The measure decrease for each vertex x is ω_n if $x \in ((N(u) \cup N(v)) \cap \mathcal{U})$ and $1 - \omega_n$ if $x \in (N(u) \cap \text{Not}\mathcal{K}) \cup (N(v) \cap \text{Not}\mathcal{K})$.

Table 7.1.: Extensive list of reduction rules.

Proof. **(1)/(2)** Otherwise the labeling is not valid, i.e, it violates items 2, 4 or 6 of Definition 7.2.2.

(3) An isolated vertex $v \in (\mathcal{G}_e \cup \mathcal{K}_e)$ cannot be paired with a second vertex, i.e., items 3 and 5 of Definition 7.2.2 are violated.

(4) As an isolated vertex $v \in (\text{Not}\mathcal{K} \cup \text{Not}\mathcal{G})$ cannot be paired it must be wilderness as otherwise items 3 and 5 of Definition 7.2.2 are violated.

(5) An isolated vertex $u \in \mathcal{U}$ can be made an internal king such that any labeling remains valid (Def. 7.2.2).

(6)-(9) Consider a complete labeling L for the graph G' obtained after the reduction rule has been applied. Adding the edge which was deleted by one of the rules keeps L valid according to Def. 7.2.2.

(10) Consider a complete valid labeling L extending the current one (i.e. $\bar{V} = \emptyset$) such that $|\mathcal{W}| + |\mathcal{G}_e|$ is minimum. Note that if $v \in \mathcal{W}$ we have that u in \mathcal{K}_i as otherwise we can decrease the number of wilderness and garden vertices. If $v \in \mathcal{K}_i$ then we have $u \in \mathcal{W}$. Then by setting $v \in \mathcal{W}$ and $u \in \mathcal{K}_i$ we obtain a solution of equal size. Thus, $v \in \mathcal{G}_e \cup \mathcal{K}_e$. If $v \in \mathcal{K}_e$, w.l.o.g, we can assume that $u \in \mathcal{G}_e$ is the paired garden vertex. Then making v a wilderness vertex and u an internal king yields an equivalent labeling. If $v \in \mathcal{G}_e$ the argumentation is analogously.

(11) To keep the labeling valid according item 2 of Def. 7.2.2 this has to be done.

(12) *a)* If $d(u) = 1$ then the only possibility not to violate item 5 of Def. 7.2.2 in a complete labeling is to put $v \in \mathcal{K}_i$. If $d(v) = 1$ then, w.l.o.g., we can put v in \mathcal{K}_e and pair it with u as the labeling remains valid. *Case b)* follows analogously.

(13) *a)* If $u \in \mathcal{U}$ then putting u into \mathcal{K} violates item 4 of Def. 7.2.2. If $u \in \text{Not}\mathcal{G}$ then as u also cannot be a king we can label u as wilderness. *Case b)* follows analogously.

(14) By items 4 and 6 of Def. 7.2.2 no vertex in $(N(v) \cap \mathcal{U})$ can be a king and no vertex in $(N(u) \cap \mathcal{U})$ a garden. The same items justify to put all $x \in (N(u) \cap \text{Not}\mathcal{K}) \cup (N(v) \cap \text{Not}\mathcal{G})$ into \mathcal{W} .

As $0 \leq \omega_n \leq 0.5 \leq \omega_\ell \leq 1$ and $\omega_n + \omega_\ell \leq 1$ no reduction rule increases the measure ρ . □

Lemma 7.3.2: In a reduced instance, a vertex $v \in \text{Not}\mathcal{K} \cup \text{Not}\mathcal{G}$ may have at most one neighbor $u \in \mathcal{G}_e \cup \mathcal{K}_e$; more precisely, if such u exists, then $u \in \mathcal{G}_e$ if and only if $v \in \text{Not}\mathcal{G}$ and $d(v) \geq 2$ (Thus, v must have a neighbor z that is not in $\mathcal{G}_e \cup \mathcal{K}_e$).

Proof. Consider, w.l.o.g., $v \in \text{Not}\mathcal{G}$.

Assume that $u \in N(v) \cap (\mathcal{G}_e \cup \mathcal{K}_e)$ exists. Note that the alternative $u \in \mathcal{K}_e$ is resolved by Reduction Rule 7. Hence, $u \in \mathcal{G}_e$. If v had no other neighbor but u , then Reduction Rule 12 would have triggered. So, $d(v) \geq 2$. Let $z \in N(v) \setminus \{u\}$. If the claim were false, then $z \in \mathcal{G}_e \cup \mathcal{K}_e$. The case $z \in \mathcal{K}_e$ is ruled out by Reduction Rule 7. The case $z \in \mathcal{G}_e$ is dealt with by Reduction Rule 13. Hence, $z \notin \mathcal{G}_e \cup \mathcal{K}_e$. \square

7.3.2. The Algorithm

Now consider Algorithm 11.

7.3.2.1. Correctness of the Algorithm

Lemma 7.3.3: In each step of a recursive call of CO-IR in Algorithm 11 there are no two neighbors u, v such that $u \in \mathcal{K}_{e_a}$ and $v \in \mathcal{G}_{e_a}$ in the given labeled graph.

Proof. Clearly the statement holds for the initial input graph. The only reduction rule which could create such a situation is Rule 12. But here the two vertices u, v will be immediately inactivated. In each line where recursive calls are made in Algorithm 11 it is easily checked if such a situation as described could occur. If so, the affected pair u, v is removed from V_a in every case. \square

Lemma 7.3.4: Whenever our algorithm encounters a reduced instance, a vertex $v \in \mathcal{G}_e$ obeys $N(v) \subseteq \mathcal{U} \cup \text{Not}\mathcal{G}$. Symmetrically, if $v \in \mathcal{K}_e$, then $N(v) \subseteq \mathcal{U} \cup \text{Not}\mathcal{K}$.

Proof. Consider $z \in N(v)$, where $v \in \mathcal{G}_e$. $N(v) \not\subseteq \mathcal{U} \cup \text{Not}\mathcal{G}$ is ruled out by reduction rules and the previous lemma: (a) $z \in \mathcal{G}_e$ violates Rule 6.

(b) $z \in \mathcal{K}_e$ violates the invariant shown in Lemma 7.3.3. (c) $z \in \text{Not}\mathcal{K}$ violates Rule 7. (d) $z \in \mathcal{W}$ violates Rule 8. The case where $v \in \mathcal{K}_e$ follows analogously. \square

Note that the irredundance numbers can be computed in polynomial time on graphs of bounded treewidth, see [149, Page 75f.], and the corresponding dynamic programming easily extends also to labeled graphs, since the labels basically correspond to the states of the dynamic programming process.

Comments on the Algorithm and the Labeling Since $(\text{Not}\mathcal{K} \cup \text{Not}\mathcal{G} \cup \mathcal{U}) \subseteq V_a$, we have obtained a complete labeling once we leave our algorithm in Line 4, returning YES. During the course of the algorithm, we deal with (incomplete) labelings $L = (\mathcal{K}_i, \mathcal{G}_e, \mathcal{K}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a)$, a tuple of subsets of V that also serve as input to our algorithm, preserving the invariant that $V = \mathcal{K}_i \uplus \mathcal{G}_e \uplus \mathcal{K}_e \uplus \text{Not}\mathcal{G} \uplus \text{Not}\mathcal{K} \uplus \mathcal{W} \uplus \mathcal{U}$. A complete labeling corresponds to a labeling with $\text{Not}\mathcal{G} = \text{Not}\mathcal{K} = \mathcal{U} = V_a = \emptyset$. We say that a labeling $L' = (\mathcal{K}_i', \mathcal{G}_e', \mathcal{K}_e', \text{Not}\mathcal{G}', \text{Not}\mathcal{K}', \mathcal{W}', V_a')$ *extends* the labeling $L = (\mathcal{K}_i, \mathcal{G}_e, \mathcal{K}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a)$ if $\mathcal{K}_i \subseteq \mathcal{K}_i', \mathcal{G}_e \subseteq \mathcal{G}_e', \mathcal{K}_e \subseteq \mathcal{K}_e', \text{Not}\mathcal{G} \subseteq \mathcal{W}' \cup \mathcal{K}_e', \text{Not}\mathcal{K} \subseteq \mathcal{W}' \cup \mathcal{G}_e', \mathcal{W} \subseteq \mathcal{W}', V_a' \subseteq V_a$. We also write $L \prec_G L'$ if L' extends L . We can also speak of a complete labeling extending a labeling in the sense described above. Notice that reduction rules and recursive calls only extend labelings (further).

Algorithm 11 An algorithm for CO-MAXIR.

Algorithm CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a$):Input: Graph $G = (V, E)$, $k \in \mathbf{N}$, labels $\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \subseteq V$ Output: YES if a co-irredundant set $CI \subseteq V$ such that $|CI| \leq k$ exists.

- 01: Consecutively apply the procedure CO-IR to components containing V_a -vertices.
- 02: Apply all the reduction rules exhaustively.
- 03: **if** $\varphi(k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, V_a) < 0$ **then return** NO.
- 04: **if** $V_a = \emptyset$ **then return** YES.
- 05: **if** $\max\text{degree}(G[V_a]) \leq 2$ **then** solve the remaining instance by dynamic programming.
- 06: **if** $\text{Not}\mathcal{G} \neq \emptyset$ **then**
- 07: choose $v \in \text{Not}\mathcal{G}$; **if** $\exists z \in N(v) \cap \mathcal{G}_e$ **then** $I := \{v, z\}$ **else** $I := \emptyset$.
- 08: **return** CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e \cup \{v\}, \mathcal{G}_e, \text{Not}\mathcal{G} \setminus \{v\}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus I$) **or**
CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G} \setminus \{v\}, \text{Not}\mathcal{K}, \mathcal{W} \cup \{v\}, V_a \setminus \{v\}$);
- 09: **if** $\text{Not}\mathcal{K} \neq \emptyset$ **then**
- 10: choose $v \in \text{Not}\mathcal{K}$; **if** $\exists z \in N(v) \cap \mathcal{K}_e$ **then** $I := \{v, z\}$ **else** $I := \emptyset$.
- 11: **return** CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e \cup \{v\}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K} \setminus \{v\}, \mathcal{W}, V_a \setminus I$) **or**
CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K} \setminus \{v\}, \mathcal{W} \cup \{v\}, V_a \setminus \{v\}$);
- 12: **if** there is an unlabeled $v \in V$ with exactly two neighbors u, w in $G[V_a]$, where $u \in \mathcal{G}_{ea}$ and $w \in \mathcal{K}_{ea}$ **then**
- 13: **return** CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e \cup \{v\}, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{v, u\}$) **or**
CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e \cup \{v\}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{v, w\}$);
- 14: **if** $\mathcal{K}_{ea} \cup \mathcal{G}_{ea} \neq \emptyset$ **then**
- 15: Choose some $v \in \mathcal{K}_{ea} \cup \mathcal{G}_{ea}$ of maximum degree.
- 16: **if** $v \in \mathcal{K}_{ea}$ **then**
- 17: **return**
 $\exists u \in N(v) : \text{CO-IR}(G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e \cup \{u\}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{u, v\})$
- 18: **else then**
- 19: **return**
 $\exists u \in N(v) : \text{CO-IR}(G, k, \mathcal{K}_i, \mathcal{K}_e \cup \{u\}, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{u, v\})$
- 20: **else** Choose $v \in \mathcal{U}$ of maximum degree, preferring v with some $u \in N(v)$ of degree two.
- 21: **return** CO-IR($G, k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W} \cup \{v\}, V_a \setminus \{v\}$)
or CO-IR($G, k, \mathcal{K}_i \cup \{v\}, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{v\}$)
or $\exists u \in N(v) : \text{CO-IR}(G, k, \mathcal{K}_i, \mathcal{K}_e \cup \{v\}, \mathcal{G}_e \cup \{u\}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{u, v\})$
or $\exists u \in N(v) : \text{CO-IR}(G, k, \mathcal{K}_i, \mathcal{K}_e \cup \{u\}, \mathcal{G}_e \cup \{v\}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a \setminus \{u, v\})$

Notice that \prec_G is a partial order on the set of labelings of a graph $G = (V, E)$. The maximal elements in this order are precisely the complete labelings. Hence, the complete labeling L_I corresponding to a maximal irredundant set I is maximal, with $\varphi(k, L_I) \geq 0$ iff $|I| \geq |V| - k$. Conversely, given a graph $G = (V, E)$, the labeling $L_G = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, V)$ is the smallest element of \prec_G ; this is also the initial labeling that we start off with when first calling Algorithm 11. If L, L' are labelings corresponding to the parameter lists of nodes n, n' in the search tree such that n is ancestor of n' in the search tree, then $L \prec_G L'$. The basic strategy of Algorithm 11 is to exhaustively consider all complete labelings (only neglecting cases that cannot be optimal). This way, also all important maximal irredundant sets are considered.

Correctness

Lemma 7.3.5: If $\varphi(k, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, V_a) < 0$, then for weights $0 \leq \omega_n \leq 0.5 \leq \omega_\ell \leq 1$ with $\omega_n + \omega_\ell \leq 1$ and for any complete labeling $L = (\mathcal{K}_i^L, \mathcal{G}_{e_i}^L, \mathcal{K}_{e_i}^L, \mathcal{W}^L)$ extending the labeling $\Lambda := (\mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, \mathcal{W}, V_a)$ we have $\varphi(k, \mathcal{K}_i^L, \mathcal{G}_{e_i}^L, \mathcal{K}_{e_i}^L, \emptyset, \emptyset, \mathcal{W}^L, \emptyset) < 0$.

Proof.

1	$\mathcal{K}_{ea} \rightarrow \mathcal{K}_{e_i}^L$	$-\omega_\ell$
2	$\mathcal{G}_{ea} \rightarrow \mathcal{G}_{e_i}^L$	$1 - \omega_\ell$
3	$\text{Not}\mathcal{G} \rightarrow \mathcal{K}_{e_i}^L$	$-\omega_n$
4	$\text{Not}\mathcal{G} \rightarrow \mathcal{W}^L$	$1 - \omega_n$
5	$\text{Not}\mathcal{K} \rightarrow \mathcal{G}_{e_i}^L$	$1 - \omega_n$
6	$\text{Not}\mathcal{K} \rightarrow \mathcal{W}^L$	$1 - \omega_n$
7	$\mathcal{U} \rightarrow \mathcal{W}^L$	1
8	$\mathcal{U} \rightarrow \mathcal{K}_i^L$	0
9	$\mathcal{U} \rightarrow \mathcal{K}_{e_i}^L$	0
10	$\mathcal{U} \rightarrow \mathcal{G}_{e_i}^L$	1

We give a table for every possible label transition from Λ to its extension L . Note that Algorithm CO-IR only computes such solutions. All entries except two cause a non-increase of φ . The entries number 1 and 3 expose an increase in φ . By the problem definition, there exists a bijection $f : \mathcal{K}_e^L \rightarrow \mathcal{G}_e^L$. So for a vertex v in $\mathcal{K}_{e_i}^L \cap \mathcal{K}_{ea}$ we must have $f(v) \in \mathcal{U} \cup \text{Not}\mathcal{K}$ by Lemma 7.3.4. Taking now into account the label transition of $f(v)$ which must be of the form $\mathcal{U} \rightarrow \mathcal{G}_{e_i}^L$ or $\text{Not}\mathcal{K} \rightarrow \mathcal{G}_{e_i}^L$, we see that a total decrease with respect to v and $f(v)$ of at least $1 - \omega_n - \omega_\ell \geq 0$ can be claimed. If $v \in \text{Not}\mathcal{G} \cap \mathcal{K}_{e_i}^L$ then by arguing analogously we get a total decrease of at least $1 - \omega_\ell - \omega_n \geq 0$ as by the reduction rules 7, 8 and 9 we have $f(v) \subseteq \mathcal{U} \cup \text{Not}\mathcal{K} \cup \mathcal{G}_e$. \square

Lemma 7.3.6: Assume that there is an unlabeled vertex v , which has exactly two neighbors $v_G \in \mathcal{G}_e$ and $v_K \in \mathcal{K}_e$ in $G[V_a]$. In the corresponding branching process, we may then omit the case $v \in \mathcal{W}$.

Proof. We are looking for an inclusion-maximal irredundant set. Hence, only the positions of the kings matter, not the positions of the gardens. So, in particular we cannot insist on the garden of v_K being placed on some neighbor u of v_K different from v . In this sense, any solution that uses v as wilderness can be transformed into a no worse solution with $v \in \mathcal{G}_e$: Simply pair up v and v_K , turning the hitherto garden of v_K into wilderness. So, no optimum solution is lost by omitting the case $v \in \mathcal{W}$ in the branching. \square

Lemma 7.3.7: Algorithm 11 correctly solves CO-MAXIR.

Proof. The algorithm correctly answers NO in line 3 by Lemma 7.3.5. If $V_a = \emptyset$ in line 4 we can deduce $k \geq |\mathcal{W}| + |\mathcal{G}_{e_i}|$ from the definition of the measure ρ . Thus, we can correctly answer YES. The recursive calls in lines 8, 11, 17, 19 and 21 are all exhaustive branchings and therefore no optimum solution is neglected. In the call in line 13 we do not consider the possibility of setting $v \in \mathcal{W}$ which is justified by Lemma 7.3.6. \square

7.3.2.2. Run Time Analysis

Theorem 7.3.8: CO-MAXIR can be solved in time $\mathcal{O}^*(3.069^k)$.

Proof. The correctness of the algorithm has been reasoned above already. In particular, notice Lemma 7.3.5 concerning the correctness of the abort.

For the running time, we now provide a partial analysis leading to recurrences that estimate an upper bound on the search tree size $T(\varphi, h)$, where φ denotes the measure and h the height of the search tree. The claimed running time would then formally follow by an induction over h .

1. Assume that the algorithm branches on some vertex $v \in \text{Not}\mathcal{G}$, the case $v \in \text{Not}\mathcal{K}$ being completely analogous. By reduction rules, $N(v) \subseteq \mathcal{U} \cup \mathcal{G}_{e_a} \cup \text{Not}\mathcal{K}$.
 - a) If $N(v) \subseteq \mathcal{U} \cup \text{Not}\mathcal{K}$, we derive the following branch in the worst case:
 $T(\varphi, h) \leq T(\varphi - (1 - \omega_n), h - 1) + T(\varphi - (\omega_\ell - \omega_n), h - 1)$. This follows from a simple branching analysis considering the cases that v becomes wilderness or that v becomes a king. See also Table 7.2 where the entries (1a)#1 and (1a)#2 correspond to the reduction of the first and second branch.
 - b) Assume now that $N(v) \cap (\mathcal{G}_{e_a}) \neq \emptyset$ and let $u \in N(v) \cap \mathcal{G}_{e_a}$. Lemma 7.3.2 ensures that there can be at most one element in $N(v) \cap \mathcal{G}_e$ and that $d(v) \geq 2$. Due to Reduction Rule 12, $d(u) \geq 2$.
 - i. First assume that $d(u) = 2$, i.e., $N(u) = \{v, x\}$. Then, we arrive at the following recursion: $T(\varphi, h) \leq T(\varphi - (2 - \omega_\ell - \omega_n), h - 1) + T(\varphi - (1 - \omega_\ell + \omega_n), h - 1)$, see entries (1bi.#1) and (1bi.#2) of Table 7.2. This is seen as follows. By setting $v \in \mathcal{W}$, due to Reduction Rule 8, u will be of degree one and hence will be paired with its neighbor x due to Reduction Rule 12. If $x \in \mathcal{U}$, the measure decreases by $2 - \omega_\ell - \omega_n$. If $x \in \text{Not}\mathcal{G}$, it decreases by $2 - \omega_\ell - 2\omega_n$. But then by Lemma 7.3.2 there is $y \in N(x) \setminus \{u\}$ such that $y \in \text{Not}\mathcal{K} \cup \mathcal{U}$. Then by Reduction Rule 12 y is moved to $\mathcal{W} \cup \text{Not}\mathcal{G}$ giving some additional amount of at least ω_n . Note that $y \neq v$ as $v \in \text{Not}\mathcal{G}$.
 If we set $v \in \mathcal{K}_e$, then u and v will be paired by Reduction Rule 14 and decrease φ by an amount $1 - \omega_\ell - \omega_n$. Thereafter, the other neighbor x of u will become a member of $\text{Not}\mathcal{K}$ or of \mathcal{W} due to rule 14., depending on its previous status. Moreover, there must be a further neighbor $z \in \mathcal{U} \cup \text{Not}\mathcal{K}$

Weight	1	0	0	1	ω_ℓ	ω_ℓ	ω_n	ω_n	0	---
Case	\mathcal{W}	\mathcal{K}_i	(\mathcal{K}_{ei})	\mathcal{G}_{ei}	\mathcal{K}_{ea}	\mathcal{G}_{ea}	Not \mathcal{G}	Not \mathcal{K}	(\mathcal{U})	measure diff. \geq
(1a)#1	$+v$						$-v$			$1 - \omega_n$
(1a)#2					$+v$		$-v$			$\omega_\ell - \omega_n$
(1bi)#1	$+v$		$+x$	$+u$		$-u$	$-v$		$-x$	$2 - \omega_n - \omega_\ell$
(1bi)#2			$\{v, x\}$	$+u$		$-u$	$-v, +z$	$+x$	$-x, -z$	$1 + \omega_n - \omega_\ell$
(1bii)#1	$+v$						$-v$			$1 - \omega_n$
(1bii)#2			$+v$	$+u$		$-u$	$-v, +z$	$\{x_1, x_2\}$	$\{x_1, x_2, z\}$	$1 + 2\omega_n - \omega_\ell$
(2a)#1			$+v$	$+v_G$		$-v_G$	$+N(v_G) \setminus \mathcal{G}_e$		$-v$	$1 - \omega_\ell + 2 \cdot \omega_n$
(2a)#2			$+v_K$	$+v$	$-v_K$			$+N(v_K) \setminus \mathcal{K}_e$	$-v$	$1 - \omega_\ell + 2 \cdot \omega_n$
(2b)#1			$\{v, v_K\}$	$\{v_G, x\}$	$-v_K$	$-v_G$	$+N(v_G) \setminus \{v\}$	$-x (x \in N(v_K) \setminus \{v\})$	$-v$	$2 - 2\omega_\ell + \omega_n$
(2b)#2			$\{v_K, x\}$	$\{v, v_G\}$	$-v_K$	$-v_G$	$-x \in N(v_G) \setminus \{v\}$	$+N(v_K) \setminus \{v\}$	$-v$	$2 - 2\omega_\ell + \omega_n$
(2c)#1			$\{v, v_K\}$	$\{v_G, x\}$	$-v_K$	$-v_G$	$+N(v_G) \setminus \{v\}$	$-x (x \in N(v_K) \setminus \{v\})$	$-v$	$2 - 2\omega_\ell + \omega_n$
(2c)#2			$+v_K$	$+v$	$-v_K$			$+N(v_K) \setminus \mathcal{K}_e$	$-v$	$1 - \omega_\ell + \omega_n$
(3)#j			$+u$	$+v$		$-v$	$+N(u) \setminus \{v\}$	$+N(v) \setminus \{u\}$	$-u$	$1 - \omega_\ell + d(v) \cdot \omega_n$
(4a)#1	$+N(u) \cup \{v\}$	$+u$							$-N(u) \cup \{v\}$	2
(4a)#2	$+N(v)$	$+v$							$-N[v]$	$d(v)$
(4a)#j			$+v$	$+u$			$+N(v) \setminus \{u\}$	$+N(u) \setminus \{v\}$	$-N[v] \cup N(u)$	$1 + d(v) \cdot \omega_n$
(4a)#j			$+u$	$+v$			$+N(u) \setminus \{v\}$	$+N(v) \setminus \{u\}$	$-N[v] \cup N(u)$	$1 + d(v) \cdot \omega_n$
(4b)#1	$+N(u) \cup \{v\}$			$+u$					$-N(u) \cup \{v\}$	1
(4b)#2	$+N(v)$	$+v$							$-N[v]$	$d(v)$
(4b)#j			$+v$	$+u$			$+N(v) \setminus \{u\}$	$+N(u) \setminus \{v\}$	$-N[v] \cup N(u)$	$1 + (d(v) + 1) \cdot \omega_n$
(4b)#j			$+u$	$+v$			$+N(u) \setminus \{v\}$	$+N(v) \setminus \{u\}$	$-N[v] \cup N(u)$	$1 + (d(v) + 1) \cdot \omega_n$

Table 7.2.: Overview over different branchings; symmetric branches due to exchanging roles of kings and gardens are not displayed. Neither are possibly better branches listed.

of v (by Lemma 7.3.2 and the fact that u is the unique \mathcal{G}_{ea} neighbor) that will become member of $\text{Not}\mathcal{G}$ or \mathcal{W} due to rule 14. This yields the claimed measure change if $z \neq x$ as we get an additional decrease of φ of at least $2\omega_n$. If $z = x$, then z is in \mathcal{U} and the vertex will be put into \mathcal{W} resulting in a decrease of φ of one. Thus, the recursively considered instance has complexity $T(\varphi - (2 - \omega_\ell - \omega_n), h - 1) \leq T(\varphi - (1 - \omega_\ell + \omega_n), h - 1)$.

- ii. Secondly, assume that $d(u) \geq 3$ (keeping the previous scenario otherwise). This yields the following worst-case branch: $T(\varphi, h) \leq T(\varphi - (1 - \omega_n), h - 1) + T(\varphi - (1 - \omega_\ell + 2\omega_n), h - 1)$, see entries (1bii.#1) and (1bii.#2) of Table 7.2, where $x_1, x_2 \in N(u)$ and $z \in N(u)$.

This is seen by a similar (even simpler) analysis as all vertices in $N(v) \cup N(u)$ get relabeled by rule 14. Note that all $z \in N(v) \cap N(u) \subseteq \mathcal{U}$ get labeled \mathcal{W} in the second branch.

We will henceforth not present the recurrences for the search tree size in this explicit form, but rather point to Table 7.2 that contains the same information. There, cases are differentiated by writing B_j for the j th branch.

2. Assume that all active vertices are in $\mathcal{U} \cup \mathcal{G}_e \cup \mathcal{K}_e$, with $\mathcal{G}_{ea} \cup \mathcal{K}_{ea} \neq \emptyset$. Then, the algorithm would pair up some $v \in \mathcal{G}_{ea} \cup \mathcal{K}_{ea}$. First note that by Lemma 7.3.4 and the fact that $\text{Not}\mathcal{G} \cup \text{Not}\mathcal{K} = \emptyset$ we have $N(v) \subseteq \mathcal{U}$.

Assume that there is an unlabeled vertex v that has exactly two neighbors $v_G \in \mathcal{G}_e$ and $v_K \in \mathcal{K}_e$. Observe that we may skip the possibility that $v \in \mathcal{W}$ due to Lemma 7.3.6. Details of the analysis are contained in Table 7.2.

- (a) Assume $d(v_G) \geq 3$ and $d(v_K) \geq 3$. Then, the worst-case recursion given in Table 7.2 arises. The two branches $v \in \mathcal{K}_e$ and $v \in \mathcal{G}_e$ are completely symmetric: e.g., if $v \in \mathcal{K}_e$, then it will be paired with v_G (by inactivating both of them), and Reduction Rule 14 will put all (at least two) neighbors of $N(v_G) \setminus \mathcal{G}_e$ into $\text{Not}\mathcal{G}$, and symmetrically all neighbors of $N(v_K) \setminus \mathcal{K}_e$ into $\text{Not}\mathcal{K}$ in the other branch.
- (b) Assume that v_G and v_K satisfy $d(v_G) = 2$ and $d(v_K) = 2$. Then, the recursion given in Table 7.2 arises. Assume first that v_G and v_K do not share another neighbor. Then, when v is put into \mathcal{K}_e , then v is paired up with v_G . Since the degree of v_K will then drop to one by Reduction Rule 6, v_K must have its garden on the only remaining neighbor s . This will be achieved with Reduction Rule 12. Observe that at this point we must have $s \in \mathcal{U}$. Therefore, all in all the measure decreases by $2 \cdot (1 - \omega_\ell)$; moreover, we turn at least one neighbor of v_G into a $\text{Not}\mathcal{K}$ -vertex giving a reduction of ω_n .

Secondly, it could be that v_G and v_K share one more neighbor, i.e., $N(v_G) = N(v_K) = \{q, v\}$. If q has any further neighbor z , then $z \in \mathcal{U}$ (confer Reduction Rule 13). Otherwise these vertices form a small component that is handled in line 5, since it has maximum degree two, which is a contradiction to the

current case. Thus, as q is paired with v_K the vertex z is turned into a Not \mathcal{K} -vertex.

The branch where we set $v \in \mathcal{G}_e$ is symmetric.

- (c) We now assume that $N(v) = \{v_K, v_G\}$, $d(v_G) \geq 3$, and $d(v_K) = 2$. Then, the worst-case recursion given in Table 7.2 arises. This can be seen by combining the arguments given for the preceding two cases.

3. Assume that all active vertices are in $\mathcal{U} \cup \mathcal{G}_e \cup \mathcal{K}_e$, with $\mathcal{G}_{e_a} \cup \mathcal{K}_{e_a} \neq \emptyset$. Note that $N(v) \subseteq \mathcal{U}$. Then, the algorithm tries to pair up some $v \in \mathcal{G}_{e_a} \cup \mathcal{K}_{e_a}$ of maximum degree. There are $d(v)$ branches for the cases labeled (3)# j . Since the two possibilities arising from $v \in \mathcal{G}_{e_a} \cup \mathcal{K}_{e_a}$ are completely symmetric, we focus on $v \in \mathcal{G}_{e_a}$. Exactly one neighbor u of $v \in \mathcal{G}_{e_a}$ will be paired with v in each step, i.e., we set $u \in \mathcal{K}_e$. Pairing the king on u with the garden from v will inactivate both u and v . Then, reduction rules will label all other neighbors of v with Not \mathcal{K} (they can no longer be kings), and symmetrically all other neighbors of u with Not \mathcal{G} . Note that $N(u) \setminus (\mathcal{K}_e \cup \{v\}) \neq \emptyset$, since otherwise a previous branching case or Reduction Rules 13 or 12 would have triggered. Thus, there must be some $q \in N(u) \cap \mathcal{U}$. From q , we obtain at least a measure decrease of ω_n , even if $q \in N(v)$. This results in a set of recursions depending on the degree of v as given in Table 7.2.
4. Finally, assume $V_a = \mathcal{U}$. Since an instance consisting of paths and cycles can be easily seen to be optimally solvable in polynomial time, we can assume that we can always find a vertex v of degree at least three to branch at. Details of the analysis are contained in Table 7.2. There are $d(v)$ branches for each of the cases (4 x)# j .

There are two cases to be considered: (a) either v has a neighbor u of degree two or (b) this is not the case.

The analysis of (a) yields the recursion given in Table 7.2. The first term can be explained by considering the case $v \in \mathcal{W}$. In that case, Reduction Rules 8 and 10 trigger with respect to u (as $d(u) = 2$) and yield the required measure change. Thus, u is put into \mathcal{K}_i and $N(u)$ in \mathcal{W} . Notice that all vertices have minimum degree of two at this stage due to Reduction Rule 10 and the fact that $V_a = \mathcal{U}$. In the case where $v \in \mathcal{K}_i$, the neighbors of v are added to \mathcal{W} , yielding the second term.

The last two terms are explained as follows: We simply consider all possibilities of putting $v \in \mathcal{K}_e \cup \mathcal{G}_e$ and looking for its partner in the neighborhood of v . Once paired up with some $u \in N(v)$, the other neighbors of $\{u, v\}$ will be placed into Not \mathcal{K} or Not \mathcal{G} , respectively.

In case (b), we obtain the recursion given in Table 7.2. This is seen by a slightly simplified but similar argument to what is written above. Notice that we can assume that $d(v) \geq 4$, since the case when $d(v) = 3$ (excluding degree-1 and degree-2 vertices in $N(v)$) that are handled either by Reduction Rule 10 or by the

previous case) will imply that the graph $G[V_a]$ is 3-regular due to our preference to branching on high-degree vertices. However, this can happen at most once in each path of the recursion, so that we can neglect it with respect to the \mathcal{O}^* -notation. Also note that for the vertices u and v which are paired up we have $d(v) + d(u) - 2 \geq d(v) + 1$ as $d(u) \geq 3$.

Finally, to show the claimed running time, we set $\omega_\ell = 0.7455$ and $\omega_n = 0.2455$ in the recurrences. If the measure drops below zero, then we argue that we can safely answer NO, as shown in Lemma 7.3.5. \square

Further Discussion of the Branching Cases We shall now show that out of the infinite number of recurrences in cases (3) and (4) that we derived for CO-MAXIR, actually only finitely many need to be considered.

Case (3)# j For this case we derived a recurrence of the following form where $i := d(v)$.

$$\begin{aligned} T(\varphi, h) &\leq i \cdot T(\varphi - ((1 - \omega_\ell) + i \cdot \omega_n), h - 1) \\ &\leq i \cdot 3.069^{k - ((1 - \omega_\ell) + i \cdot \omega_n)} \\ &= 3.069^k \cdot f(i) \end{aligned}$$

The first inequality should follow by induction on the height h of the search tree, while this entails $T(\varphi, h) \leq 3.069^k$ if $f(i) \leq 1$ for all i . We now discuss $f(i) = i \cdot 3.069^{-((1 - \omega_\ell) + i \cdot \omega_n)}$.

We had a closer look at the behavior of this function $f(i)$. Its derivative with respect to i is:

$$3.069^{(-0.2545 - 0.2455i)} - 0.2455 \cdot i \cdot 3.069^{(-0.2545 - 0.2455i)} \cdot \log(3.069)$$

The zero of this expression is at

$$z = 2000/491 / \log(3.069) \approx 3.6325$$

We validated that this is indeed a saddle point of f , and f is strictly decreasing from there on, yielding a value $f(z) < 0.9998$ for $z \geq 4$. Hence, it is enough to look into all recursions up to $i = 4$ in this case as also $f(3) < 0.988$.

Case4a# j We further have to look into:

$$T(\varphi, h) \leq T(\varphi - i, h - 1) + 2 \cdot i \cdot T(\varphi - (1 + i \cdot \omega_n), h - 1) + T(\varphi - 2, h - 1)$$

Hence, we have to discuss for $i \geq 3$

$$f(i) = 3.069^{-i} + 2 \cdot i \cdot 3.069^{-(1 + i \cdot \omega_n)} + 3.069^{-2}.$$

We find that $f(3) < 1$ and $f(4) < 1$ and that a saddle-point of $f(i)$ is between 3 and 4 (namely at 3.2) and f is strictly decreasing from 4 on. So, from $i = 4$ on, all values are strictly below one.

Case4b#j Finally, we investigate

$$T(\varphi, h) \leq T(\varphi - i, h - 1) + 2 \cdot i \cdot T(\varphi - (1 + (i + 1) \cdot \omega_n), h - 1) + T(\varphi - 1, h - 1)$$

So, investigate for $i \geq 4$ the function

$$f(i) = 3.069^{-i} + 2 \cdot i \cdot 3.069^{-(1+(i+1)\cdot\omega_n)} + 3.069^{-1}.$$

Again, we found that the saddle-point of f is between 3 and 4 and that f is strictly decreasing from 4 on, with $f(4) < 0.998$.

Corollary 7.3.9: $\text{IR}(G)$ can be computed in time $\mathcal{O}^*(1.96^n)$.

Proof. This can be seen by the balancing “win-win” approach described above, exhaustively testing irredundant candidate sets up to size $\frac{n}{2} - \epsilon$ where $\epsilon = 0.1001 \cdot n$ and running Algorithm 11 with all parameters $k \leq \frac{n}{2} + \epsilon$. \square

7.4. Conclusions

We presented a parameterized route to the solution of a yet unsolved question in exact algorithms. More specifically, we obtained a algorithm for computing the irredundance number running in time less than $\mathcal{O}^*(2^n)$ by devising appropriate parameterized algorithms (where the parameterization is via a bound k on the co-irredundant set) running in time less than $\mathcal{O}^*(4^k)$.

The natural question arises if one can avoid this detour to parameterized algorithmics to solve such a puzzle from exact exponential-time algorithmics. A possible non-parameterized attack on the problem is to adapt the measure φ . Doing this in a straightforward manner, we arrive at a measure $\tilde{\varphi}$:

$$\tilde{\varphi}(n, \mathcal{K}_i, \mathcal{K}_e, \mathcal{G}_e, \mathcal{W}, \text{Not}\mathcal{G}, \text{Not}\mathcal{K}, V_a) = n - |\mathcal{W}| - |\mathcal{G}_{ei}| - |\mathcal{K}_{ei}| - \tilde{\omega}_\ell (|\mathcal{K}_{ea}| + |\mathcal{G}_{ea}|) - \tilde{\omega}_n (|\text{Not}\mathcal{G}| + |\text{Not}\mathcal{K}|)$$

It is quite interesting that by adjusting the recurrences with respect to $\tilde{\varphi}$ a run time less then $\mathcal{O}^*(2^n)$ was not possible to achieve. For example, the recurrences under (1a) and (1bii) translate to $\tilde{T}(\varphi, h) \leq \tilde{T}(\varphi - (1 - \tilde{\omega}_n), h - 1) + \tilde{T}(\varphi - (\tilde{\omega}_\ell - \tilde{\omega}_n), h - 1)$ and $\tilde{T}(\varphi, h) \leq \tilde{T}(\varphi - (1 - \tilde{\omega}_n), h - 1) + \tilde{T}(\varphi - (2 - \tilde{\omega}_\ell + 2\tilde{\omega}_n), h - 1)$. Now optimizing over $\tilde{\omega}_\ell, \tilde{\omega}_n$ and the maximum over the two branching numbers alone we already arrive at a run time bound of $\mathcal{O}^*(2.036^n)$ (whereas $\tilde{\omega}_\ell = 1.13$ and $\tilde{\omega}_n = 0.08$). Thus, the parameterized approach was crucial for obtaining a run time upper bound better than the trivial enumeration barrier $\mathcal{O}^*(2^n)$. Observe that for these particular problems, allowing a weight of $\tilde{\omega}_\ell \in [0, 2]$ is valid, while usually only weights in $[0, 1]$ should be considered.

It would be interesting to see this approach used for other problems, as well. Some of the vertex partitioning parameters discussed in [149] seem to be appropriate.

We believe that the *Measure&Conquer*-approach could also be useful to find better algorithms for computing the lower irredundance number.

More broadly speaking, we think that an extended exchange of ideas between the field of Exact Exponential-Time Algorithms, in particular the *Measure&Conquer*-approach, and that of Parameterized Algorithms, could be beneficial for both areas. In our case, we would not have found the good parameterized search tree algorithms if we had not been used to the *Measure&Conquer*-approach, and conversely only via this route and the corresponding way of thinking we could break the 2^n -barrier for computing irredundance numbers.

Part II.

Applications of Reference Search Trees

Chapter 8.

An Exact Exponential Time Algorithm for POWER DOMINATING SET

8.1. Introduction

We study an extension of DOMINATING SET. The extension originates not from an additional required property of the solution set (e.g., CONNECTED DOMINATING SET) but by adding a second rule. To be precise we look for a vertex set, called power dominating set, such that every vertex is observed according to the next two rules:

Observation Rule 1 (OR1): A vertex in the power domination set observes itself and all of its neighbors.

Observation Rule 2 (OR2): If an observed vertex v of degree $d \geq 2$ is adjacent to $d - 1$ observed vertices, then the remaining unobserved neighbor becomes observed as well.

By skipping the second rule we would exactly arrive at DOMINATING SET. The second rule is responsible for the non-local character of the problem as it implements a kind of propagation. Due to this propagation mechanism a vertex can observe another vertex at arbitrary distance. Also the sequence of **OR2** applications can be arbitrary but leading to the same set of observed vertices. Indeed, many arguments relying on the locality of DOMINATING SET fail. There is no transformation to SET COVER and thus the algorithm of Fomin, Gradoni and Kratsch [74] for DOMINATING SET cannot simply be applied.

The problem occurs in the context of monitoring electric power networks. One wishes to place a minimum number of measurement devices (so called phase measurement units (PMU)) at certain points in the network to measure the state variables, which are fed back to the central control. The state of a power system is expressed in terms of state variables, such as voltage at a load and phase angle at a generator. In that sense **OR2** stands for Kirchhoff's law. See also D.J. Brueni and L.S. Heath [17] for a more technical overview of the applicability of the problem.

In this way, we arrive at the definition of the central problem:

POWER DOMINATING SET (PDS)

Given: An undirected graph $G = (V, E)$, and the parameter k .

We ask: Is there a set $P \subseteq V$ with $|P| \leq k$ which observes all vertices in V with

respect to the two observation rules **OR1** and **OR2**?

8.1.1. Discussion of Related Results

The study of PDS was initiated by T.W. Haynes *et al.* [88] where they showed \mathcal{NP} -hardness and gave a first polynomial time algorithm for trees. J. Guo *et al.* [86] and J. Kneis *et al.* [100] studied this problem independently with respect to parameterized complexity. They proved $W[2]$ -hardness if the parameter is the size of the solution by reducing DOMINATING SET to PDS. As a by-product it turns out that PDS is still \mathcal{NP} -hard on graphs with maximum degree four and that there is a lower bound for any approximation ratio of $\Omega(\log n)$ modulo standard complexity assumptions. Additionally, they showed fixed-parameter tractability of PDS with respect to tree-width, where [86] also gave a concrete algorithm. The authors of [86] achieve this by transforming PDS into an orientation problem on undirected graphs. The problem was also studied in the context of special graph classes like interval graphs (C.-S. Liao and D.-T. Lee [111]) and block graphs (G. Xu *et al.* [157]) where linear time algorithms were obtained. A. Aazami and M.D. Stilp [1] raised the approximation lower bound to $\Omega(2^{\log^{1-\epsilon} n})$ and gave an $\mathcal{O}(\sqrt{n})$ -approximation for planar graphs. On the other hand also domination problems have been studied in exact algorithmics. F.V. Fomin, F. Gradoni and D. Kratsch [74] gave a $\mathcal{O}^*(1.5137^n)$ -algorithm for DOMINATING SET where they use the power of the *Measure&Conquer*-approach. This could be improved by J.M.M. van Rooij and H.L. Bodlaender [151] to $\mathcal{O}^*(1.5134^n)$ and thereafter by J.M.M. van Rooij *et al.* [153] to $\mathcal{O}^*(1.5048^n)$. F.V. Fomin, F. Gradoni and D. Kratsch [73] showed that the variant CONNECTED DOMINATING SET can be solved in time $\mathcal{O}^*(1.9407^n)$. Under the alternative name MAXIMUM LEAF SPANNING TREE this upper bound could be updated to $\mathcal{O}^*(1.8966^n)$ by H. Fernau *et al.* [51]. D.J. Brueni and L.S. Heath [17] showed that there exists a power dominating set of size at most $\lceil n/3 \rceil$ for a graph with at least three vertices. By cycling through all the candidate sets this implies an algorithm with run time $\mathcal{O}^*(1.89^n)$.

8.1.2. New Results

First, we show that PDS remains \mathcal{NP} -hard for cubic graphs. As PDS is polynomial time solvable for max-degree-two graphs and \mathcal{NP} -hardness was shown for max-degree-four graphs [86, 100], this result closes the gap in-between. Furthermore, this justifies to follow a branching strategy even in the case of cubic graphs. Note that it is not always true that generally \mathcal{NP} -hard graph problems remain \mathcal{NP} -hard for cubic graphs. FEEDBACK VERTEX SET is a problem where as with PDS cycles play a role (see [86]). But in contrast to general graphs, it is solvable in polynomial time on cubic graphs [146]. Secondly, we present an algorithm solving PDS in time $\mathcal{O}^*(1.7548^n)$, which improves upon the simple enumeration based algorithm with run time upper bound $\mathcal{O}^*(1.89^n)$. The run time analysis proceeds in an amortized fashion using the *Measure&Conquer*-approach (see chapter 2). Furthermore, we use the concept of a reference search tree. In an ordinary search tree we usually cut off branches due to local structural conditions.

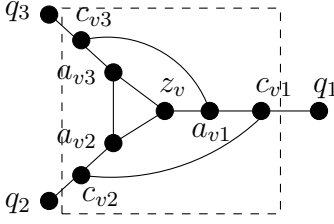


Figure 8.1.: The gadget T_v . The vertices q_1, q_2, q_3 correspond to vertices of the form c_{z_i} of some other gadget T_z such that $z \in V$.

In a reference search tree we also will cut off branches if we can point to another node of the search tree where we can find no worse solutions. This node must not be a neighbor of the current node but can be anywhere in the search tree, as long as the overall search structure remains acyclic.

8.1.3. Terminology and Notation

A possible solution set will be denoted P . We call a vertex $v \in V \setminus P$ *directly observed* by $u \in N(v)$ if u is in the solution, i.e., $u \in P$. The vertex $v \in V \setminus P$ will be called *indirectly observed* by $u \in V \setminus P$ if v is observed due to the application of **OR2** onto u .

8.2. \mathcal{NP} -hardness of Planar Cubic Power Dominating Set

We will reduce PLANAR CUBIC VERTEX COVER to PLANAR CUBIC PDS. Due to [76] VERTEX COVER remains \mathcal{NP} -complete on planar cubic graphs. For any planar cubic graph $G(V, E)$ and any $v \in V$ we can denominate the neighbors of v as follows: $N(v) = \{n_{v_1}, n_{v_2}, n_{v_3}\}$.

The reduction works as follows: Given a planar cubic graph $G(V, E)$ introduce for every $v \in V$ the gadget T_v depicted in Figure 8.1, which consists of the vertices in the dotted square. For any $\{u, v\} \in E$ we can find $1 \leq b, c \leq 3$ such that $u = n_{v_b}$ and $v = n_{u_c}$. By introducing the edge $\{c_{v_b}, c_{u_c}\}$ we finally get $G'(V', E')$ which is planar and cubic.

Lemma 8.2.1: G has a vertex cover of size $\leq k$ iff G' has a PDS of size $\leq k$.

Before proving Lemma 8.2.1, we exhibit some properties of a PDS P for G' on which we can rely on.

Lemma 8.2.2: If G' has a PDS P with $|P| \leq k$ then there is also a PDS P' with $|P'| \leq |P|$ such that for all gadgets T_v we have:

1. $|V(T_v) \cap P'| \leq 1$.
2. If $|V(T_v) \cap P'| = 1$ then $V(T_v) \cap P' = \{z_v\}$ and $\{z_v\}$ is a PDS for $G'(N[V(T_v)])$.
3. If $|V(T_v) \cap P'| = 0$ then a_{v_i} is indirectly observed by c_{v_i} , $1 \leq i \leq 3$, and z_v is indirectly observed by a_{v_1} .

Proof.

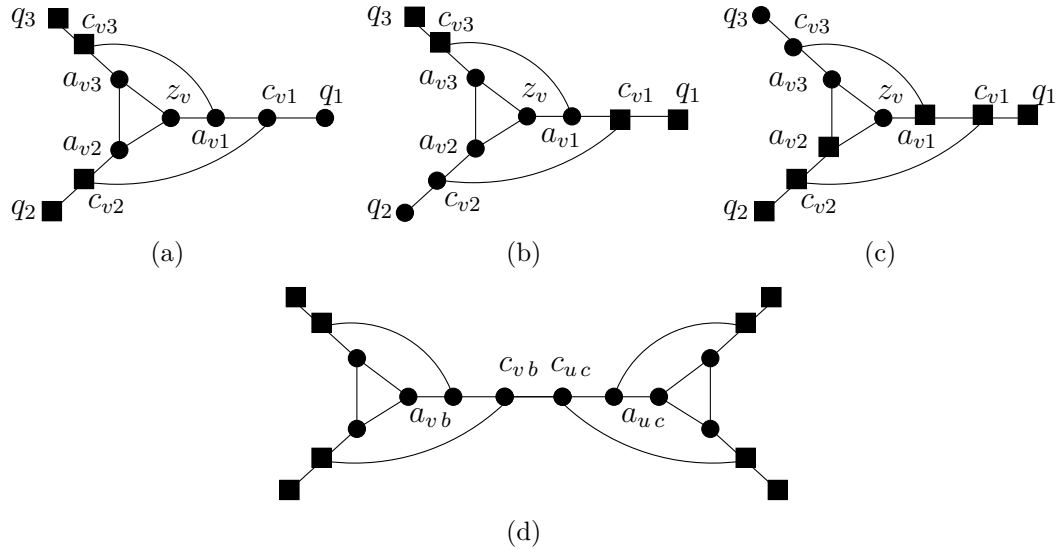


Figure 8.2.: Observed vertices are depicted as filled squares.

1,2: Let P be an arbitrary PDS for G' such that $|P| \leq k$ and $r := |P \cap (\cup_{v \in V} z_v)|$ is maximum. Assume that $T_v \cap P \neq \emptyset$ but $z_v \notin P$. Set $P' = (P \setminus V(T_v)) \cup \{z_v\}$. Independently of the vertices in $P' \setminus \{z_v\}$, due to $z_v \in P'$ and by exhaustively applying **OR2**, the vertices a_{vi} , q_i and c_{vi} ($1 \leq i \leq 3$) will be observed.

As $G' \setminus V(T_v)$ only interacts via the q_i 's with T_v and as they are also observed in the mentioned manner, P' must be a PDS. But we have $|P' \cap (\cup_{v \in V} z_v)| > r$ which is a contradiction. This shows 1) and 2).

3: To show 3) observe the following. If $|G[T_v] \cap P'| = 0$ then T_v must be observed by using **OR2** and therefore there must be a c_{vi} which is indirectly observed by a vertex outside $V(T_v)$. If only one of the c_{vi} 's is indirectly observed (necessarily by q_i) then observation cannot propagate further by **OR2** as any c_{vi} would have two unobserved neighbors within T_v . To be explicit, in case c_{v1} is observed the two vertices are a_{v1}, c_{v2} , in case c_{v2} we have a_{v2}, c_{v1} and in case c_{v3} the vertices a_{v3}, a_{v1} are unobserved. Thus there are unobserved vertices in T_v and P is not a PDS. Now we must rule out that there are exactly two vertices c_{vi}, c_{vj} being each observed by q_i and q_j . There are exactly three possibilities for this which are depicted in Figures 8.2(a), 8.2(b) and 8.2(c) where we also applied **OR2** exhaustively. In any case some unobserved vertices remain. Note that if in Figure 8.2(c) also c_{v3} is observed then an **OR2** application is triggered on a_{v1} which observes z_v . Then due to another **OR2** application on c_{v3} observation reaches a_{v3} and hence T_v is observed showing 3). \square

Lemma 8.2.1. \Rightarrow : Let S be a vertex cover with $|S| \leq k$ for G . Let $V = \{v_1, \dots, v_n\}$. We want to point out that for every vertex in $u \in V \setminus S$ we have $N(u) \subseteq S$. Otherwise, S is not a vertex cover. Now construct a PDS P the following way. For every $v \in S$

add the vertex z_v of T_v to P . Observe that any gadget T_v with $v \in P$ is now completely observed after applying **OR2** exhaustively (independent of the vertices in $P \setminus \{v\}$ by Lemma 8.2.2.2). We also can apply **OR2** to the c_{vi} such that their neighbor q_i ' outside T_v will be observed. If $v \notin S$ then due to $N(v) \subseteq S$ and our last observation we have that any a_{vi} is indirectly observed by q_i . Then by **OR2** observation propagates to all other vertices in T_v analogously as in Lemma 8.2.2.3.

\Leftarrow : Let P be a PDS with $|P| \leq k$ for G' . Due to Lemma 8.2.2.2 we assume that, w.l.o.g., $P = \{z_1, \dots, z_\ell\}$ with $\ell \leq k$. Also due to Lemma 8.2.2.3 in a gadget T_{v_t} with $t > \ell$ the corresponding vertices c_{vt_i} are indirectly observed by q_i . Now suppose that $VC := \{v_1, \dots, v_\ell\}$ is not a vertex cover in G . This means there is an edge $\{u, v\} \cap VC = \emptyset$ (*). Thus, there must be some c_{vb} and c_{uc} ($1 \leq b, c \leq 3$) with $\{c_{vb}, c_{uc}\} \in E_{G'}$, see Figure 8.2(d). Since $u, v \notin VC$ and, thus, $u, v \notin P$ (due to (*)) and by Lemma 8.2.2.3 follows that c_{vb} is indirectly observed by c_{uc} and vice versa, see Figure 8.2(d). This is a contradiction to the definition of **OR2**. \square

As the gadget T_v is planar the graph G' posses the same property. According to Lemma 8.2.1 we can conclude

Corollary 8.2.3: PDS remains \mathcal{NP} -hard on planar cubic graphs.

8.3. An Exact Algorithm for Power Dominating Set

8.3.1. Annotated Power Dominating Set

8.3.1.1. Definitions

Annotation. In what follows we assume that the vertices of the given graph $G(V, E)$ are annotated. To be precise we have a function s which assigns a label to every vertex:

$$s : V(G) \rightarrow \{\text{active}, \text{inactive}, \text{blank}\}.$$

An *active* (*inactive*, resp.) vertex has already been determined to be (not to be, resp.) part of P . For a *blank* vertex this decision has been not made yet. We will abbreviate the three attributes by (a) , (i) and (b) . We also define $A := \{v \in V(G) \mid s(v) = (a)\}$, $I := \{v \in V(G) \mid s(v) = (i)\}$ and $B := \{v \in V(G) \mid s(v) = (b)\}$.

For any given set A we can determine which vertices are already observed by applying exhaustively **OR1** and **OR2**. Due to this we introduce

$$s' : V(G) \rightarrow \{(o)\text{bserved}, (u)\text{nobserved}\}$$

and the sets $O := \{v \in V(G) \mid s'(v) = (o)\}$ and $U := V(G) \setminus O$. By O we will always refer to the set of observed and by U to the set of unobserved vertices (with respect to the current partial solution P). The *state* of a vertex v is the tuple $(s(v), s'(v))$. During the course of the algorithm the states of the vertices (i.e., the labels s, s') will be modified in a way that they represent choices already made.

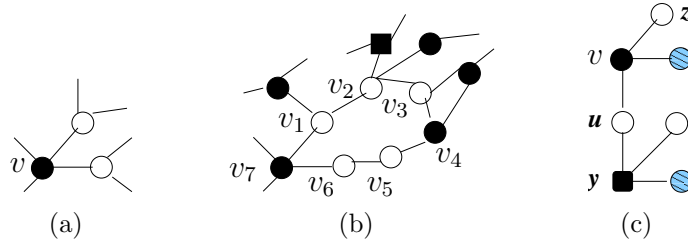


Figure 8.3.: Filled vertices are observed, white vertices are unobserved. Round vertices are blank, square vertices are inactive. Shaded vertices are active.

Special Neighborhoods. We set $N^*(v) := \{w \mid \{w, v\} \in E, s'(w) = (u)\}$ and $d^*(v) := |N^*(v)|$. $N^*(v)$ represents the unobserved neighbors of v . Let $\Delta^*(G) := \max_{v \in B} d^*(v)$ and $M(G) = \{v \in B \mid d^*(v) = \Delta^*(G)\}$. Thus, in $M(G)$ we find those blank vertices with the maximum number of unobserved neighbors.

We define $N^{(i)}(v) = N^*(v) \cap I$ and $d^{(i)}(v) = |N^{(i)}(v)|$. In $N^{(i)}(v)$ we find those unobserved neighbors of v , which are additionally inactive. We will write $d_G^*(v)$, $N_G^*(v)$, $d_G^{(i)}$, $N_G^{(i)}(v)$, $s_G(v)$ and $s'_G(v)$ when we are referring to a particular annotated graph G by which the functions are induced. We omit the subscript when it is clear from the context. A vertex $v \in V(G)$ such that $s'(v) = (o)$ and $d^*(v) = 2$ will be called a *trigger*, see Figure 8.3(a). A *triggered path* between $v_1, v_k \in V(G)$ with $s(v_1) = s(v_k) = (b)$, $s'(v_1) = s'(v_k) = (u)$ is a path v_1, \dots, v_k such that $s(v_i) = (b)$, $s(v_i) = (u)$ and $d^*(v_i) \leq 2$, or v_i is a trigger for $1 < i < k$. In Figure 8.3(b) the vertices v_1, \dots, v_6 form a triggered path. A *triggered cycle* is a triggered path with $v_1 = v_k$. Observe that for all $u \in O$ we have $d^*(u) \neq 1$ due to **OR2**.

8.3.2. Algorithm

In this section we present reduction rules and the algorithm. Their correctness and run time will be analyzed in the next section.

Reduction Rules. We state the following reduction rules:

Isolated: Let $v \in O \cap B$ such that $d^*(v) = 0$ then set $s(v) \leftarrow (i)$.

TrigR: Let $v \in V$ be a trigger and $s(v) = (b)$. Then set $s(v) \leftarrow (i)$.

Blank2: Let $v \in V(G)$ with $d^*(v) \leq 2$, $v \in B \cap U$, $y \in N^*(v)$ and $s(y) = (i)$. Then set $s(v) \leftarrow (i)$.

Obs3: Let $v \in V(G)$ such that $v \in B \cap U$, $d^*(v) \leq 1$ and $y \in N(v)$ with $y \in I \cap O$ and $d^*(y) \geq 3$. Then set $s(v) \leftarrow (i)$.

Trig2: Let $v \in V(G)$ such that $v \in B \cap U$ and $d^*(v) \leq 1$. If there is a trigger u with $N^*(u) = \{v, y\}$ and $y \in I \cap U$ then set $s(v) = (i)$.

In [88] it is shown that we can assume that for all $v \in P$ we have $d(v) \geq 3$. But observe that for degree-2 vertices there is no valid contraction rule, see Figure 8.3(c). If we

deleted u and connected x and y observation would propagate to z due to **OR2**. We are now ready to state Algorithm 12:

Algorithm 12 An exact algorithm for POWER DOMINATING SET

- 1: Apply **OR2** exhaustively.
 - 2: Apply **Isolated**, **TrigR**, **Blank2**, **Obs3** and **Trig2** exhaustively.
 - 3: Select from $M(G)$ a vertex v according to the priorities:
 - 4: a) $s(v) = (u)$. { We prefer unobserved vertices }
 - b) $d^{(i)}(v) < d^*(v)$ { We prefer vertices such that not all unobserved neighbors are inactive }
 - 5: **if** $d^*(v) \geq 4$ **then**
 - 6: Branch on v by setting 1) $s(v) \leftarrow (i)$ and 2) $s(v) \leftarrow (a)$ in either of the branches.
 - 7: **else if** $d^*(v) = 3$ **then**
 - 8: Branch on v : 1) $s(v) \leftarrow (i)$ and 2) $s(v) \leftarrow (a)$ and for all $u \in N^*(v)$ with $s(u) = (b)$ set $s(u) \leftarrow (i)$.
 - 9: **else if** $d^*(v) \leq 2$ **then**
 - 10: Branch on v by setting 1) $s(v) \leftarrow (i)$ and 2) $s(v) \leftarrow (a)$ in either of the branches.
 - 11: **end if**
-

8.3.2.1. Correctness

We will prove correctness of Algorithm 12 and the reduction rules using the concept of a reference search tree (see Definition 1.4.1). Intuitively, this is quite appropriate for our problem at hand. In PDS the second propagation rule **OR2** gives the problem a non local character, which will be reflected by item 4.(b) of Definition 1.4.1. We have to define $\mathcal{U} := V(G)$, $\mathcal{S} := \{S \subseteq V(G) \mid S \text{ is a PDS for } G\}$ and $c(Y) = |Y|$ for every $Y \in \mathcal{P}(\mathcal{U})$. The function $label : V(D) \rightarrow \{(e_1, \dots, e_n) \mid e_i \in \{(a), (i), (b)\}\}$ then expresses which vertices are no more blank, i.e., are active or inactive. Here (a) refers to 1, (i) to 0 and (b) to the \star -symbol defined in the function $label$ of Definition 1.4.1. According to this we set $(\bar{a}) = (i)$ and $(\bar{i}) = (a)$. Thus, referring to Definition 1.4.2 when we set $s(v) \leftarrow (i)$ then we are excluding the vertex v from the future solution. By setting $s(v) \leftarrow (a)$ it will be included. The subsequent correctness proofs proceed as follows: Every time we skip a possible solution we show that we can insert a reference to some node $u \in V(D)$ of the search tree such that we can find a solution z with $label(z) \preceq label(u)$ which is no worse. Additionally, we show that the global references always point from the left to the right (with respect to the x -coordinate) and the local references point downwards in the proper drawing of $D(V, T)$. This way we assure acyclicity of the final rst $D(V, T \cup R \cup L)$ which is implicitly built up by the algorithm using Lemma 1.4.1.

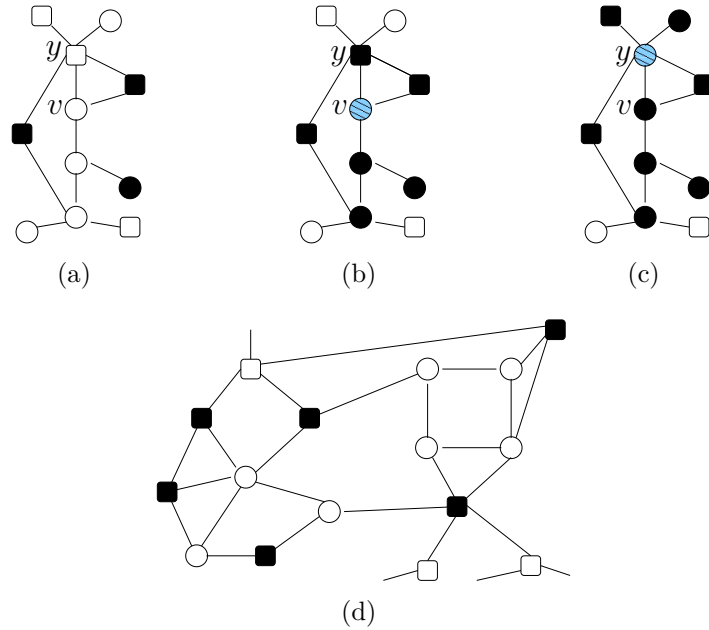


Figure 8.4.: Filled vertices are observed, white vertices are unobserved. Round vertices are blank, square vertices are inactive. Shaded vertices are active.

Correctness Proofs.

Lemma 8.3.1: Let us fix an annotated PDS instance $G(V, E)$ that corresponds to some node $q \in V(D)$ in the reference search tree. Let $u \in V(G)$ with $u \in I \cap O$ and $d_G^*(u) \geq 3$. Then $d_G^q \in D(V, T)$ is a full node.

Proof. Suppose the contrary. Due to $d_G^*(u) \geq 3$ none of the reduction rules in Algorithm 12 have set $s(u) \leftarrow (i)$. The only remaining possibility is the second part of the branch in step 8 of Algorithm 12. Now suppose by setting $s(v) \leftarrow (a)$ for some $v \in V(G)$ the algorithm has set also $s(u) \leftarrow (i)$ and $s'(u) = (o)$ implicitly. We now examine the situation right before this happened. This situation is reflected by some annotated graph $G'(V, E)$. We must have $d_{G'}^*(v) = 3$ and $s'_{G'}(u) = (u)$. Suppose at this point $s'_{G'}(v) = (u)$. From this it follows that $d_{G'}^*(u) \geq 4$ due to our premise. But this contradicts the choice of v as branch vertex. Therefore we must have $s'_{G'}(v) = (o)$. But once more this contradicts the choice of v since we have $s'_{G'}(u) = (u)$ at that point (as step 8 applied u should have been observed by v directly). \square

Lemma 8.3.2: Applying **Blank2**, **Obs3**, **Trig2**, **TrigR**, **Isolated** and step 8 of Algorithm 12 is correct.

Proof. We will prove the following:

1) For every vertex $v \in V(G)$ with $s(v) = (i)$ either $p(d_v)$ is a full node or if it is flat there is a node $h \in V(D)$ such that $p(d_h)$ is a full node and we inserted a reference $(p(d_v), h)$ where $h = r(p(d_h))$. Additionally, we require that if there is a solution represented by the solvec $t \preceq \text{label}(p(d_v))$ with $t(v) = (a)$ (i.e., $t \in ST_{r(p(d_v))}$), we can find a no worse

solution represented by solvec z (i.e., $c(z) \leq c(t)$) such that $z \preceq \text{label}(h)$.

2) Every reference is pointing from the left to the right in the proper drawing of $D(V, T)$. As step 8 makes use of this fact, it will be proven in parallel. The proof is by induction on the height s of the search tree. In case $s = 0$ nothing is to show. If $s > 0$ we will distinguish between the different operations:

Blank2 Let q be the current search tree node and, w.l.o.g.,

$\text{label}(q) = (e_1, \dots, e_{l-1}, (b), \dots, (b))$ such that e_l corresponds to v and e_1 to y (with v and y we are referring to the definition of **Blank2**). Suppose **Blank2** applies to v and $p(d_y^q)$ is a full node (see Figure 8.4(a)). Suppose there is a solution $k := (e_1, \dots, e_{l-1}, (a), e_{l+1}, \dots, e_n) \preceq \text{label}(q)$ as indicated in Figure 8.4(b). Then also $k' := (\bar{e}_1, \dots, e_{l-1}, (i), e_{l+1}, \dots, e_n)$ is a solution (due to $d^*(v) \leq 2$ and **OR2**, see Figure 8.4(c)). Hence we insert a reference $(q, r(p(d_y^q)))$ as $k' \preceq \text{label}(r(p(d_y^q)))$ which is pointing from the left to the right. This reference means that we can find a no worse solution compared to k . We find this solution in the sub search tree $ST_{r(p(d_y^q))}$ as $k' \preceq \text{label}(r(p(d_y^q)))$ or else we have to follow another reference to the right. Therefore we can skip the possibility of setting $s(v) \leftarrow (a)$.

If $p(d_y^q)$ is a flat node than due to the induction hypothesis there must be a reference $(p(d_y^q), h)$ such that $h \in V(D)$ is the right child of $p(h) \in V(D)$ which is a full node and $(p(d_y^q), h)$ points from the left to the right. We can rule out the solution k again as k' is no worse. Due to $(p(d_y^q), h)$ also k' is skipped as there must be a alternative solution z with $\text{label}(z) \preceq h$ such that z is no worse than k' . Thus we can insert the reference (q, h) pointing also from the left to the right.

Trig2 The proof is completely analogous to the first item.

Obs3 From Lemma 8.3.1 we have that y is a full node. The correctness follows now analogously to the first part of the first item.

TrigR Let $N^*(v) = \{x, y\}$ and $q \in D(V)$ be the current search tree node before applying **TrigR** and let, w.l.o.g., $\text{label}(q) = (e_1, \dots, e_l, (b), \dots, (b))$, e_1 corresponds to y and e_{l+1} to v . Let P be a solution which contains v (i.e. $s(v) = (a)$) such that $sv_P \preceq \text{label}(q)$. a) Suppose that, w.l.o.g., $s(y) = (b)$. Then P could be replaced by $P' = P \setminus \{v\} \cup \{y\}$. Thus, by $s(y) = (b)$ we can set $s(v) \leftarrow (i)$ as we will find a solution which is no worse later in the sub search tree $ST_{l(q)}$.

b) Now suppose $s(x) = s(y) = (i)$. If there is a solution corresponding to $k := (e_1, \dots, e_l, (a), e_{l+2}, \dots, e_n) \preceq \text{label}(q)$ then also $k' := (\bar{e}_1, \dots, e_l, (i), e_{l+1}, \dots, e_n)$ is one which is no worse. If $p(d_y^q)$ is a full node we have that $k' \preceq \text{label}(r(p(d_y^q)))$ and we insert a reference $(q, r(p(d_y^q)))$. If $p(d_y^q)$ is a flat node there is a reference $(p(d_y^q), h')$. Analogously as in the cases before we can insert (q, h') .

Isolated As every vertex in $N[v]$ is already observed it is valid to exclude v from a minimum solution. Thus, we can set $s(v) \leftarrow i$.

Step 8 We only have to consider the second part of the branch. Here we set $s(v) \leftarrow (a)$ and for all $u \in N^*(v)$ with $s(u) = (b)$ also $s(u) \leftarrow (i)$. Thus, we have to argue that indeed the neighbors of v could have been set inactive.

Let $N^*(v) = \{a, b, c\}$. We make a case distinction concerning $d^{(i)}(v)$. Let G the current annotated graph and $q \in D(V)$ be the current search tree node before branching and let $\text{label}(q) = (e_1, \dots, e_l, (b), \dots, (b))$, where the entries e_1 corresponds to v , e_2 to a and e_3 to b . Therefore we have $e_1 = (b)$. Assume there is a PDS $P \ni v$ with $sv_P \preceq \text{label}(q)$.

$d_G^{(i)}(v) = 0$: If $|P \cap N^*(v)| \in \{2, 3\}$ then also $P \setminus \{v\}$ is a PDS due to **OR2**. If, w.l.o.g, $P \cap N^*(v) = \{a\}$ then also $P' := P \setminus \{v\} \cup \{b\}$ is a PDS where P' is covered by the first part of the branch where we set $s(v) \leftarrow (i)$.

$d_G^{(i)}(v) = 1$: The only cases $|P \cap N^*(v)| = 1$ and $|P \cap N^*(v)| = 2$ can be handled analogously to the case $d^{(i)} = 0$.

$d_G^{(i)}(v) = 2$: W.l.o.g., $P \cap N^*(v) = \{a\}$. Let $q_1 \in V(D)$ be the node such that $s(v) = a$ is set and $q_2 \in V(D)$ such that $s(a) = (i)$ is set, i.e. $q_1 = r(q)$ and q_2 is the only child of q_1 .

Assume there is a PDS corresponding to $k := ((a), (a), (i), e_4, \dots, e_n) \preceq \text{label}(q_1)$; By **OR2** $k' := ((i), (a), (a), e_4, \dots, e_n)$ is then a solution, too. Suppose $p(d_b^q)$ is a full node. Then $k' \preceq \text{label}(r(p(d_b^q)))$ and hence we insert a reference $(q_1, r(p(d_b^q)))$. If $p(d_b^q)$ is a flat node there must be a reference $(p(d_b^q), h')$. Then insert (q_1, h') .

The global references inserted by **Blank2**, **Trig2**, case b) of **TrigR**, **Obs3** and **Step 8** in case $d_G^{(i)}(v) = 2$ are pointing all from the left to the right in the proper drawing of $D(V, T)$.

Local references which point downwards are inserted by case a) of **TrigR**, **Isolated** and **Step 8** in cases $d_G^{(i)}(v) = 1$ and $d_G^{(i)}(v) = 2$. It can be checked easily that for an inserted local reference $(a, b) \in L$ we have $(b, a) \notin L$.

This ensures acyclicity of $D(V, T \cup R \cup L)$ by Lemma 1.4.1 and, thus, also the correctness. \square

Note that the reduction rules treated in Lemma 8.3.2 are not valid on their own. They are only correct because they are referring to solutions which Algorithm 12 definitely will consider. In other words, if we are given an annotated graph G , where the annotation is not due to Algorithm 12 we cannot apply these reduction rules.

8.3.2.2. Run Time Analysis

The Measure. We define the following sets:

$$\hat{I} = \{v \in I \mid s'(v) \neq (o), \exists u \in N^*(v) : s(u) = (b)\},$$

$$\hat{O} = (O \cap B), \quad \hat{B} = B \cap U.$$

Here \hat{I} comprises the inactive vertices, which are not observed such that they have at

least one unobserved neighbor which is blank. In \hat{O} we find the observed vertices for which we have not yet decided if there are active or not. Also for any $v \in \hat{O}$ we have $d^*(v) \geq 3$ (**Isolated**, **OR2** and **TrigR**). \hat{B} contains the unobserved blank vertices. The measure we use in our run time estimation is the following one:

$$\varphi(G) = |\hat{B}| + \beta \cdot |\hat{O}| + \gamma \cdot |\hat{I}| \text{ with } \beta = 0.51159, \gamma = 0.48842$$

The Different Branching Cases. We will now analyze the different branchings in Algorithm 12. In general we can find integers ℓ, k with $\ell + k = d^*(v)$ such that $\ell = |N^*(v) \cap \hat{I}|$ and $k = |N^*(v) \cap \hat{B}|$.

$d^*(\mathbf{v}) \geq 4$: The first case is when we have chosen a vertex v with $d := d^*(v)$. We will explicitly analyze the case when $d = 4$. We show that any case occurring for $d > 4$ is run time upper bounded by some case when $d = 4$. First we will distinguish between the circumstances that $s'(v) = (o)$ and $s'(v) = (u)$.

$s'(v) = (u)$ In the branch where we set $s(v) \leftarrow (a)$ we get a reduction in $\varphi(G)$ of $1 + \ell \cdot \gamma + k \cdot (1 - \beta)$. This is due to v becoming observed and active, the vertices in $N^*(v) \cap \hat{I}$ becoming inactive and observed and $N^*(v) \cap \hat{B}$ becoming observed and blank. In the branch setting $s(v) \leftarrow (i)$ we reduce $\varphi(G)$ by at least $(1 - \gamma)$ (we obtain a greater reduction if v drops out of \hat{I}). As $(1 - \beta) < \gamma$ the worst case is the branch $(1 + 4(1 - \beta), 1 - \gamma)$ which is upper bounded by $\mathcal{O}^*(1.6532^n)$.

$s'(v) = (o)$ In the branch where we set $s(v) \leftarrow (a)$ we get a reduction in $\varphi(G)$ of $\beta + \ell \cdot \gamma + k \cdot (1 - \beta)$. Here we get only a reduction of β from v as it is already observed. In case $s(v) \leftarrow (i)$ the reduction is again β as v drops out of \hat{O} . As $(\beta + 4(1 - \beta), \beta)$ is the worst branch we have a upper bound of $\mathcal{O}^*(1.7548^n)$.

We examine now cases with $d > 4$. Here the worst case is analogously when $k = d$. But it is also no worse as the case when $k = 4$ and $d = 4$, which was already considered.

$d^*(\mathbf{v}) = 3$: We first focus on the case where $\ell \leq 2$. As we get a reduction of one for every vertex in $N^*(v) \cap \hat{B}$ the worst case is when $\ell = 2$. Now if $s'(v) = (u)$ then this results in a $(2 + 2 \cdot \gamma, 1 - \gamma)$ branching. If $s'(v) = (o)$ we have a $(\beta + 2 \cdot \gamma + 1, \beta)$ -branching. $\mathcal{O}^*(1.7489^n)$ is an upper bound for both.

Now due to the priorities in step 4 of Algorithm 12 we select a vertex v such that $\ell = 3$ with least priority.

We first examine the case where $s'(v) = (u)$ and $\ell = 3$. Now suppose for all $u \in N^*(v)$ we have $N^*(u) \cap B = \{v\}$ (\star). Then in the branch $s(v) \leftarrow (i)$ we get an additional amount of $3 \cdot \gamma$. This is due to the fact that the vertices in $N^*(v)$ will drop out of \hat{I} as v is their only unobserved neighbor which is blank. Hence, we have a $(1 + 3 \cdot \gamma, (1 - \gamma) + 3 \cdot \gamma)$ branch.

Conversely, there is a $u \in N^*(v)$ with $N^*(u) \cap B = \{v, u_1, \dots, u_s\}$ and $s \geq 1$ (\ast).

If $s'(u_1) = (o)$ then due to **TrigR** and **OR2** we can not have $d^*(u_1) \leq 2$. Due to the choice of v $d^*(u_1) > 4$ is not possible and, thus, $d^*(u_1) = 3$. In $s(v) \leftarrow (a)$ u will become a trigger and is reduced away from $\varphi(G)$ due to **TrigR**. This means we have a $(1 + 3 \cdot \gamma + \beta, 1 - \gamma)$ branch.

If $s'(u_1) = (u)$ then we have $d^*(u_1) = 3$ due to **Blank2** and the choice of v . Also it holds that $d^*(u_1) = d^{(i)}(u_1) = 3$ again by the choice of v . Hence in $s(v) \leftarrow (a)$ the \star -degree of u_1 drops by one. Therefore **Blank2** applies on u_1 and it will not appear in $\varphi(G)$ anymore which leads to a $(2 + 3 \cdot \gamma, 1 - \gamma)$ branch. $\mathcal{O}^*(1.6489^n)$ upper bounds both possibilities.

The second possibility for v is $s'(v) = (o)$, yielding a $(\beta + 3 \cdot \gamma, \beta + 3 \cdot \gamma)$ -branch for case (\star) . In case of (\star) , $s'(u_1) = (o)$ is necessary or otherwise, we contradict the choice of v ($d^*(u_1) \geq 3$), or **Blank2** applies to u_1 ($d^*(u_1) \leq 2$). Again we have $d^{(i)}(u_1) = 3$ by the choice of v . Hence, this gives a $(2\beta + 3\gamma, \beta)$ -branch, as by setting $s(v) \leftarrow (a)$ u_1 becomes a trigger and therefore **TrigR** applies. An upper bound for both cases is $\mathcal{O}^*(1.7488^n)$.

$d^*(v) \leq 2$: Note that at this point by **Isolated**, **TrigR** and **OR2** we must have $s'(v) = (u)$. We first show an auxiliary lemma:

Lemma 8.3.3: In step 10 of Algorithm 12 we have:

1. For all $u \in V$ with $d^*(u) \geq 3$ it follows that $s(u) = (i)$.
2. Let $v \in V(G)$ with $s(v) = (b)$ and $s'(v) = (u)$ chosen for branching then:
 - a) For all $u \in N^*(v) : s(u) = (b)$.
 - b) If $d^*(v) \leq 1$ then for all $u \in N(v) \setminus N^*(v) : s(u) = (i)$ and $d^*(u) = 2$.
3. $O = O \cap I$. ($O \cap B = \emptyset$, alternatively).

Proof. 1. Otherwise, we have a contradiction to the choice of v .

2. a) Otherwise, **Blank2** applies.
 - b) Note that $s'(u) = (o)$. Suppose $s(u) = (b)$ then either **TrigR** or **OR2** applies ($d^*(u) \leq 2$) or we have a contradiction to the choice of v ($d^*(u) \geq 3$). From $s(u) = (a)$ it follows that $s'(v) = (o)$, a contradiction. If we had $d^*(u) \geq 3$ and $s(u) = (i)$ then **Obs3** applies. This contradicts $s(v) = (b)$.
3. Let $u \in O \setminus I$ then $d^*(u) \in \{0, 1, 2\}$ is ruled out by **Isolated**, **OR2** and **TrigR**. If $d^*(u) \geq 3$ then from item 1. it follows that $u \in I$, a contradiction. □

Let v be the vertex chosen in step 9 of Algorithm 12. Let $\tilde{G} := G[B]$ and note that $B = \hat{B}$ due to Lemma 8.3.3.3. \tilde{G} consists of paths and cycles formed by vertices in \hat{B} due

to Lemma 8.3.3.2a and the fact that for all $z \in B$ we have $d^*(z) \leq 2$ (see Figure 8.4(d)). The vertex v belongs to one of those components.

Explore G the following way:

1. For all $u \in \hat{B}$ set $visited(u) \leftarrow f$.
2. If there is $u \in N^*(v)$ with $visited(u) = f$ then set $visited(v) \leftarrow t$ and $v \leftarrow u$.
3. If there is $t \in N(v)$ with $t \in O$ (due to Lemma 8.3.3.2b u is a trigger) such that $N^*(t) = \{v, u\}$ and $visited(u) = f$ then set $visited(v) \leftarrow t$ and $v \leftarrow u$.
4. If one of the steps 3 or 4 applied goto 2.. Else set $visited(v) \leftarrow t$ and stop.

Let $W := W_1 \cup W_2$ where $W_1 := \{u \in \hat{B} \mid visited(u) = t\}$ and $W_2 := \{u \in O \mid N^*(u) = \{x_1, x_2\} \ \& \ visited(x_i) = t \ (1 \leq i \leq 2)\}$. W_1 comprises the visited vertices in \hat{B} .

Proposition 8.3.4: Let $u, v \in W_1$

1. If there is a solution P respecting the current annotation such that $v \in P$ then $u \notin P$.
2. If there is a solution P respecting the current annotation such that $v \notin P$ then, w.l.o.g., $u \notin P$.

Proof. The vertex set W induces a path or a cycle in G containing at least two vertices from \hat{B} (as long $|V(G)| > 2$). Either v has a blank neighbor or and due to Lemma 8.3.3.2b it has a trigger as neighbor. Now observe that any vertex in W_1 is equally likely to be set active: Once there is an active vertex from W_1 the whole vertex-set W will be observed due to **OR2**. Also any additional trigger $t' \notin W_2$ which is a neighbor of some $v' \in W_1$ only depends on v' being observed.

Considering the branch $s(v) \leftarrow (a)$ due to exhaustively applying **OR2** for any $v' \in W$ we have $N(v') \subseteq O$ afterwards. Hence W will drop out of \hat{B} but will also not be included in \hat{O} due to **Isolated**. Thus there is a reduction of $|W| \geq 2$.

In case $s(v) \leftarrow (i)$ due to applying **Blank2** and **Trig2** we have that $W \subset I \setminus \hat{I}$ (Lemma 8.3.3.2a/2b) and thus a reduction of $|W|$. \square

Summing up we have a (2, 2) branch which we upper bound by $\mathcal{O}^*(1.415^n)$. This enables us to conclude

Theorem 8.3.5: POWER DOMINATING SET can be solved in time $\mathcal{O}^*(1.7548^n)$.

We like to comment that Algorithm 12 achieves a run time of $\mathcal{O}^*(1.6212^n)$ on cubic graphs. This can be seen by modifying the general analysis. Simply choose $\beta = 0.8126$ and $\gamma = 0.3286$ and skip the part where $d^*(v) \geq 4$.

\mathcal{NP} -hardness of the case $d^*(v) \leq 2$. Note that the instances occurring at this point of Algorithm 12 are still \mathcal{NP} -hard to solve. There is a simple reduction from CUBIC PDS. For any vertex v with $N(v) = \{n_{v1}, n_{v2}, n_{v3}\}$, create a triangle $C_v = j_{v1}, j_{v2}, j_{v3}$. If $\{u, v\} \in E$ we find $1 \leq b, c \leq 3$ with $u = n_{vb}$ and $v = n_{uc}$. Then connect free vertices $j_{uc} \in C_u$ and $j_{vb} \in C_v$ with an inactive trigger, i.e. introduce an observed and inactive vertex tr and edges $\{j_{ub}, tr\}, \{j_{vc}, tr\}$. This leads to a graph \mathbb{G} . Now the following is true for a PDS P for \mathbb{G} : A triangle C_u is observed iff a) $V(C_u) \cap P \neq \emptyset$ or b) Two vertices of C_u , w.l.o.g. j_{v1}, j_{v2} , are indirectly observed from vertices outside C_u and j_{v3} is indirectly observed by j_{v2} .

With this property it is easy to show that G has a size k PDS iff \mathbb{G} has one. So in step 10 of Algorithm 12, we have no alternative to continue with the branching as we can not expect a polynomial time algorithm for this case.

8.4. Conclusion and Further Perspectives

Speed-Up With Exponential Space. If we do not restrict ourselves to polynomial space consumption we can achieve a speed-up for Algorithm 12 using the memoization technique (see also [68]). We precompute optimal solutions for all vertex induced subgraphs G' such that $\varphi(G') \leq \omega \cdot n$ where $\omega = 0.059$. For such a graph we have $|V(G')| \leq c \cdot \omega \cdot n$ where $c = \max\{1/\gamma, 1/\beta\} = 2.0475$. For each subset $S \subseteq V$ with $|S| \leq c\omega n$ and for all $\Theta \subseteq S$ we compute the entries of the following table:

$$T[S, \Theta] = \min \left\{ |P'| : \begin{array}{l} P' \text{ is a PDS for } G[S] \text{ s.t. } \forall v \in \Theta : s'(v) = (o) \\ \& \forall v \in S : s(v) = (b) \& \forall v \in S \setminus \Theta : s'(v) = (u) \end{array} \right\}$$

So, Θ is supposed to be an already observed subset of vertices in $G[S]$. As $|S| \leq c\omega n$ there are $\mathcal{O}^*\left(\binom{n}{c\omega n}\right)$ subgraphs to be inspected. For any such subgraph there are $2^{c\omega n}$ possibilities to choose Θ . Thus, the size of the table is $\mathcal{O}^*\left(\binom{n}{c\omega n} \cdot 2^{c\omega n}\right) \subseteq \mathcal{O}^*(1.5719^n)$. By solving each of these instances by using the algorithm of [17] which enumerates all vertex subsets of S of maximum size $\lceil c\omega n/3 \rceil$, we spend $1.89^{c\omega n}$ steps for any induced subgraph $G[S]$ with predetermined observation pattern Θ . Thus we need $\mathcal{O}^*\left(\binom{n}{c\omega n} 3.78^{c\omega n}\right) \in \mathcal{O}^*(1.6975^n)$ steps for building up a the table T .

Let $R_G = V(G) \setminus \{v \in O \mid N[v] \subseteq O\}$. Once we arrived at a graph G with $|R_G| \leq c\omega n$ it follows that $\varphi(G[R_G]) \leq \omega n$. Thus, in Algorithm 12, we can look up the rest of the solution by inspecting the table entry which is determined by R_G and $R_G \cap O$, i.e. $T[R_G, R_G \cap O]$. Thus Algorithm 12 will run in $\mathcal{O}^*(1.7548^{(1-\omega)\varphi(G)}) \subseteq \mathcal{O}^*(1.7548^{(1-\omega)n}) \subseteq \mathcal{O}^*(1.6975^n)$. It is important to notice that we ignored the fact that there might be active and inactive vertices in $G[R_G]$. The correctness follows from the fact that the state of observation of $G[V(G) \setminus R_G]$ is independent of how $G[R_G]$ is observed. Also any solution for $G[R_G]$ which ignores the labels active and inactive cannot be worse than one that does not.

By choosing $\omega = 0.0337$. the same algorithm solves CUBIC PDS in $\mathcal{O}^*(1.5954^n)$ steps using $\mathcal{O}(1.49456^n)$ space.

Theorem 8.4.1: By allowing an exponential amount of space POWER DOMINATING SET can be solved in time $\mathcal{O}^*(1.6975^n)$ and on cubic graphs in time $\mathcal{O}^*(1.5954^n)$.

Notice that this type of speed-up relies on the fact that no branching or reduction rule ever changes the (underlying) graph itself, but rather, the existing graph is annotated. This property is also important when designing improved algorithms with the help of reference search trees, since it might be tricky to argue to find a solution not worse than the ones to be expected in a particular branch of a search tree somewhere else in the tree, when the instance is (seemingly) completely changed.

Also note that the table T has also to take into account the measure. It is not sufficient to build up T for all $S \subseteq V$ such that $|S| \leq \lambda n$ for some constant λ . A set of λn vertices will generally have a weight less than λn under the measure φ . Put it another way: Algorithm 12 is analyzed with respect to φ . If the rest graph has weight z under φ then it usually will have drastically more vertices than z . This is caused by the fact that the weight of each vertex is often less than one.

As the run time of Algorithm 12 is measured in terms of φ , T should have an entry for every $S \subseteq V$ with $\varphi(G[S]) \leq \lambda n$. Then as soon as the graph in the course of Algorithm 12 has weight less than λn under φ we can look up the solution in T . Hence, the runtime is $\mathcal{O}^*(1.7548^{(1-\lambda)\varphi(G)})$ as $\varphi(G) \leq \lambda n$.

Résumé. We designed an exact algorithm for POWER DOMINATING SET consuming $\mathcal{O}^*(1.7548^n)$ time. To achieve this we made intensive use of the concept of a reference search tree. This means that we were able to cut off branches by referring to arbitrary locations in the search tree where one can find equivalent solutions. Maybe the term search-DAG expresses this property also quite well. We proved the correctness of a reduction rule whose application was critical for the run time. We expect that we can exploit reference search trees further by designing exact algorithms for non-local problems or improving existent ones (e.g., CONNECTED DOMINATING SET/CONNECTED VERTEX COVER or MAX INTERNAL SPANNING TREE). For this kind of problems it seems we are not allowed to delete vertices due to selecting vertices into the solution or not. We rather have to label them. Many algorithms come to decisions by respecting them. We rather try to make use of them. Let x be a labeled vertex not selected into the solution and y an unlabeled vertex. Suppose by re-labeling x (taking x into the solution) and excluding y from the solution we have a solution which is no worse to the possibility of taking y into the solution. Then we can skip this last possibility by inserting a reference. We imagine that this arguing is also possible when several vertices are involved.

Chapter 9.

Exact Algorithms for Maximum Acyclic Subgraph on a Superclass of Cubic Graphs.

9.1. Introduction and Definitions

9.1.1. Our problems

We consider the problem of finding large acyclic subgraphs in directed graphs. More formally, we consider the following problem:

MAXIMUM ACYCLIC ARC-INDUCED SUBGRAPH MAAS

Given: a directed graph $G(V, A)$, and the parameter k .

We ask: Is there a subset $A' \subseteq A$, with $|A'| \geq k$, which is acyclic.

The Vertex-induced problem version of MAAS is called MAXIMUM ACYCLIC VERTEX-INDUCED SUBGRAPH MAVS

9.1.2. Motivation.

Both the FEEDBACK VERTEX SET and the FEEDBACK ARC SET problems were on the list of 21 problems that was presented by R. M. Karp [97] in 1972, exhibiting the first \mathcal{NP} -complete problems ever. These problems have numerous applications [59], ranging from program verification, VLSI and other network applications to graph drawing, where in particular the re-orientation of arcs in the first phase of the Sugiyama approach to hierarchical layered graph drawing is equivalent to DIRECTED FEEDBACK ARC SET (DFAS), see [6, 147].

Mostly, we focus on a class of graphs that, to our knowledge, has not been previously described in the literature. Let us call a directed graph $G = (V, E)$ $(1, \ell)$ -graph if, for each vertex $v \in V$, its indegree $d^-(v)$ obeys $d^-(v) \leq 1$ or its outdegree $d^+(v)$ satisfies $d^+(v) \leq 1$ (i.e., $\forall v \in V : \min\{d^-(v), d^+(v)\} \leq 1$). In particular, graphs of maximum degree three are $(1, \ell)$ -graphs. Notice that MAAS, restricted to cubic graphs, is still \mathcal{NP} -complete.

For some applications from graph drawing (e.g., laying out “binary decision diagrams” where vertices correspond to yes/no decisions) even the latter restriction is not so severe

at all. Having a closer look at the famous paper of I. Nassi and B. Shneiderman [122] where they introduce structograms to aid structured programming (and restricting the use of GOTOs), it can be seen that the resulting class of flowchart graphs is that of $(1, \ell)$ -graphs. Namely, a certain position of the structogram is either branching out (resulting from the very start of if-then-else statements, case statements, loops) or it is collecting several branches of the program (basically, at the end of the aforementioned statements). Here, we can derive a further application of our algorithms: for debugging purposes, a programmer might want to output values of variables etc., but she wants to place these watchdog program fragments at only a few number of places, still being able to survey the run of the program in each possible case. This means, in particular, that one fragment should be inserted in each loop of the program, which corresponds to identifying small feedback arc sets (or feedback vertex sets, depending on how we model the watchdogs) within the flowchart graph of the program code.

Cubic graphs also have been discussed in relation to approximation algorithms: A. Newman [124] showed a factor $\frac{12}{11}$ -approximation.

This largely improves on the general situation, where only a factor of 2 is known [6]. We point out that finding a minimum feedback arc set (in general directed graphs) is known to possess a factor $\log n \log \log n$ -approximation, see [59], and hence shows an approximability behavior much worse than MAAS.

9.1.3. Discussion of related results.

MAAS on general directed graphs can be solved in time and space $\mathcal{O}^*(2^k)$ and $\mathcal{O}^*(2^n)$, shown by V. Raman and S. Saurabh in [138], with n being the number of vertices. The same authors show a run time of $\mathcal{O}^*(4^k)$ for MAAS requiring only polynomial space in [137]. J. Chen, Y. Liu, S. Lu., B. O'Sullivan and I. Razgon [23] showed that DIRECTED FEEDBACK VERTEX SET $\in \mathcal{FPT}$. In contrast to MAAS, it still admits a fairly vast run time of $\mathcal{O}^*(4^k k!)$. This easily translates to a parameterized algorithm for DFAS. As an aside, let us mention that this makes finding feedback arc sets one of the few known natural examples where the problem and its parameterized dual are in \mathcal{FPT} . This is not true for the vertex case: It is not hard to see that MAVS on general graphs is W[1]-complete.¹

Likewise, I. Razgon [140] provided an exact (non-parameterized) $\mathcal{O}^*(1.9977^n)$ -algorithm for FEEDBACK VERTEX SET (FVS), which translates to a DFAS-algorithm with the same base, but measured in m (where m , as usual, denotes the number of arcs in the given graph instance).

The complexity picture changes when one considers undirected graphs. The task of removing a minimum number of edges to obtain an acyclic graph can be accomplished in polynomial time, basically by finding a spanning forest. The task of removing a

¹Hardness follows by observing that $I \subseteq V$ is an independent set in a given undirected graph $G = (V, E)$ if and only if I induces an acyclic graph in the digraph $D = (V, A)$, where A contains both (u, v) and (v, u) for each edge $\{u, v\} \in E$. For membership in W[1], construct a nondeterministic TM that first guesses a vertex set and then verifies its acyclicity, all in time $f(k)$, where k gives the size of the vertex set, see [19].

minimum number of vertices to obtain an acyclic graph is (again) \mathcal{NP} -complete, but can be approximated to a factor of two, see V. Bafna *et al.* [4], and is known to be solvable in $\mathcal{O}^*(5^k)$ with J. Chen *et al.* [20] being the leading party in a run time race. Also, exact algorithms have been derived for this problem by F. V. Fomin *et al.* [63].

9.1.4. Our contributions.

As we have described, feedback set problems are quite hard on directed graphs from a parameterized perspective. Our main technical contribution is to derive a parameterized $\mathcal{O}^*(1.2471^k)$ -time and polynomial space algorithm for MAAS on $(1, \ell)$ -graphs. On cubic graphs the run time reduces to $\mathcal{O}^*(1.201^k)$. We also derive an exact algorithm for MAAS on $(1, \ell)$ -graphs and as a by-product another for DIRECTED FEEDBACK VERTEX SET on cubic graphs with run times $\mathcal{O}^*(1.2133^m)$ and $\mathcal{O}^*(1.282^n)$, respectively.

The algorithms that we present for these special graph classes possess quite a simple overall structure, but the analysis is quite intricate and seems to offer a novel way of amortized search tree analysis that might be applicable in other situations in parameterized algorithmics, as well. We therefore use the notion of reference search trees introduced in section 1.4. It is also one of the fairly rare applications of the *Measure&Conquer* paradigm [74] in parameterized algorithmics. Moreover, as testified in [87], there are only quite few problems on directed graphs ever approached from the viewpoint of parameterized complexity; the present problem adds to this list.

9.1.5. Fixing terminology.

A subset S of $V(G)$ ($A(G)$, resp.) is a directed Feedback Vertex Set (directed Feedback Arc Set, resp.), if every directed cycle C of G contains at least one vertex (arc, resp) of S , i.e., $V(C) \cap S \neq \emptyset$ ($A(C) \cap S \neq \emptyset$, resp.). We call the complement $V(G) \setminus S$ ($A(G) \setminus S$, resp.) of S an *acyclic vertex subset* (*acyclic arc subset*, resp.) of G because it induces an acyclic subgraph of G . A maximum acyclic vertex subset (maximum acyclic arc subset, resp.) is the complement of a minimum directed feedback vertex set (minimum directed feedback arc set, resp.).

We call an arc (u, v) a *fork* if $d^+(v) \geq 2$ (but $d^-(v) = 1$) and a *join* if $d^-(u) \geq 2$ (but $d^+(u) = 1$). With \mathcal{MAS} , we refer to a set of arcs of the original graph which is acyclic and is viewed as a partial solution.

9.2. The Algorithm

9.2.1. Preprocessing

Firstly, we can assume that our instance $G(V, A)$ forms a strongly connected component. Every arc not in such a component can be taken into a solution, and two solutions of two such components can be simply joined.

In [59, 124], a set of preprocessing rules is already mentioned:

Pre-1: For every $v \in V$ with $d^-(v) = 0$ or $d^+(v) = 0$, delete v and $N_A(v)$, take $N_A(v)$ into \mathcal{MAS} and decrement k by $|N_A(v)|$.

Pre-2: For every $v \in V$ with $N_A(v) = \{(i, v), (v, o)\}$, $v \neq i$ and $v \neq o$, delete v and $N_A(v)$ and introduce a new arc (i, o) . Decrease k by one.

These preprocessing rules will be carried out exhaustively. Afterwards, the resulting graph has no vertices of degree less than three.

Definition 9.2.1. An arc g is an α -arc if it is a fork and a join and if it is contained in at least two cycles.

Because G is strongly connected, there are no α -arcs which are in no cycle. For every arc g which is a fork and a join one can determine if it is an α -arc the following way. Find the smallest cycle C_g which contains g via BFS. If g is contained in a second cycle C'_g , then there is an arc $a \in C_g$ with $a \notin C'_g$. So for all $a \in C_g$, remove a and restart BFS, possibly finding a second cycle.

We need the next lemma, which is a sharpened version of [124, Lemma 2.1] and follows the same lines of reasoning.

Lemma 9.2.1: Any two non-arc-disjoint cycles in a $(1, \ell)$ -graph with minimum degree at least 3 share an α -arc.

Proof. Suppose cycles C_1 and C_2 share a path $P = u_1 \dots u_\ell$. We show that at least one arc of P must be an α -arc. Suppose (for the purpose of contradiction) that every arc (u_i, u_{i+1}) with $1 \leq i \leq \ell - 1$ is not an α -arc. By induction on i , $1 \leq i \leq \ell - 1$, we show that any (u_i, u_{i+1}) is a join. For $i = 1$ this is clear as C_1 and C_2 both enter P at u_1 . This means that two arcs point towards u_1 and by the $(1, \ell)$ -property we have that (u_1, u_2) is a join. Now suppose the claim holds for any (u_j, u_{j+1}) with $j \leq i$. Consider the vertex u_{i+1} . Because we have $d(u_{i+1}) \geq 3$, there must be another arc a incident to u_{i+1} with $a \notin A(P)$. As (u_i, u_{i+1}) is a join but not an α -arc, it follows that $a = (v, u_{i+1})$ for some $v \in V$. Hence, (u_{i+1}, u_{i+2}) is a join. Especially $(u_{\ell-1}, u_\ell)$ is a join. But on the other hand it must also be a fork. This is due to the two arcs a_1, a_2 leaving u_ℓ ($init(a_1) = init(a_2) = u_\ell$) such that $a_1 \in A(C_1 \setminus C_2)$ and $a_2 \in A(C_2 \setminus C_1)$. Thus, $(u_{\ell-1}, u_\ell)$ is an α -arc which contradicts our assumption. \square

We partition A in A_α containing all α -arcs and $A_{\bar{\alpha}} := A \setminus A_\alpha$. By Lemma 9.2.1, the cycles in $G[A_{\bar{\alpha}}]$ must be arc-disjoint. This justifies the next preprocessing rule.

Pre-3 Let C be a cycle contained in $G[A_{\bar{\alpha}}]$. Pick an arbitrary $a \in C$, adjoin $C \setminus \{a\}$ to \mathcal{MAS} and decrease k by $|C| - 1$. In G , delete the arc set of C .

After exhaustively applying the preprocessing rules in the given order up to the point where non of them applies, every cycle has an α -arc. Observe also that **Pre-3** will remove any loop.

9.2.1.1. A Simple Algorithm

For $v \in V$ with $A^-(v) = \{a_1, \dots, a_s\}$ ($A^-(v) = \{c\}$, resp.) and $A^+(v) = \{c\}$ ($A^+(v) = \{a_1, \dots, a_s\}$), it is always better to delete c than one of a_1, \dots, a_s . Therefore, we adjoin a_1, \dots, a_s to \mathcal{MAS} , adjusting k accordingly. Having applied this rule on every vertex, we adjoined the set $\{(u, v) \in A \mid \min\{d^+(u), d^-(v)\} = 1\}$ to \mathcal{MAS} and therefore the next lemma is valid.

Lemma 9.2.2: If for $(u, v) \in A$ we have $\min\{d^+(u), d^-(v)\} = 1$ then, w.l.o.g, we can assume $(u, v) \in \mathcal{MAS}$.

So, the next task is to find $S \subseteq A_\alpha$ with $|\mathcal{MAS} \cup S| \geq k$ so that $G[\mathcal{MAS} \cup S]$ is acyclic. We have to branch on the α -arcs, deciding whether we take them into \mathcal{MAS} or if we delete them. The preprocessing rules give us another simple brute-force algorithm for MAAS: Within the preprocessed graph with m arcs, there could be at most $m/3$ arcs that are α -arcs. It is obviously sufficient to test all possible $2^{m/3} \leq 1.26^m$ many possibilities of choosing α -arcs to be put into the (potential) feedback arc set.

We also can use the 2-approximation of [6] to get a kernel of size $2k$. The solution $A_S \subseteq A$ which is returned by this approximation algorithm is analyzed with respect to the input A . This means we have derived that $\frac{|A|}{|A_S|} \leq 2$. If $|A_S| \geq k$, we stop, and otherwise we know that $|A| \leq 2k$. In the latter case, we apply preprocessing and cycle again through all subsets $S \subseteq A_\alpha$. Now due to $m \leq 2k$, this takes $\mathcal{O}^*(1.5875^k)$ steps. In the remainder of this chapter, we are going to improve on this algorithm by introducing reduction rules that can deal with appropriately defined weighted instances that are produced by branching and that are tailored to work with the MEASURE&CONQUER approach.

9.2.2. Reduction Rules

9.2.2.1. The Overall Strategy

There is a set of reduction rules from [124] for cubic graphs which also work for $(1, \ell)$ -graphs. We want to use the power of these reduction rules also in our algorithm. For the purpose of measuring the complexity of the algorithm, we will deal with two parameters k and k' , where k measures the size of the partial solution and k' will be used for purposes of run-time estimation: We do not account the arcs in $A_{\bar{\alpha}}$ immediately into k' . For every branching on an α -arc, we count only a portion of them into k' .

More precisely, upon first seeing an arc $b \in A_{\bar{\alpha}}$ within the neighborhood $AN(g)$ of an α -arc g on which we branch, we will count b only by an amount of ω , where $0 < \omega < 0.5$ will be determined later. So, we will have two weighting functions w_k and $w_{k'}$ for k and k' with $w_k(a) \in \{0, 1\}$ and $w_{k'}(a) \in \{0, (1 - \omega), 1\}$ with $a \in A$, indicating each how much of the arc has not been counted into k , or k' respectively, yet. In the very beginning, we have $w_k(a) = w_{k'}(a) = 1$ for all $a \in A$ and in the course of the algorithm $w_k(a) \leq w_{k'}(a)$. For a set $A' \subseteq A$, we define $w_{k'}(A') := \sum_{a' \in A'} w_{k'}(a')$ and $w_k(A')$ accordingly.

Observe that for $a \in A$ we have $a \in \mathcal{MAS}$ iff $w_k(a) = 0$. Notice that the arc set A may change due to reduction rules; more specifically, some arcs will be deleted. However,

\mathcal{MAS} is also containing possibly deleted arcs, i.e., it is always referring to the originally given graph instance.

α -arcs which we take into \mathcal{MAS} will be called *red*. However, we will describe situations with the reduction rules where the label *red* could be also carried by non- α -arcs, indicating the former existence of a red α -arc “at that position.” Therefore, the label *red* of some arc will not get lost, although that arc might turn from an α -arc to a non- α -arc (for example, by applying the very first reduction rule from below). The very last sentence of Rule **RR-3** makes names of α -arcs dominant in the following sense: Every α -arc in the original graph has a specific name. In the course of the algorithm, the arc need no longer be an α -arc (for example due to arc deletions) but will still carry the name of the original α -arc. This is assured by **RR-3** as if the arc which is contracted was an α -arc, then the remaining arc will carry its name (even though it need not to be necessarily an α -arc). We say it carries an α -name. The very same also holds for red arcs. They necessarily carry α -names because we will only branch on α -arcs.

9.2.2.2. Reduction Rules for Weighted Arcs

The set of reduction rules from [124] must be adapted and modified to deal with weighted arcs. In addition, we define a predicate *contractible* (that can be tested in linear time) for all $a \in A$ as follows.

$$contractible(a) = \begin{cases} 0 & : w_k(a) = 1, \exists \text{ cycle } C \text{ with } a \in C \text{ and } w_k(C \setminus \{a\}) = 0 \\ 1 & : \text{else} \end{cases}$$

The meaning of this predicate is the following: if $contractible(a) = 0$, then a is the only remaining arc of some cycle, which is not already determined to be put into \mathcal{MAS} . Thus, a has to be deleted. To compute $contractible(a)$ we simply look for a cycle in $G[(\mathcal{MAS} \cap A(G)) \cup \{a\}]$.

In the following, **RR-(i-1)** is always carried out exhaustively before **RR-i**, cf. the procedure `Reduce()` in Fig. 9.1(b). Moreover, for simplicity we always indicate the handling of w_k and \mathcal{MAS} , although in many cases the corresponding deduction has already been performed, for example, because the corresponding arcs refer to non- α -arcs of the original instance. However, as long as only arc deletions are involved, we refrain from mentioning that A_α and $A_{\bar{\alpha}}$ might need updating with each rule.

RR-1 For $v \in V$ with $d^-(v) = 0$ or $d^+(v) = 0$, take $N_A(v)$ into \mathcal{MAS} , delete v and $N_A(v)$ and decrease k by $w_k(N_A(v))$ and k' by $w_{k'}(N_A(v))$.

RR-2 If for $g \in A$, we have $contractible(g) = 0$, then delete g .

RR-3 For $v \in V$ with $N_A(v) = \{a, b\}$ let $z = \arg \max\{w_{k'}(a), w_{k'}(b)\}$ and $y \in N_A(v) \setminus \{z\}$. Contract y , decrement k by $w_k(y)$, k' by $w_{k'}(y)$. If y was red, then z becomes red. Modify A_α and $A_{\bar{\alpha}}$ accordingly, i.e., delete y from $A_{\bar{\alpha}}$ and move z from $A_{\bar{\alpha}}$ to A_α if z has become an α -arc by this contraction. Finally, if z has become red only because y had been red or if y carried an α -name and z not (or both), rename z as y (and make according modifications to A_α , $A_{\bar{\alpha}}$ and \mathcal{MAS} if necessary).

We point out that, due to **RR-3**, also non- α -arcs may become red. But it is still true for a α -arc a that $a \in \mathcal{MAS}$ iff a is red. Further notice that, due to **RR-2**, the *contractible* predicate is always true (i.e., one) for subsequent reduction rules and need not be tested in the following.

RR-4 Let C be a cycle contained in $G[A_{\bar{\alpha}}]$. Pick an arbitrary $a \in C$ with $w_k(a) = 1$, adjoin $C \setminus \{a\}$ to \mathcal{MAS} and decrease k by $w_k(C \setminus \{a\})$ and k' by $w_{k'}(C \setminus \{a\})$. Delete the arc set of C in G .

RR-5 If $a, b \in A$ form an undirected 2-cycle then let $z = \arg \min\{w_{k'}(a), w_{k'}(b)\}$, decrease k by $w_k(\{a, b\})$, k' by $w_{k'}(z)$, take both arcs from the undirected 2-cycle into \mathcal{MAS} and delete z .

RR-6 Having $C = \{(u, v), (v, w), (u, w)\} \subseteq A$ (an undirected 3-cycle), decrease k by $w_k(C)$, k' by $w_{k'}((u, w))$, take C into \mathcal{MAS} and delete (u, w) .

RR-7 Delete the single α -arc g in any remaining necessarily directed 2- or 3-cycle.

In Section 9.3.1 we will show the correctness of the reduction rules. This deferral is due to the fact that the correctness of **RR-3** is dependent on the branching strategy, in particular when it comes to setting $w_k(z) = 0$ on (new) red α -arcs z .

9.2.3. The Concrete Algorithm

We now are putting together the reduction rules and the branching strategy. The obtained algorithm is Algorithm 13. In particular, notice that when YES is returned in one leaf of the search tree, then the set variable \mathcal{MAS} would contain the corresponding solution in terms of the arcs of the original input graph. Let $A_\alpha^U := \{a \in A_\alpha \mid a \text{ is non-red}\}$.

9.3. The Analysis

In this section we want to show that Algorithm 13 traverses a reference search tree (see chapter 1.4). Algorithm 13 simply branches on α -arcs g of G . Either it deletes g or it is taken into \mathcal{MAS} . In the second case g will be called red. Also, we can rely on the fact that every time we meet a red α -arc that there has been a branch on it. After the branching, the reduction rules will be exhaustively carried out in the subsequent recursive call.

MAXIMUM ACYCLIC ARC-INDUCED SUBGRAPH can be modeled as a combinatorial maximization problem. In the following description and in the rest of the chapter we are referring to the instance created by the exhaustive application of the preprocessing rules. The universe \mathcal{U} is the set A_α which comprises all α -arcs in G . Let $D(U, T)$ be the out-tree build up by Algorithm 13. The *label*-function is induced by the current partial solution in some node u of the search tree. Hence, any such u will be mapped to a solvec sv_u . The vector sv_u has length $|A_\alpha|$. For $g \in A_\alpha$, we find the following situations:

Algorithm 13 A parameterized algorithm for MAXIMUM ACYCLIC ARC-INDUCED SUBGRAPH on $(1, \ell)$ -graphs

- 1: Apply the preprocessing rules exhaustively.
- 2: $\mathcal{MAS} \leftarrow A_{\bar{\alpha}}$, $k' \leftarrow k$, $k \leftarrow k - w_k(A_{\bar{\alpha}})$, $w_k(A_{\bar{\alpha}}) \leftarrow 0$
- 3: Sol3MAS($\mathcal{MAS}, G(V, A), k, k', w_{k'}, w_k$)

Procedure: Sol3MAS($\mathcal{MAS}, G(V, A), k, k', w_k, w_{k'}$):

- 1: ($\mathcal{MAS}, G(V, A), k, k', w_k, w_{k'}$) \leftarrow Reduce($\mathcal{MAS}, G(V, A), k, k', w_k, w_{k'}$)
 - 2: **if** $k \leq 0$ **then**
 - 3: **return** YES
 - 4: **else if** there is a component C with at most 9 arcs **then**
 - 5: Test all possible solutions for C .
 - 6: **else if** there is an α -arc $g \in A_{\alpha}^U$ **then**
 - 7: **if not** Sol3MAS($\mathcal{MAS}, G[A \setminus \{g\}], k, k', w_k, w_{k'}$) **then**
 - 8: $k \leftarrow k - 1$, $k' \leftarrow k' - w_{k'}(g)$, $w_k(g) \leftarrow w_{k'}(g) \leftarrow 0$, $\mathcal{MAS} \leftarrow \mathcal{MAS} \cup AN(g) \cup \{g\}$.
 - 9: **for all** $a \in AN(g)$ **do**
 - 10: Adjust $w_{k'}$, see Figure 9.1(a).
 - 11: **end for**
 - 12: **return** Sol3MAS($\mathcal{MAS}, G(V, A), k, k', w_k, w_{k'}$)
 - 13: **else**
 - 14: **return** YES
 - 15: **end if**
 - 16: **else**
 - 17: **return** NO
 - 18: **end if**
-

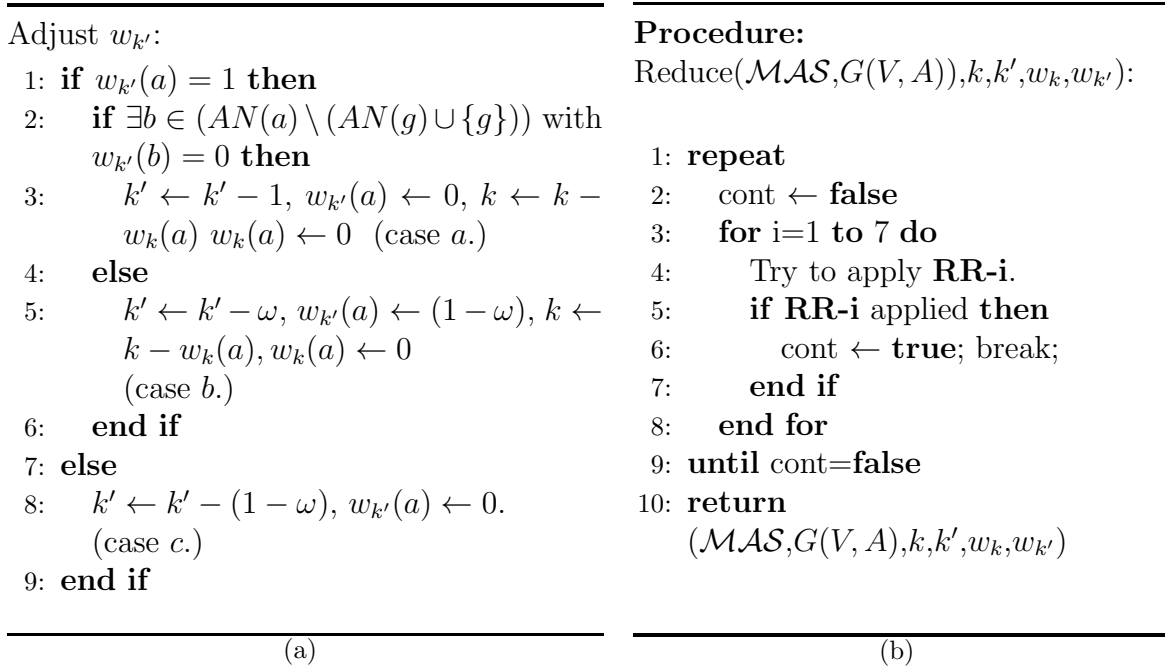


Figure 9.1.: 9.1(a): In case $a.$, we set $w_{k'}(a) = 0$, because there will not be any other neighboring non-red α -arc of a . In case $b.$, this might not be the case, so we count only a portion of ω . In case $c.$, we will prove that $w_{k'}(a) = (1 - \omega)$ and that there will be no other non-red neighboring α -arc of a , see Theorem 9.3.4.5.

1. If $sv_u[g] = 1$, then $g \in \mathcal{MAS}$ (i.e., g is red).
2. If $sv_u[g] = 0$, then $g \notin \mathcal{MAS}$ (i.e., g has been deleted).
3. If $sv_u[g] = \star$, then there had been no branching on g yet.

The value function will be $c(q) = |\{i \mid q[i] = 1\}|$ (i.e., the number of ones in the solvec q). The solution space \mathcal{S} are all subsets $S \subseteq A_\alpha$ which leave $G[A_\alpha \cup S]$ acyclic.

9.3.1. Analyzing the Reduction Rules

We start with listing some simple properties of reduced instances.

Lemma 9.3.1: After the application of Reduce(), see Figure 9.1(b), we are left with

1. a $(1, \ell)$ -graph
2. such that no cycle is without an α -arc,
3. no directed or undirected 2- or 3-cycle remains and
4. no $v \in V$ with $d(v) = 2$ or $\min\{d^-(v), d^+(v)\} = 0$.

5. Every red arc carries an α -name.
6. Every α -arc created by **RR-3** carries an α -name.

Proof. 1. Since deleting arcs preserves the $(1, \ell)$ -property, the only critical reduction rule is **RR-3**. Here we contract an arc $a = (u, v)$ with $d^-(u) = 1$ or $d^+(u) = 1$. Both cases do not violate the $(1, \ell)$ -property.

2. Note that after the exhaustive application of **RR-4**, any cycle has at least one α -arc.
3. After the exhaustive application of **RR-5** and **RR-6** there are no undirected 2- or 3-cycles, respectively, left. Observe that any directed 2- or 3- cycle has at most one α -arc. Then by **RR-7** these α -arcs will be deleted.
4. Due to **RR-1** there is no $v \in V$ with $\min\{d^-(u), d^+(v)\} = 0$. By **RR-3** we have $d(u) \geq 3$.
5. When a red arc g is created by branching g is a α -arc. Due to the dominance of α -names in **RR-3** it keeps its name.
6. If $(u, v), (v, t)$ give rise to a α -arc (u, t) due to **RR-3**, then before an arc (j, v) or (v, j) must have been deleted before (as our instance is preprocessed). Hence, (u, v) or (v, t) was an α -arc and by this (u, t) will carry a α -name. \square

Lemma 9.3.2: Alg. 13 either puts a selected α -arc g into \mathcal{MAS} , or it deletes g . If arcs are deleted, reduction rules can be triggered in the subsequent recursive call. This can be also due to triggering **RR-2** after putting g into \mathcal{MAS} . No reduction rule adds a non-red α -arc into the partial solution \mathcal{MAS} .

Proof. In the case where g is deleted we immediately trigger **RR-1**, as we always have a graph with minimum degree 3 in any node of the search tree, see Lemma 9.3.1.4. Subsequently, other reduction rules might be triggered. If there is a cycle C with $w_k(C \setminus g) = 1$ then there must be another arc $b \in C \setminus g$ with $w_k(b) = 1$. By taking g into \mathcal{MAS} we trigger **RR-2**, which deletes b . If such a cycle C does not occur we never trigger any reduction rule by taking g into \mathcal{MAS} . The last point can be seen with moderate effort by inspecting the reduction rules one after the other. \square

Lemma 9.3.3: The reduction rules are sound with respect to Alg. 13.

Proof. Basically, we show the soundness of the reduction rules by proving that they guarantee item 4(a) of Definition 1.4.1. We will not go in to detail in each case as this will follow implicitly from our arguing. An exception will be **RR-3**. This reduction rule will also make use of item 4(b) of the definition. It is the only reduction rule that will actually insert global references into the search tree.

RR-1 A vertex $v \in V$ with $d^-(v) = 0$ or $d^+(v) = 0$ cannot be entered and left by a cycle, so the incident arcs are not part of any cycle.

RR-2 If an α -arc is not contractible, it must be deleted because it is the only arc not in \mathcal{MAS} for some cycle, so **RR-2** is correct.

RR-3 For a vertex v with $N_A(v) = \{a, b\}$, we have to delete at most one arc from $\{a, b\}$ in order to cut a cycle. So, we can take one into \mathcal{MAS} and contract it. But additionally we must be sure if the arc we want to contract is not the last remaining arc on a cycle, which is not in \mathcal{MAS} . This check has already been done by **RR-2**.

Notice that our rule **RR-3** differs from the similar one in [124] by the fact that red arcs and α -names are dominant. It is possible that we create a new α -arc (w, v) by this rule, (w, v) being red. The arc (w, v) could evolve out of two arcs (w, t) and (t, v) the first being red, the second carrying an α -name. This is justified by the following claim:

Claim: If an α -arc (w, v) was created by merging a red arc (w, t) and a non-red arc (t, v) , then, w.l.o.g., $(w, v) \in \mathcal{MAS}$ (i.e., (w, v) is red).

Proof. To prove the claim we now are going to show that Algorithm 13 traverses a reference search tree (see chapter 1.4) where the global references are inserted by **RR-3**. The arc (w, t) carries an α -name by Lemma 9.3.1.5. If (t, v) does not carry an α -name it is predetermined to be in \mathcal{MAS} and the claim is correct. Thus, both carry an α -name which can be found in \mathcal{U} . Let $P := wtv$ be a directed path.

In the present search node $N_1 \in V(D)$ of the search tree, just before the merging of (t, v) and (w, t) , w.l.o.g., we have $label(N_1) = (e_1, \dots, e_y, \star, \dots, \star)$. W.l.o.g., it is of the form where $e_i \in \{0, 1\}$ for $1 \leq i \leq y$. We assume that e_1 corresponds to (w, t) and e_{y+1} to (t, v) . If (w, v) would not be an α -arc in G , then we do not have to branch and the reduction rule is correct. But (w, v) is such a seemingly new α -arc. To destroy any cycle passing through P , we have to delete at most one arc of P . So, deleting (t, v) would be equivalent to deleting (w, t) , i.e., the solvecs $(e_1, \dots, e_y, 0, \star, \dots, \star)$ and $(\bar{e}_1, \dots, e_y, 1, \star, \dots, \star)$ are equivalent in the sense that any solution $(e_1, \dots, e_y, 0, e_{y+2}, \dots, e_\ell)$ can be replaced by $(\bar{e}_1, \dots, e_y, 1, e_{y+2}, \dots, e_\ell)$ and vice versa.

Now, if we follow the path from the present node N_1 of the search tree to the root, we find the node N' where (w, t) was set to be red, (i.e., $N' = d_{(w,t)}^{N_1}$ speaking in terms of chapter 1.4). By the last point of Lemma 9.3.2 we know that red arcs have been created by branching. Thus, $p(N')$ is a full node, i.e., there is $N_2 \in V(D)$ which is the right child of $p(N')$ in the search tree (i.e., $r(p(N)') = N_2$) which considers the deletion of (w, t) and a second child (namely N') considering the addition of (w, t) to \mathcal{MAS} . We have $label(N_2) = (\bar{e}_1, e_2, \dots, e_x, \star, \dots, \star)$ where $x \leq y$. Thus, we have $(\bar{e}_1, \dots, e_y, 1, \star, \dots, \star) \preceq label(N_2)$. Summarizing, we have that if there is some $\varphi_1 := (e_1, \dots, e_y, 0, e_{y+2}, \dots, e_\ell) \in \mathcal{S}$ then it follows that $\varphi_2 := (\bar{e}_1, \dots, e_y, 1, e_{y+2}, \dots, e_\ell) \in \mathcal{S}$, too. Also, $c(\varphi_1) = c(\varphi_2)$ holds. Consequently, as $label(\varphi_2) \preceq label(N_2)$, we do not have to consider the possibility

$(e_1, \dots, e_y, 0, \star, \dots, \star)$ in the current search tree node N_1 . The global reference inserted this way into D is (N_1, N_2) . By this the deletion of (t, v) has been neglected and only its addition to \mathcal{MAS} was considered.

Up to this point, we have shown items 1, 3, and 4 of Definition 1.4.1. We now prove that $D(V, T \cup R)$ is acyclic. Now consider the definition of a proper drawing of $D(V, T)$ (Definition 1.4.2). In this sense excluding an α -arc from the future solution (or equivalently adjoining it to \mathcal{MAS}) is making it red, including it means deleting it. Hence, we have $pos_x(N_1) < pos_x(N')$ and $pos_x(N') < pos_x(N_2)$. Therefore the global reference (N_1, N_2) is pointing from the left to the right in the drawing. This is true for any global reference as they are only inserted by **RR-3**. By Lemma 1.4.1 the overall structure $D(V, T \cup R)$ is acyclic and therefore **RR-3** is sound. \square

The proof of the claim was quite lengthy but is needed to analyze the interactions between the reduction rules and the branching strategy later on. It surely also applies in the case when we merge a red arc (t, v) with a non-red arc (w, t) such that (w, v) becomes an α -arc. The case where two red arcs are merged correctly creates a single red arc.

RR-4 By Lemma 9.2.1 the considered cycle contains no α -arc. Thus, any of its arcs is equally likely to be deleted.

RR-5 Let $u, v \in V$ be the endpoints of an undirected 2-cycle. W.l.o.g., there are arcs a_1, a_2 with $init(a_1) = init(a_2) = u$ and $ter(a_1) = ter(a_2) = v$. The arcs a_1, a_2 can not be α -arcs, so it is safe to take them into \mathcal{MAS} . For any arc set P we have: $P \cup \{a_1\}$ is a cycle iff $P \cup \{a_2\}$ is a cycle. This means we can, w.l.o.g., erase a_2 as long as any cycle C containing a_1 will be cut by an arc different from a_1 . This is assured because we put a_1 into \mathcal{MAS} .

RR-6 If we have $(u, v), (v, w), (u, w) \in A$, there must be also $(a, u), (w, b) \in A$ and w.l.o.g., $(v, c) \in A$, because of the absence of vertices of degree less than three. The arcs $(u, v), (v, w), (u, w)$ are not α -arcs, so their exclusion from \mathcal{MAS} must not be considered. For any arc set P we have: $P \cup \{(u, v), (v, w)\}$ is a cycle iff $P \cup \{(u, w)\}$ is a cycle. Thus, if we take care only of cycles passing through (u, v) and (v, w) we also cover those passing through (u, w) . This justifies the deletion of (u, w) similarly as in the previous item.

RR-7 Any directed cycle must contain an α -arc due to **RR-4**'s higher priority. Cycles of length 2 or 3 cannot contain more than one α -arc due to their structure. Thus, this unique α -arc must be deleted.

Note that **RR-2** and **RR-7** cut off branches in the reference search tree which do not provide any solution. Thus, any kind of reference can be omitted. **RR-4** inserts local references as only one possible arc deletion of the at most $|C|$ is considered. This local reference points downward. **RR-1** puts arcs into \mathcal{MAS} . Any solution which omits one of these arcs can be improved locally and hence can be skipped. **RR-5** and **RR-6** also

put arcs into \mathcal{MAS} . If such an arc (a, b) does not carry an α -name this can be done without any inserted reference. If (a, b) carries an α -name then due to skipping the possibility of its deletion a local reference can be inserted as it is no α -arc in the current graph. \square

9.3.2. Analyzing the Algorithm

Observe that the handling of the second parameter k' is only needed for the run-time analysis and could be avoided when implementing the algorithm. Thus, the branching structure of the Alg. 13 is quite simple, as expressed in the following:

9.3.2.1. Combinatorial Observations.

While running the algorithm we have $k \leq k'$. Now, substitute in line 2 of Sol3MAS of Algorithm 13 k by k' . If we run the algorithm, it will create a search tree $T_{k'}$. The search tree T_k of the original algorithm must be contained in $T_{k'}$, because $k \leq k'$. If $|T_{k'}| \leq c^{k'}$, then it follows that also $|T_k| \leq c^{k'} = c^k$, because in the very beginning, $k = k'$. So in the following, we will state the different recurrences derived from Algorithm 13 in terms of k' . For a good estimate, we have to calculate an optimal value for ω .

Theorem 9.3.4: In every node of the search tree, after applying Reduce(), we have

1. For all $a = (u, v) \in A_{\bar{\alpha}}$ with $w_{k'}(a) = (1 - \omega)$, there exists a red fork (u', u) or a red join (v, v') .
2. For all non-red $a = (u, v) \in A_{\bar{\alpha}}$ with $w_{k'}(a) = 0$, we find a red fork (u', u) and a red join (v, v') . We will also say that a is *protected* (by the red arcs).
3. For all red arcs $d = (u, v)$ with $w_{k'}(d) = 0$, if we have only non-red arcs in $N_A(u) \setminus \{d\}$ ($N_A(v) \setminus \{d\}$, resp.), then d is a join (d is a fork, resp.).
4. For each red arc $d = (u, v)$ with $w_{k'}(d) = 0$ that is not a join (fork, resp.), if there is at least one red arc in $N_A(u) \setminus \{d\}$ (in $N_A(v) \setminus \{d\}$, resp.), then there is a red fork (red join, resp.) in $G[N_A(u)]$ ($G[N_A(v)]$, resp.).
5. Let $g \in A_{\alpha}^U$ then for all $a \in AN(g)$, we have: $w_{k'}(a) > 0$.
6. For all $g \in A_{\alpha}^U$ we have $w_{k'}(g) = 1$.

Proof. We use induction on the depth of the reference search tree. Clearly, all claims are trivially true for the original graph instance, i.e., the root node. This is also easily verified for the situation obtained after a first exhaustive application of the reduction rules as no red arcs are produced.

Now consider the reference search tree which is built up. More generally, we can associate to each node with out-degree at least two in the reference search tree a reduced instance (so, no reduction rules apply). These are exactly the nodes where Alg. 13 was branching.

We are going to argue on these instances with our induction argument.

As induction hypothesis, we assume that the claim is true for all reference search tree nodes up to depth n . Let us discuss a certain reference search tree node s at depth $n+1$. Let $G = (V, A)$ be the graph instance associated with s . Let k and k' be the parameter values at node s . Let \bar{s} be the immediate predecessor node of s in the search tree. We will refer with $\bar{G} = (\bar{V}, \bar{A})$, \bar{k} , \bar{k}' to the corresponding instance and parameter values. Notice that each claim has the form $\forall a \in A : X(a) \implies Y(a)$. Here, X and Y express local situations affecting a . Therefore, we have to analyze how $X(a)$ could have been created by branching. According to Lemma 9.3.2, we have to discuss what happens (1) if a certain α -arc had been put into \mathcal{MAS} and (2) if reduction rules were triggered. As a third point, we must consider the possibility that $X(a)$ is true both in the currently observed reference search tree node s and in its predecessor, but that $Y(a)$ was possibly affected upon entering s .

Exemplary, we will give a very detailed proof of the very first assertion. The other parts can be similarly shown, so that we only indicate the basic steps of a complete formal proof.

1. Consider an arc $a = (u, v) \in A_{\bar{\alpha}}$ with $w_{k'}(a) = (1 - \omega)$. This situation could have been due to three reasons:

(A) In node \bar{s} , we branched at an arc $\bar{d} \in AN(\bar{a})$ with $w_{\bar{k}'}(\bar{a}) = 1$, see Figure 9.2(a) (where $d = (w, u)$). We consider here the case that \bar{d} is turned red. Namely, according to case *b.* of the procedure “Adjust” (see Figure 9.1(a)), $w_{k'}(a) = (1 - \omega)$. Since we only branch at α -arcs, \bar{d} is even both a fork and a join. As detailed in (B), \bar{a} could give rise to $a \in V$ by a sequence of **RR-3**-applications in possible combination with other rule applications, such that $w_{k'}(a) = (1 - \omega)$. As described in (C), \bar{d} will yield, as a red neighbor of \bar{a} , again by a (possibly empty) sequence of reduction rule applications, in particular of **RR-3**-applications, a fork or join that is neighbor of a in G as required. (B) In node \bar{s} , we branched at some arc c . We consider here the case that (possibly due to a following application of **RR-2**) some arc b is deleted (possibly $b = c$). This triggers some reduction rules. How could the situation have been created by reduction rule applications? The only possibility is to (eventually) use **RR-3**. All other reduction rule delete arcs.

In that case, there would have been two neighbored arcs a', a'' in \bar{G} with $\max\{w_{\bar{k}'}(a'), w_{\bar{k}'}(a'')\} = (1 - \omega)$. Hence, at least one of these arcs, say a' , actually carried the weight $(1 - \omega)$. In actual fact, there could have been a whole cascade of **RR-3**-applications along a path P in \bar{G} (P consists of a sequence of subsequently neighbored arcs from $\bar{A}_{\bar{\alpha}}$), eventually leading to a , but by an easy inductive argument one can see that there must have been some $\bar{a} \in \bar{A}_{\bar{\alpha}}$ within this cascade to which the induction hypothesis applies, so that we conclude that, in \bar{G} , \bar{a} has a neighboring arc \bar{d} that is a fork or a join. Since \bar{d} is a fork or a join, it cannot be neighbor to two arcs from the path P (as long as P and \bar{d} do not induce a cycle. Then **RR-3** will create a loop or a 2-cycle and **RR-4** or **RR-7** will delete \bar{d} . This is handled in (C)).

Therefore, w.l.o.g., \bar{a} is the first arc on P (without predecessors on P), and \bar{d} is a fork. After the sequence of **RR-3**-applications on P (possibly interrupted by reduction rules not affecting P), a has been created with \bar{d} as a neighboring red fork. We will show in (C) that the fork \bar{d} will eventually lead to a fork d that is neighbor of a in G .

(C) We consider the scenario that already in node \bar{s} , $w_{k'}(a) = (1 - \omega)$. By induction hypothesis, assume that (w.l.o.g.) $a = (u, v)$ has a neighbored red fork $\bar{d} = (u', u)$. If \bar{d} is deleted by using reduction rules, then u would have (intermediately) in-degree zero, so that **RR-1** triggers on a , contradicting our very scenario in G that we are discussing. Therefore, the local situation could only change by applications of **RR-3** involving \bar{d} . If those mergers refer to neighbors of \bar{d} via the tip of \bar{d} , then either a is directly deleted or merged with \bar{d} . Both possibilities would destroy the scenario we discuss, since a would disappear. Therefore, such mergers could be only via the tail of \bar{d} . Since \bar{d} is red, a merger with \bar{d} will be red, as well. Moreover, this merger would be also a fork. Again by an easy induction, one can conclude that the neighbor d of a in G that results from a sequence of mergers using rule **RR-3** on a path ending at \bar{d} in \bar{G} would be a red fork as required.

2.-4. We will actually prove points 2. through 4. by a parallel induction. To improve readability of our main argument, we refrain from giving all possible details how, in particular, the employment of **RR-3** may affect (but not drastically change) the situation. This means that we do not discuss the possibility that the premise of the statement(s) was true at some reference search tree node \bar{s} , but the conclusion might no longer hold due to intermediate applications of reduction rules.

2. How can $a = (u, v) \in A_{\bar{\alpha}}$ with $w_{k'}(a) = 0$ have been created?

Firstly, it could be due to a **RR-3**-contraction with a non-red arc t with $w_{k'}(t) = 0$. But then t was not protected, which is a contradiction to the induction hypothesis. Secondly, it could be due to branching on a neighboring α -arc b , say $b = (v, w)$ with b a join, in two different ways:

(1) either we branched at b at a point of time when $w_{k'}(a) = (1 - \omega)$ (case c . of Procedure “Adjust”, see Figure 9.1(a)), or (2) we branched at b when $w_{k'}(a) = 1$ (case a ., see Figure 9.1(a)). The case $w_{k'}(a) = 0$ is impossible as by induction a is protected.

In case (1), there must have been a red arc e incident to a by item 1 of our property list, see Figure 9.2(b). The arc e is not incident to v , since it must be a fork. Hence, $e = (y, u)$. This displays the two required red arcs (namely b and e) in this case. In case (2), a was created by case a . of Procedure “Adjust.” Obviously, b is red after branching. Since we have branched according to case a ., there is another arc h incident with a (but not with b) such that $w_{k'}(h) = 0$. There are four subcases to be considered:

(a) $h = (u, u')$ is not red, see Figure 9.2(c). By induction (item 2.), there must be a red fork arc (u'', u) . Hence, a is protected.

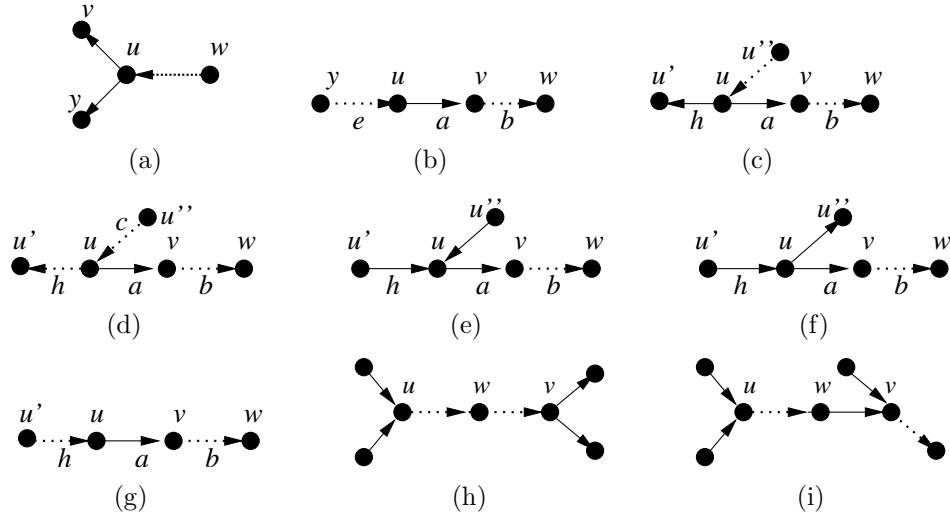


Figure 9.2.: Dotted lines indicate red arcs.

(b) $h = (u, u')$ is red, see Figure 9.2(d). Consider all other arcs incident to u . Since we are dealing with reduced instances and by the $(1, \ell)$ -property, there must be exactly one of the form $c = (u'', u)$, since otherwise u would be a source. Suppose h is the only red arc in $N_A(u)$. Then this contradicts item 3. Now, suppose c is not red. Then there is a red arc of the form (u, \bar{u}) . By item 4. c must be also red, a contradiction, Therefore, c is the red fork which protects a .

(c) $h = (u', u)$ is not red. All other arcs incident to u could be of the form (u'', u) , see Figure 9.2(e). Since h must be protected, by induction, a should be red, contradicting our assumption on a . Thus, all these arcs are of the form (u, u'') , see Figure 9.2(f). This contradicts item 2., since there is no red join protecting h .

(d) $h = (u', u)$ is red, see Figure 9.2(g). Suppose h is not a fork. Then, all other arcs incident with u beside a and h are of the form (\tilde{u}, u) . If none of them is red we have a contradiction concerning item 3. If one of them is red, then by item 4., a is red, which contradicts our assumptions. Hence, h is a fork, so that a will be protected.

3. How could d have been created? If it had been created by branching, then there are two cases: (1) d was put into \mathcal{MAS} ; (2) d was neighbor of an arc b which we put into \mathcal{MAS} . The deletion of b would result in a deletion of d .

In case (1), the claim is obviously true. In case (2), let, w.l.o.g., u be the common neighbor of b and d . After putting b into \mathcal{MAS} , there will be a red arc (namely b), incident to u , so that there could be only non-red arcs incident with v that have the claimed property by induction. Hence, the premise for d is not true with respect to u .

If d has been created by reduction rules, it must have been through **RR-3**. So, there have been (w.l.o.g.) two arcs (u, w) and (w, v) with $w_{k'}$ -weights zero. One of them must be red. W.l.o.g., assume that (u, w) is red. If (w, v) is also red, see

Figure 9.2(h), then the claim holds by induction. If (w, v) is not red, see Figure 9.2(i), then (w, v) must be protected due to item 2. Hence, the premise is falsified for d with respect to the vertex v .

4. We again discuss the possibilities that may create a red d with $w_{k'}(d) = 0$.
 If d was created by taking it into \mathcal{MAS} during branching, then d would be both fork and join in contrast to our assumptions.
 If we branch in the neighborhood of d , then the claim could be easily verified directly. Finally, d could be obtained from merging two arcs $e = (u, w)$, $f = (w, v)$ with $w_{k'}(e) = w_{k'}(f) = 0$. If both e and f are red, the claim follows by induction. If only f is red and e is non-red, then there is a red fork, which protects e by item 2. Again, by induction the claim follows. The case where only e is red is symmetric.
5. Assume the contrary. Discuss a neighbor arc a of g with $w_{k'}(a) = 0$. W.l.o.g., we have $a = (u, v)$, $g = (v, w)$ and $b = (z, v)$.
 If a is not red, then g must be red due to item 2., contradicting $g \in A_\alpha^U$. If a is red, then discuss another arc b that is incident to the common endpoint of a and g . If there is no red b , then the situation contradicts item 3. So, there is a red b . This picture contradicts item 4.
6. Here we must consider α -arcs which are created by **RR-3**. As a matter of principle this situation has the following property: We have two arcs (u, t) and (t, v) such that u is a join and v is a fork. This **RR-3**-application became possible because an arc a incident to t had been deleted in a previous reduction step. Now a must be either of the form (r, t) or (t, r) . Thus, either (u, t) or (t, v) was an α -arc before a 's deletion. W.l.o.g., we assume (u, t) was this α -arc.

We now induce on the number n of **RR-3**-applications involving an α -arc. Clearly, for $n = 0$ the claim holds. Suppose now it is also true for some n . Now (u, t) is an α -arc. If (u, t) is not red then by induction hypothesis we have $w_{k'}((u, t)) = 1$. Hence, after the **RR-3** application we have $w_{k'}((u, v)) = 1$. If (u, t) was red then the emerging α -arc (u, v) is also red and hence the premise does not apply.

□

We would like to stress the fact that the dominance of the red arcs is crucial especially for Theorem 9.3.4.6. Without this dominance, it would be possible to generate non-red α -arcs by **RR-3** such that their $w_{k'}$ -weight is smaller than one. We give an example: Let (u, v) be a red α -arc such that exactly (v, z_1) , (v, z_2) are additionally incident to v . If all three arcs have $w_{k'}$ -weight smaller one then by deleting (v, z_1) and applying **RR-3** we would obtain an (non-red) α -arc with weight smaller one. This would affect the run time as we will see in the next section.

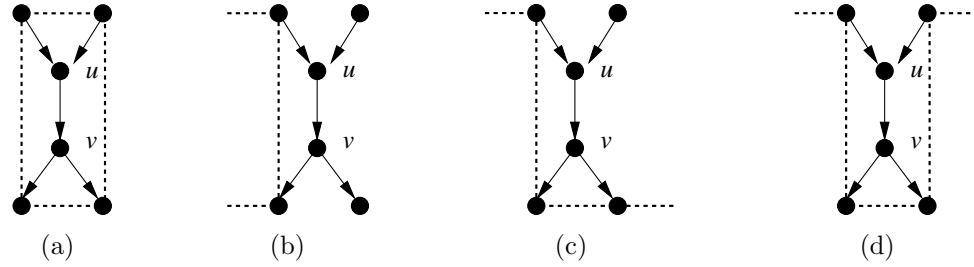


Figure 9.3.: Dotted undirected lines indicate arcs which can be directed in both ways.

9.3.2.2. Estimating the Running Time for Max-Degree-3 Graphs.

For an arc $a \in AN(g)$, where g is an α -arc, let $AN_{ecl}(a) := AN(a) \setminus (AN(g) \cup \{g\})$. In Algorithm 13, depending on in which case of Figure 9.1(a) (procedure "Adjust") we end up with, we decrement k' by a different amount for each arc $a \in AN(g)$ in the case that we put g into \mathcal{MAS} . Due to 9.3.4.6 for the arc g we can decrement k' by one. We can be sure that we may decrement k' by at least $(1 - \omega)$ for each neighbor $a \in AN(g)$ due to item 5. of Theorem 9.3.4.

If we do not put g into \mathcal{MAS} , we delete g and $AN(g)$ immediately afterwards by **RR-1**, decrementing k' accordingly (by $w_{k'}(AN(g))$). Moreover, if case *b.* applies to $a \in AN(g)$, we know that the two arcs $d, e \in AN_{ecl}(a)$ obey $w_{k'}(d) \cdot w_{k'}(e) > 0$ (observe that we do not have triangles). By deleting a , no matter whether **RR-1** or **RR-3** applies to d and e (this depends on the direction of the arcs), we can decrement k' by an extra amount of at least $(1 - \omega)$, cf. the handling of k' by these reduction rules. This is true even if $V(d), V(e) \subseteq V(AN(g))$ as we will argue in the following claim.

Proposition 9.3.5: Let $a \in AN(g)$ for some $g \in A_\alpha^U$ which matches case *b.* Then we get a reduction of at least $2 - \omega$ with respect to a in case of deleting g .

Proof. Note that if for all $a \in AN(g)$ we have that $V(AN_{ecl}(a)) \subseteq V(AN(g))$, then $A(V(AN(g)))$ is a component of 9 arcs, see Figure 9.3(a), which are handled separately in line 5 of Alg. 13. We now examine the case where there are $a_1, a_2 \in AN(g)$ with $AN_{ecl}(a_1) \cap AN_{ecl}(a_2) \neq \emptyset$ and case *b.* applies to both. Let a_1, \dots, a_ℓ (with $2 \leq \ell \leq 4$) be a maximal sequence of arcs from $AN(g)$ such that $AN_{ecl}(a_i) \cap AN_{ecl}(a_{i+1}) \neq \emptyset$ ($1 \leq i \leq \ell - 1$). The arcs in $\bigcup_{i=1}^{\ell} AN_{ecl}(a_i)$ form a directed or an undirected path P as indicated in Figure 9.3 (for $\ell = 2$ see Figure 9.3(b), for $\ell = 3$ see Figure 9.3(c) and for $\ell = 4$ see Figure 9.3(d)). Let $s_0, s_1, \dots, s_\ell, s_{\ell+1}$ be the vertices of P . Observe that we must have $s_0 \neq s_{\ell+1}$ for any $\ell \in \{2, 3, 4\}$. In case $\ell \in \{2, 3\}$, $s_0 = s_{\ell+1}$ would imply a directed or undirected 2- or 3-cycle which contradicts Lemma 9.3.1. If $\ell = 4$ then $A(V(AN(g)))$ is a component which was already excluded. Summarizing P is a undirected path of $\ell + 1$ arcs having each $w_{k'}$ -weight at least $(1 - \omega)$. Suppose there is a vertex of P which is a source or a sink after deleting $AN(g) \cup \{g\}$. Then, it is rather obvious that **RR-1** will delete all arcs of P . This yields a reduction of $(\ell + 1) \cdot (1 - \omega)$ with respect to P . Thus, we can say that we get a reduction of at least $(2 - \omega)$ for each a_i . If no vertex of P is a source or a sink after the deletion of $AN(g) \cup \{g\}$ then P is

α -arc g	$a.$	$b.$	$c.$	$b'.$
\mathcal{MAS}	1	ω	$(1 - \omega)$	ω
Deletion	1	$(2 - \omega)$	$(1 - \omega)$	1

Table 9.1.: Summarizes by which amount k' can be decreased for $a \in AN(g)$, subject to if we take g into \mathcal{MAS} or delete g and to the case applying to a .

also directed. Therefore, **RR-3** yields a reduction of $\ell(1 - \omega)$ for P . Now considering all paths of this form, which occur in $\bigcup_{a \in AN(g)} AN_{ecl}(a)$ finally proves the proposition. \square

Let i denote the number of arcs $a \in AN(g)$ for which case $a.$ applies. In the analogous sense j stands for the case $b.$ and q for $c.$ For every positive integer solution of $i + j + q = 4$, we can state a total of 15 recursions T_1, \dots, T_{15} according to Table 9.1 depending on ω (ignoring the last column for the moment). Find an overview of those recurrences in Table 9.2(a). For every T_i and for a fixed ω , we can calculate a constant $c_i(\omega)$ such that for the branching number c of T_i we have $c \in \mathcal{O}^*(c_i(\omega)^k)$. We want to find a ω with subject to minimize $\max\{c_1(\omega), \dots, c_{15}(\omega)\}$. We numerically obtained $\omega = 0.1687$ so that $\max\{c_1(\omega), \dots, c_{15}(\omega)\}$ evaluates to 1.201, see Table 9.2(a) for an overview of the involved recurrences.

The dominating cases are when $i = 0, j = 0, q = 4$ (T_5) and $i = 0, j = 4, q = 0$ (T_{15}). We conclude that \mathcal{MAS} on graphs G with $\Delta(G) \leq 3$ can be solved in $\mathcal{O}^*(1.201^k)$. Measuring the run time in terms of $m := |A|$ the same way is also possible. Observe that if we delete an α -arc, we can decrement m by one more. By adjusting T_1, \dots, T_{15} according to this and by choosing $\omega = 0.2016$, we derive an upper bound of $\mathcal{O}^*(1.1798^m)$, see Table 9.2(b) for an overview.

Theorem 9.3.6: \mathcal{MAS} can be solved in $\mathcal{O}^*(1.1798^m)$ and $\mathcal{O}^*(1.201^k)$ on max-degree-3 graphs.

9.3.2.3. A speed-up for the max-degree-3 case.

We will obtain a slightly better bound for the search tree by a precedence rule, aiming to improve recurrence T_5 . If we branch on an α -arc g according to this recurrence, for all $a \in AN(g)$ we have $w_{k'}(a) = (1 - \omega)$. Such α -arcs will be called α_5 -arcs. We add the following rule: branch on α_5 -arcs with least priority. Let $l := |A_\alpha^U|$.

Lemma 9.3.7: Branching on an α_5 -arc, we can assume: $\lfloor \frac{1}{5-4\omega} k' \rfloor \leq l < \lceil \frac{1}{4-4\omega} k' \rceil$.

Proof. If $l \geq \lceil \frac{1}{4(1-\omega)} k' \rceil$, then by deleting A_α^U , we decrement k' by at least $l \cdot 4(1 - \omega) \geq k'$, returning YES. If $l < \lfloor \frac{1}{1+4(1-\omega)} k' \rfloor$, then by taking A_α^U into \mathcal{MAS} , we decrement k' by at most $l \cdot (1 + 4(1 - \omega)) < k'$, returning NO. \square

Employing this lemma, we can find a good combinatorial estimate for a brute-force search at the end of the algorithm. This allows us to conclude:

Theorem 9.3.8: MAAS is solvable in time $\mathcal{O}^*(1.1995^k)$ on maximum-degree-3-graphs.

Proof. So using Lemma 9.3.7, in general we can find an integer $b \geq 1$ such that $l = \left\lceil \frac{1}{4(1-\omega)}k' - b \right\rceil$ (\star). Again, if we decided to delete A_α^U , we decrement k' by at least $l \cdot 4(1-\omega)$, so that afterwards $k' \leq b4(1-\omega)$. If $k' \geq k > 0$, we have to step back and take some arcs of A_α^U into \mathcal{MAS} . For any such arc we can decrement k' by one more than by deleting it. Finally, we have to find at most $\lceil b4(1-\omega) \rceil$ arcs from A_α^U , which we can take in to \mathcal{MAS} without causing any cyclicity.

For this we have $\binom{l}{\lceil b4(1-\omega) \rceil}$ choices, which is biggest for $l = 2 \lceil b4(1-\omega) \rceil$. So for this representation of l and with (\star) we find that $2 \lceil b4(1-\omega) \rceil = \left\lceil \frac{1}{4(1-\omega)}k' - b \right\rceil$.

Thus, the two inequalities $2b4(1-\omega) \leq \frac{k'}{4(1-\omega)} - b + 1$ and $2b4(1-\omega) + 1 \geq \frac{k'}{4(1-\omega)} - b$ hold, which imply

$$\frac{1}{4(1-\omega)(9-8\omega)}k' + 1 \geq b \geq \frac{1}{4(1-\omega)(9-8\omega)}k' - 1.$$

Hence, the number of choices can be upper bounded asymptotically by

$$\mathcal{O}^* \left(\binom{\left(\frac{2}{(9-8\omega)}k' \right)}{\left(\frac{1}{(9-8\omega)}k' \right)} \right) \subseteq \mathcal{O}^* \left(4^{\frac{1}{(9-8\omega)}k'} \right).$$

We mention that we have to take care of the case where $k' = l$. In this case we have to check whether $G[(\mathcal{MAS} \cap A(G)) \cup A_\alpha^U]$ is acyclic and give the appropriate answer. Then the above mentioned run time for recurrence T_5 can be assumed. For $\omega = 0.1723$ we get an improved run time of $\mathcal{O}^*(1.1995^k)$, where again recurrences T_5 and T_{15} are dominating. Further note that we can not make use of Lemma 9.3.7 when we measure the run time in terms of m . \square

Corollary 9.3.9: DIRECTED FEEDBACK VERTEX SET on max-degree-3 graphs is solvable in $\mathcal{O}^*(1.282^n)$.

Proof. We argue that MAAS and DIRECTED FEEDBACK VERTEX SET are equivalent for graphs of degree at most three. If A is a feedback arc set, then we can remove instead the set S of vertices the arcs in A are pointing to in order to obtain a directed feedback vertex set with $|S| \leq |A|$. Conversely, if S is a directed feedback vertex set, then we can assume that each vertex $v \in S$ has one ingoing and two outgoing arcs or two ingoing and one outgoing arc; in the first case, let a_v be the ingoing arc, and in the second case, let a_v be the outgoing arc. Then, $A = \{a_v \mid v \in S\}$ is a feedback arc set with $|A| \leq |S|$. By $m \leq \frac{3}{2}n$ $\mathcal{O}^*(1.1798^{\frac{3}{2}n})$ is a run time upper bound. \square

9.3.2.4. Estimating the run time for $(1, \ell)$ -graphs.

There is a difference to maximum degree 3 graphs, namely the entry for case b . in case of deletion in Table 9.1. For $a \in AN(g)$ it might be the case that $|AN(a) \setminus (AN(g) \cup \{g\})| \geq$

3. When we delete g and afterwards a by **RR-1**, then whether **RR-1** nor **RR-3** applies (due to the lack of sources, sinks or degree two vertices in $AN(a) \setminus (AN(g) \cup \{g\})$). Let $\{x\} = V(a) \setminus V(g)$. Then we say case \tilde{b} . applies to a iff case b . applies and: 1) $d(x) = 3$ or 2) a is the only in- or out-arc of x .

If the conditions 1. and 2. do not hold but case b . applies we speak of case b' .

Thus the mentioned entry should be 1 for b' . As long as $|AN(g)| \geq 5$ the reduction in k' is great enough for the modified table, but for the other cases we must argue more detailed. We introduce two more reduction rules, where the first two are similar to an already mentioned one in [124].

RR-8 Let $x, u, v \in V$ such that $(x, u), (u, v) \in A$ and $d^-(u) = d^-(v) = 1$. Depending on the next cases do the following:

1. a) The arc (u, v) is non-red or b) both arcs $(x, u), (u, v)$ are red: Contract (u, v) to a vertex.
2. The arc (u, v) is red, (x, u) is non-red and $d^-(x) = 1$: Contract (x, u) to a vertex.

See Figure 9.4(a) for an illustration of **RR-8**.

RR-9 Let $x, u, v \in V$ such that $(x, u), (u, v) \in A$ and $d^+(x) = d^+(u) = 1$. Depending on the next cases do the following:

1. a) The arc (x, u) is non-red or b) both arcs $(x, u), (u, v)$ are red: Contract (x, u) to a vertex.
2. The arc (x, u) is red, (u, v) is non-red and $d^+(v) = 1$: Contract (u, v) to a vertex.

RR-10 For a red $g' \in A_\alpha$ with $w_{k'}(g') > 0$, set $k' \leftarrow k' - w_{k'}(g')$ and $w_{k'}(g') \leftarrow 0$.

We also add the next rules to Algorithm 13.

- a) In Figure 9.1(b) let the for-loop in line 3 run up to $i = 10$.
- b) Prefer α -arcs g such that $|AN(g)|$ is maximum for branching in line 6 of Algorithm 13.
- c) Forced to branch on $g \in A_\alpha^U$ with $|AN(g)| = 4$, choose an α -arc with the least occurrences of case b' .

Lemma 9.3.10: 1. The reduction rules **RR-8** and **RR-9** are sound and do not violate Theorem 9.3.4.

2. **RR-10** is sound and does not invalidate Theorem 9.3.4.

Proof. 1. We exemplarily focus on **RR-8** as the soundness of **RR-9** can be proven analogously. Let G' be the graph after the contraction of the corresponding arc. Firstly, assume case 1.a.) matches. In any solution where (u, v) is deleted it can be substituted by (x, u) . If (x, u) is non-red this clearly can be done. If (x, u) is red, then by Lemma 9.3.2 we already branched on (x, u) . If $N_1 \in V(D)$ is the current search tree node then in the sub-tree $r(p(d_{(x,u)}^{N_1}))$ (see chapter 1.4) an in cardinality equivalent solution can be found where (x, u) is deleted and (u, v) not. Analogously as in the proof of **RR-3** no cycles are introduced into the reference search tree as the inserted global references point from left to right. In case 1.b) (u, v) is red and hence will not be deleted. Then any arc set is a solution for G iff it is for G' .

Note that in both cases any vertex that is incident to a fork or join (red fork or join, resp.) in G has also this property in G' . Thus, items 1 – 4 in Theorem 9.3.4 remain valid in G' . As (u, v) is not an α -arc item 6. is not violated. Suppose item 5. is violated by the contraction. Thus, $(x, u) \in A_\alpha^U$ and there is an arc (v, z) in G with $w_{k'}((v, z)) = 0$. By item 2. of Theorem 9.3.4 (u, v) must be red in G . This contradicts the fact that either case 1.a) or case 1.b) matches.

Secondly, assume case 2.) matches. Let y be the unique in-neighbor of x . Now let (y, x) take the role of (x, u) , and (x, u) the role of (u, v) . Then the proof for the case 1.a) also applies.

2. As **RR-10** only manipulates weights of red arcs items 1 – 4 remain valid and no solution will be excluded. As **RR-10** only applies to red α -arcs also items 5 and 6 are not violated. □

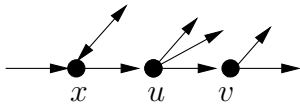
Lemma 9.3.11: We can omit branching on arcs $g \in A_\alpha^U$ with $|AN(g)| = 4$ such that there are four occurrences of case b' .

Proof. Let $g = (u, v)$ and suppose the contrary holds. Clearly, $d^+(v) = 2$. For the arc (v, a) case b' must match, see Figure 9.4(b). This means that $d^+(a) = 1$ and $d^-(a) \geq 3$ due to having case b' . If we had $d^-(a) = 1$ then case \tilde{b} applied to a . Thus, $d^+(a) = 1$. For the distinct arc $r = (a, y)$ assume we have $d^+(y) > 1$, see Figure 9.4(b).

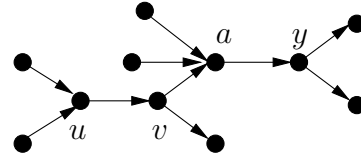
Thus r must be an non-red α -arc (otherwise, **RR-10** could be applied due to having case b' and therefore $w_{k'}(r) > 0$). Also we must have $|AN(r)| = 5$ by $d^-(a) \geq 3$. This contradicts the choice of the α -arc g , because we would have preferred r as also $w_{k'}(r) = 1$ (Theorem 9.3.4.6).

Now we discuss the case $d^+(y) = 1$. Let s be the distinct out-neighbor of y . Only if $d^+(s) > 1$, (a, y) is red and (y, s) non-red prevents **RR-9** from being applied. Thus, (y, s) is an non-red α -arc, where case \tilde{b} . applies to (a, y) with respect to (y, s) , a contradiction as we would have preferred (y, s) . □

Let x, y, z denote the occurrences of cases $a.$, b' and c . To upper bound the branchings according to α -arcs g with $|AN(g)| \geq 5$, we put up all recurrences resulting from integer solutions of $x + y + z = 5$. Note that we also use the right column of Table 9.1. To upper



(a) **RR-8** applies to (v, w) . The double-directed arc indicates that exactly one direction must occur.



(b) The contradicting situation in the proof of Lemma 9.3.11

Figure 9.4.:

bound branchings with $|AN(g)| = 4$, we put up all recurrences obtained from integer solutions of $x + y + z = 4$, except when $x = z = 0$ and $y = 4$ due to Lemma 9.3.11. Additionally, we have to cover the case where we have three occurrences of case b' and one of case \tilde{b} . ($T[k] \leq T[k - (1 + 4\omega)] + T[k - (5 - \omega)]$ and $T[m] \leq T[m - (1 + 4\omega)] + T[m - (6 - \omega)]$, resp.).

Theorem 9.3.12: On $(1, \ell)$ -graphs with m arcs, MAAS is solvable in time $\mathcal{O}^*(1.2133^m)$ ($\omega = 0.3534$) and $\mathcal{O}^*(1.2471^k)$ ($\omega = 0.3333$), respectively.

9.4. Reparameterization

M. Mahajan, V. Raman and S. Sidkar [114] have discussed a rather general setup for re-parameterization of problems according to a “guaranteed value.” In order to use their framework, we only need to exhibit a family of example graphs where Newman’s approximation bound for MAAS is sharp. For a concrete example for the following construction, we refer to Fig. 9.5. Consider $G_r(V_r, A_r)$, $r \geq 2$, with $V_r = \{(i, j) \mid 0 \leq i < r, 0 \leq j \leq 7\}$, and A_r contains two types of arcs:

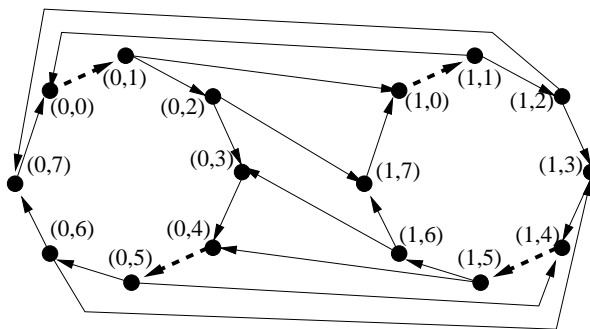
1. $((i, j), (i, (j + 1) \bmod 8))$ for $0 \leq i \leq r$ and $0 \leq j \leq 7$.
- 2a. $((i, j), ((i + 1) \bmod r, (1 - j) \bmod 8))$ for $0 \leq i < r$ and $j = 1, 2$.
- 2b. $((i + 1) \bmod r, (1 - j) \bmod 8, (i, j))$ for $0 \leq i < r$ and $j = 3, 4$.

For $r = 2$ we find an example in Figure 9.5. G_r is cubic with $|V_r| = 8r$ and $|A_r| = 12r$. Its α -arcs are $((i, 0), (i, 1))$ and $((i, 4), (i, 5))$ for $0 \leq i < r$. Since we have to destroy all ‘rings’ as described by the arcs from 1., any feasible solution to these instances require r arcs to go into the feedback arc set. Also r arcs suffice, namely $((0, 4), (0, 5))$ and $((i, 0), (i, 1))$ for $0 < i < r$, giving the ‘tight example’ as required in [114] to conclude:

Corollary 9.4.1: For any $\epsilon > 0$, the following question is not fixed-parameter tractable unless $\mathcal{P} = \mathcal{NP}$: Given a cubic directed graph $G(V, E)$ and a parameter k , does G possess an acyclic subgraph with at least $(\frac{11}{12} + \epsilon) |E| + k$ many arcs ?

9.5. Conclusions

We presented algorithms for solving MAXIMUM ACYCLIC ARC-INDUCED SUBGRAPH first on general graphs, and then more specific on $(1, \ell)$ -graphs, which form a super-

Figure 9.5.: The graph $G_r(V_r, A_r)$ for $r = 2$

class of cubic graphs. In contrast to the algorithms themselves, the analysis concerning correctness and run time is rather involved.

In the case of $(1, \ell)$ -graphs, the analysis was accomplished in an amortized fashion showing quite a better run time than by a naive analysis. This kind of analysis is also applied in terms of 'parameterized' run time. This is one of the few occasions where this seems to be possible. We think that at least for problems on special graph classes (e.g., cubic graphs), one can find further examples. We have used the concept of a reference search tree which generalizes ordinary search trees. It has been first introduced in [134] in order to solve a dominating set variant (named **POWER DOMINATING SET**) in time $\mathcal{O}^*(1.7548^n)$. Using this concept, we showed the correctness of one reduction rule (**RR-3**) which is crucial for the run time improvement. We would like to point out that the validity of this rule depends on the algorithm. It is not valid on its own. We are looking forward to solve more problems using this concept. It seems that especially problems are suited where we cannot simply delete a vertex once the decision has been made whether to take it into the solution or not. For example problems where the solution must be connected (e.g., **MAX INTERNAL SPANNING TREE** or **CONNECTED DOMINATING SET**) or (directed) feedback problems on general graphs.

One might ask if the algorithm for $(1, \ell)$ -graphs is extensible to arc-weighted graphs. We point out that there is no natural generalization as the α -arcs lose their dominance. In Figure 9.4(b), for example, it could make sense not to delete the arc (u, v) , but instead to delete the incoming arcs of u (as the weight of (u, v) could be greater than the weight sum of the incoming arcs).

Coming back to the feedback problems considered in this chapter, there are quite a few natural questions to ask:

1. The break of the 2^n - resp. 2^m -barrier for DFVS and DFAS do not offer huge improvements at the moment. So, the quest is for better algorithms in these cases.
2. Also, the algorithm needing time 2^n for DFAS (based on dynamic programming, see [138]) has not yet been improved below that barrier. Is this possible with our techniques?
3. Does DFAS allow for any improvements over the previously published algorithm

(for DFVS) in [23]? For example, if there were an $\mathcal{O}^*(2^{10k})$ algorithm for this problem, then the two-sided attack on exact problems described in [20] would yield an $\mathcal{O}^*(2^{((10*1)/(10+1))n})$, i.e., an $\mathcal{O}(1.88^n)$ -algorithm for DFAS thanks to the $\mathcal{O}^*(2^k)$ -algorithm for MAAS from [138].

9.6. Recursions and run times

9.6.1. The maximum degree three case

In column five and six of Table 9.2(a) we state the 15 recurrences necessary for solving the parameterized version of MAAS on maximum degree 3 graphs, which we needed to estimate the run time in Theorems 9.3.6. They are derived from the positive integer solutions of $i+j+q=4$. Here and in the following, we write k although strictly speaking we refer to k' . Similarly, the last two columns of Table 9.2(a) displays the recurrences for the exact, non-parameterized case (measured in m).

Table 9.2.:

(a)

No.	j	q	i	Derived recursion with respect to k	Upper bnd.	Derived recursion with respect to k	Upper bnd.
1	0	0	4	$T[k] \leq T[k-4] + T[k-5]$	$\mathcal{O}^*(1.1674^k)$	$T[m] \leq T[m-5] + T[m-5]$	$\mathcal{O}^*(1.1487^m)$
2	0	1	3	$T[k] \leq T[k-(4-\omega)] + T[k-(5-\omega)]$	$\mathcal{O}^*(1.1745^k)$	$T[m] \leq T[m-(5-\omega)] + T[m-(5-\omega)]$	$\mathcal{O}^*(1.156^m)$
3	0	2	2	$T[k] \leq T[k-(4-2\omega)] + T[k-(5-2\omega)]$	$\mathcal{O}^*(1.1822^k)$	$T[m] \leq T[m-(5-2\omega)] + T[m-(5-2\omega)]$	$\mathcal{O}^*(1.168^m)$
4	0	3	1	$T[k] \leq T[k-(4-3\omega)] + T[k-(5-3\omega)]$	$\mathcal{O}^*(1.191^k)$	$T[m] \leq T[m-(5-3\omega)] + T[m-(5-3\omega)]$	$\mathcal{O}^*(1.1709^m)$
5	0	4	0	$T[k] \leq T[k-(4-4\omega)] + T[k-(5-4\omega)]$	$\mathcal{O}^*(1.201^k)$	$T[m] \leq T[m-(5-4\omega)] + T[m-(5-4\omega)]$	$\mathcal{O}^*(1.1798^m)$
6	1	0	3	$T[k] \leq T[k-(5-\omega)] + T[k-(4+\omega)]$	$\mathcal{O}^*(1.167^k)$	$T[m] \leq T[m-(6-\omega)] + T[m-(4+\omega)]$	$\mathcal{O}^*(1.151^m)$
7	1	1	2	$T[k] \leq T[k-(5-2\omega)] + T[k-4]$	$\mathcal{O}^*(1.174^k)$	$T[m] \leq T[m-(6-2\omega)] + T[m-4]$	$\mathcal{O}^*(1.158^m)$
8	1	2	1	$T[k] \leq T[k-(5-3\omega)] + T[k-(4-\omega)]$	$\mathcal{O}^*(1.1817^k)$	$T[m] \leq T[m-(6-3\omega)] + T[m-(4-\omega)]$	$\mathcal{O}^*(1.165^m)$
9	1	3	0	$T[k] \leq T[k-(5-4\omega)] + T[k-(4-2\omega)]$	$\mathcal{O}^*(1.191^k)$	$T[m] \leq T[m-(6-4\omega)] + T[m-(4-2\omega)]$	$\mathcal{O}^*(1.173^m)$
10	2	0	2	$T[k] \leq T[k-(6-2\omega)] + T[k-(3+2\omega)]$	$\mathcal{O}^*(1.171^k)$	$T[m] \leq T[m-(7-2\omega)] + T[m-(3+2\omega)]$	$\mathcal{O}^*(1.155^m)$
11	2	1	1	$T[k] \leq T[k-(6-3\omega)] + T[k-(3+\omega)]$	$\mathcal{O}^*(1.179^k)$	$T[m] \leq T[m-(7-3\omega)] + T[m-(3+\omega)]$	$\mathcal{O}^*(1.163^m)$
12	2	2	0	$T[k] \leq T[k-(6-4\omega)] + T[k-3]$	$\mathcal{O}^*(1.187^k)$	$T[m] \leq T[m-(7-4\omega)] + T[m-3]$	$\mathcal{O}^*(1.171^m)$
13	3	0	1	$T[k] \leq T[k-(7-3\omega)] + T[k-(2+3\omega)]$	$\mathcal{O}^*(1.181^k)$	$T[m] \leq T[m-(8-3\omega)] + T[m-(2+3\omega)]$	$\mathcal{O}^*(1.164^m)$
14	3	1	0	$T[k] \leq T[k-(7-4\omega)] + T[k-(2+2\omega)]$	$\mathcal{O}^*(1.19^k)$	$T[m] \leq T[m-(8-4\omega)] + T[m-(2+2\omega)]$	$\mathcal{O}^*(1.173^m)$
15	4	0	0	$T[k] \leq T[k-(8-4\omega)] + T[k-(1+4\omega)]$	$\mathcal{O}^*(1.201^k)$	$T[m] \leq T[m-(9-4\omega)] + T[m-(1+4\omega)]$	$\mathcal{O}^*(1.1798^m)$

(b)

No.	z	y	x	Derived recursion with respect to k	Upper bnd.	Derived recursion with respect to m	Upper bnd.
1	0	0	4	$T[k] \leq T[k-4] + T[k-5]$	$\mathcal{O}^*(1.1674^k)$	$T[m] \leq T[m-5] + T[m-5]$	$\mathcal{O}^*(1.149^m)$
2	0	1	3	$T[k] \leq T[k-(4-\omega)] + T[k-(5-\omega)]$	$\mathcal{O}^*(1.1766^k)$	$T[m] \leq T[m-(5-\omega)] + T[m-(5-\omega)]$	$\mathcal{O}^*(1.157^m)$
3	0	2	2	$T[k] \leq T[k-(4-2\omega)] + T[k-(5-2\omega)]$	$\mathcal{O}^*(1.187^k)$	$T[m] \leq T[m-(5-2\omega)] + T[m-(5-2\omega)]$	$\mathcal{O}^*(1.166^m)$
4	0	3	1	$T[k] \leq T[k-(4-3\omega)] + T[k-(5-3\omega)]$	$\mathcal{O}^*(1.1986^k)$	$T[m] \leq T[m-(5-3\omega)] + T[m-(5-3\omega)]$	$\mathcal{O}^*(1.1176^m)$
5	0	4	0	$T[k] \leq T[k-(4-4\omega)] + T[k-(5-4\omega)]$	$\mathcal{O}^*(1.2118^k)$	$T[m] \leq T[m-(5-4\omega)] + T[m-(5-4\omega)]$	$\mathcal{O}^*(1.1.2133^m)$
6	1	0	3	$T[k] \leq T[k-(5-\omega)] + T[k-(4+\omega)]$	$\mathcal{O}^*(1.167^k)$	$T[m] \leq T[m-(6-\omega)] + T[m-(4+\omega)]$	$\mathcal{O}^*(1.15^m)$
7	1	1	2	$T[k] \leq T[k-(5-2\omega)] + T[k-4]$	$\mathcal{O}^*(1.176^k)$	$T[m] \leq T[m-(6-2\omega)] + T[m-4]$	$\mathcal{O}^*(1.159^m)$
8	1	2	1	$T[k] \leq T[k-(5-3\omega)] + T[k-(4-\omega)]$	$\mathcal{O}^*(1.1862^k)$	$T[m] \leq T[m-(6-3\omega)] + T[m-(4-\omega)]$	$\mathcal{O}^*(1.168^m)$
9	1	3	0	$T[k] \leq T[k-(5-4\omega)] + T[k-(4-2\omega)]$	$\mathcal{O}^*(1.1978^k)$	$T[m] \leq T[m-(6-4\omega)] + T[m-(4-2\omega)]$	$\mathcal{O}^*(1.178^m)$
10	2	0	2	$T[k] \leq T[k-(6-2\omega)] + T[k-(3+2\omega)]$	$\mathcal{O}^*(1.171^k)$	$T[m] \leq T[m-(7-2\omega)] + T[m-(3+2\omega)]$	$\mathcal{O}^*(1.155^m)$
11	2	1	1	$T[k] \leq T[k-(6-3\omega)] + T[k-(3+\omega)]$	$\mathcal{O}^*(1.18^k)$	$T[m] \leq T[m-(7-3\omega)] + T[m-(3+\omega)]$	$\mathcal{O}^*(1.164^m)$
12	2	2	0	$T[k] \leq T[k-(6-4\omega)] + T[k-3]$	$\mathcal{O}^*(1.191^k)$	$T[m] \leq T[m-(7-4\omega)] + T[m-3]$	$\mathcal{O}^*(1.174^m)$
13	3	0	1	$T[k] \leq T[k-(7-3\omega)] + T[k-(2+3\omega)]$	$\mathcal{O}^*(1.1179^k)$	$T[m] \leq T[m-(8-3\omega)] + T[m-(2+3\omega)]$	$\mathcal{O}^*(1.163^m)$
14	3	1	0	$T[k] \leq T[k-(7-4\omega)] + T[k-(2+2\omega)]$	$\mathcal{O}^*(1.19^k)$	$T[m] \leq T[m-(8-4\omega)] + T[m-(2+2\omega)]$	$\mathcal{O}^*(1.173^m)$
15	4	0	0	$T[k] \leq T[k-(8-4\omega)] + T[k-(1+4\omega)]$	$\mathcal{O}^*(1.195^k)$	$T[m] \leq T[m-(9-4\omega)] + T[m-(1+4\omega)]$	$\mathcal{O}^*(1.176^m)$

Chapter 10.

Conclusion

This thesis has two main contributions: Firstly, we defined a search structure called reference search tree which goes beyond ordinary ad-hoc search trees. Secondly, it was shown that the *Measure&Conquer* method from the field of exact exponential time algorithms could also be applied to several problems in parameterized algorithmics. We call this *Parameterized Measure&Conquer*.

Reference Search Trees: In order to break the trivial enumeration barrier (which is 2^n in case of graph-vertex problems) we aim to cut off whole branches in the search tree traversed by an algorithm. The justification for such a cut-off of a sub tree ST is that any solution found in ST is worse than some solution the algorithm actually finds. Usually in a recursive call we have a set of already fixed objects, i.e., they are either predetermined to be part of the future solution or not. Then there is a set of non-fixed objects on which decisions will be made. These decisions are made in a recursive manner as either their inclusion as also their exclusion from the future solution is considered. Suppose there are two non-fixed objects u and v . Often we can argue the following way for a concrete problem: We do not have to consider the inclusion of u as any solution extending this fixed set is no better than to extend the set which evolves by excluding u and including v . In this sense we cut off the subtree where the exclusion of u is considered. Instead we (figuratively) insert a (local) reference to the sub tree emerged by the fixing of u and v in the above way. Observe that this exchange argument only involved non fixed objects whereas already fixed ones are not touched. As a consequence the inserted reference only points to some node in the search sub tree generated by the current recursive instance. Suppose in the above scenario u is already fixed in a way that it is excluded. If we can guarantee that there also has been (or will be) a recursive call which considers also the inclusion of u we can argue as above. The difference is that we leave the current recursive call and refer to a previous one (which lies further down on the stack). Therefore we call references of this type global. Note that we run into trouble if we have a dependency cycle with respect to the inserted references. Then an optimal solution might be skipped without justification. As long as we are inserting local references this problem cannot occur. This is because every such local reference points to a node further down in the current sub tree. To have a cycle consisting of local references there must be at least one pointing upwards, which by definition is impossible. Now with global references this restriction has been relaxed, they also can point upwards. To circumvent such dependency cycles we constrain them to point either left-right or right-left in the proper drawing of

the (reference) search tree. Frankly speaking, this means that exchange arguments can only be allowed in one direction. For example we skip a solution where some object u is excluded by making a reference to a solution where u is included. Then further on we cannot skip a solution where some object u is included by making a reference to a solution where u is excluded.

Nevertheless, we showed that this enriched kind of search tree can be useful to speed up algorithms for problems with non-local character. We considered POWER DOMINATING SET and MAXIMUM ACYCLIC SUBGRAPH. These two problems do not have a local character like DOMINATING SET or VERTEX COVER as we cannot argue that for any vertex v at least one vertex from $N[v]$ must be in the solution.

So, one future research direction is to find further (non-local) problems where the special structure of reference search trees can be exploited. Especially, problems with some kind of propagation rule are of interest.

Parameterized Measure&Conquer Nowadays the the technique of amortized run time estimation (called *Measure&Conquer*) for branch&reduce algorithms is well-established. Figuratively, the main difference to the traditional way can be described as follows. For a graph-vertex-selection problem we are usually given a budget of n coins. We can think of these n coins as equally distributed among the vertices. In the traditional way of run time estimation once a vertex is fixed to be included or excluded from the future solution the corresponding coin is removed. This indicates that the problem complexity decreased by one. It turned out that this kind of measuring the problem complexity was not powerful enough to break the 2^n -barrier for a lot of graph-vertex-selection problems. In *Measure&Conquer* we also can take only away a fraction of a coin without actually fixing the corresponding vertex v . This often is justified by the fact that we obtained more structural information. This should help us to reduce the problem complexity above-average in a future branch involving v .

As in parameterized algorithmics search trees are a common technique one would expect this technique to carry over nicely. In fact there are some serious obstacles. Firstly, we are only given a budget of k coins which usually is an upper bound of the solution (in case of a minimization problem). In contrast to the n coins above these cannot as easily be located unless the solution itself already is known. Also if we count some fraction of one of the k coins in advance we have to argue that indeed this whole coin has to be spent. A fraction now and the rest in a future branch. This can often be done with the help of structural information which is highly problem dependent. However, there are some rules which all chosen measure in this document had to obey. Let

$$\gamma(G) = k - \sum_{i \geq 1} \omega_i \cdot \chi_i(G)$$

where $0 \leq \omega_i \leq 1$ is a weight. $\chi_i(G)$ is an indicator function for some property and k remains fixed. $\gamma(G)$ represents the general form of all measure for parameterized problems in this thesis. During the execution of the algorithm the following properties should be true for $\gamma(G)$:

1. The application of any reduction does not increase the value of $\gamma(G)$.
2. In any recursive call the value of $\gamma(G)$ must decrease.
3. If $\gamma(G) \leq 0$ then in polynomial time we have to determine if we have a YES- or NO-instance.

In the chapters about this topic we were arguing that this way actually a run time bound of the form $\mathcal{O}^*(c^k)$ can be given.

A promising line of research is to find further problems where the use of such a measure results in a run time speed-up. In particular it is of interest if the compression phase for iterative-compression algorithms (see [144] for an introduction) can be improved for some problems. There are problems where also this compression phase takes exponential time. As an example we state the FEEDBACK VERTEX SET problem on undirected graphs (see J. Chen *et al.* [21]). If the input graph is degree restricted then it is very likely that the use of a measure provides a significant speed-up. Intuitively, this is due to special graph structures that often arise due to rather small degree of the vertices. This constitutes another field of algorithmic research.

Chapter 11.

Appendix

11.1. Code For Local Search

In this section we describe how to use the matlab program *meshsearch* which can be found in section 11.1.1. It is used to obtain a weight assignment in the Measure&Conquer approach. But it also can be used to minimize any given function via local optimization. The first five arguments are:

1. The function in ℓ variables to be minimized.
2. The size of the mesh around the current solution. Or alternatively the half of the side length of a unit hypercube in \mathbb{R}^ℓ with the current solution as midpoint.
3. This parameter defines how fine the measure should be. It determines in how many equally steps in every direction will be taken.
4. How many rounds local optimization will be applied.
5. A check-function which assures certain consistencies a solution should have. The arguments are given to the check-function as a list.

The rest of the arguments (i.e. 6 to $\ell + 5$) consists of the given starting weights. An example:

```
meshsearch(@func,0.2,4,10,@check,0.5,0.6)
```

Here we want to optimize the function *func* with two arguments. The starting values are 0.5 and 0.6. Beginning from the starting value the mesh expands in every of the four directions by an amount of 0.2. As it is a two-dimensional problem we can go in positive and negative directions for each of the two coordinates. The mesh will consist of every point $(0.5 + q_1 \cdot (0.2/4), 0.6 + q_2 \cdot (0.2/4))$ such that $0 \leq q_1, q_2 \leq 4$.

The next matlab function *func* will represent the seven recursions which upper bound the run time of Algorithm 5:

```
function RT = func(w1,w2)
loes=[];
for i=0:2
```

```
f=@(x) 1-x^-(1+i*(1-w2)+(2-i)*w1)-x^-(1-w1);
z=fzero(f,2);
loes=[loes z];
end;

for i=0:3
    f=@(x) 1-x^-(w2+i*(1-w2)+(3-i)*w1)-x^-(w2);
    z=fzero(f,2);
    loes=[loes z];
end;

erg=max(loes);
RT=erg;
```

The next procedure implements a simple test whether the given arguments are strictly between zero and one:

```
function RT = check(args)
RT=true;
n=length(args);
for i=1 : n
    if(args(i) >1 | args(i)<0)
        RT=false;
    end;
end;
```

11.1.1. Meshsearch.m

```
function RT = meshsearch(varargin)
n=nargin;
args=varargin;
func=args{1};
rad=args{2};
intvs=args{3};
check=args{5};
argsvec=[];
for i=6:n
    argsvec=[argsvec args{i}];
end
min=[inf 0 0];
for i=1:args{4}
locvec=[zeros(1,length(argsvec))];
%display(argsvec);
out=mslhp(func,rad,intvs,argsvec,locvec,1,check);
```

```

display(out);
if(out(1)<min(1))
    min=out;
else
    rad=rad/10;
    display(rad);
end;
argsvec=out(2:length(out));
end;

RT=min;

```

11.1.2. Mshlp

```

function RT = mshlp(f,rad,intvs,argsvec,locvec,j,check)

if(j>length(argsvec))
    argus=argsvec+locvec;
    %fcheck=curry(check);
    br=check(argus);
    if(br)
        func=curry(f);
        for i=1:length(argus)
            func=func(argus(i));
        end;
        RT1=func;
    else

        RT1=inf;
    end;
    RT2=argus;
    RT=[RT1 RT2];
else
    min=inf;
    argus2=zeros(1,length(argsvec));
    for i=0:intvs
        stp=(rad/intvs)*i;
        hlpzeros=[zeros(1,j-1) stp zeros(1,length(locvec)-j)];
        locvec2=locvec+hlpzeros;

        out=mshlp(f,rad, intvs,argsvec,locvec2,j+1,check);

        min2=out(1);
    end;
end;

```

```

        if(min2<min)
            min=min2;
            argus2=out(2:length(out));
        end;
    end;

    for i=0:intvs
        stp=(rad/intvs)*i*(-1);
        hlpzeros=[zeros(1,j-1) stp zeros(1,length(locvec)-j)];
        locvec2=locvec+hlpzeros;

        out=mshlp(f,rad, intvs,argsvec,locvec2,j+1,check);

        min2=out(1);
        if(min2<min)
            min=min2;
            argus2=out(2:length(out));
        end;
    end;

    RT=[min argus2];
end;

```

11.2. AMPL Code for Convex Programming

In this section we state AMPL-Code which models a program of the second form in section 2.3. The Convex Program is derived from the recursions of Algorithm 5. The optimal value for z is 0.9182958341 and therefore the run time of Algorithm 5 is at most $\mathcal{O}^*(2^{0.9182958341n}) \subset \mathcal{O}^*(1.8899^n)$.

```

var z>=0;
var w1>=0;
var w2>=0;
#-----

minimize Obj: z;
#-----
subject to C1: w1<=1;
subject to C2: w2<=1;
subject to C3: z<=1;
subject to C4: z>=w1;
subject to C5: z>=w2;

```


11.2. AMPL Code for Convex Programming

```
#-----  
subject to Line4 {i in 0..3}:  
2^(-w2)+2^(-w2-3*w1+i*w1+i*w2)*2^(-i*z) <=1;  
subject to Line2 {j in 0..2}:  
2^(w1)*2^(-z)+2^(-2*w1+j*w2+j*w1)*2^(-z-z*j) <=1;
```


Bibliography

- [1] A. Aazami and M. D. Stilp. Approximation algorithms and hardness for domination with propagation. In *Approximation, Randomization, and Combinatorial Optimization Algorithms and Techniques (APPROX-RANDOM)*, volume 4627 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007. (Cited on page 168.)
- [2] R. B. Allan and R. Laskar. On domination and independent domination numbers of a graph. *Discrete Mathematics*, 23(2):73–76, 1978. (Cited on page 145.)
- [3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation; Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. (Cited on pages 16 and 17.)
- [4] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal of Discrete Mathematics*, 12:289–297, 1999. (Cited on page 185.)
- [5] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962. (Cited on page 89.)
- [6] B. Berger and P. W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 236–243, 1990. (Cited on pages 183, 184 and 187.)
- [7] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987. (Cited on page 145.)
- [8] Daniel Binkele-Raible, Ljiljana Brankovic, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, and Peter Rossmanith. A parameterized route to exact puzzles: Breaking the 2-barrier for irredundance. In Tiziana Calamoneri and Josep Díaz, editors, *Algorithms and Complexity, 7th International Conference, CIAC 2010, Proceedings*, volume 6078 of *LNCS*, pages 311–322. Springer, 2010. (Cited on page 10.)
- [9] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Symposium on Theory of Computation (STOC)*, pages 67–74. ACM Press, 2007. (Cited on pages 45 and 46.)

- [10] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The travelling salesman problem in bounded degree graphs. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, editors, *International Colloquium on Automata, Languages and Programming I (ICALP)*, volume 5125 of *LNCS*, pages 198–209. Springer, 2008. (Cited on pages 90 and 92.)
- [11] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and manets. In *Handbook of Combinatorial Optimization*, volume B, pages 329–369. Springer, 2005. (Cited on page 127.)
- [12] B. Bollobás and E. J. Cockayne. Graph-theoretic parameters concerning domination, independence, and irredundance. *J. Graph Theory*, 3:241–250, 1979. (Cited on page 145.)
- [13] B. Bollobás and E. J. Cockayne. On the irredundance number and maximum degree of a graph. *Discrete Mathematics*, 49:197–199, 1984. (Cited on page 145.)
- [14] P. S. Bonsma, T. Brueggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In B. Rován and P. Vojtáš, editors, *Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 259–268. Springer, 2003. (Cited on page 127.)
- [15] P. S. Bonsma and F. Zickfeld. A $3/2$ -approximation algorithm for finding spanning trees with many leaves in cubic graphs. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 5344 of *LNCS*, pages 66–77. Springer, 2008. (Cited on pages 111 and 127.)
- [16] P. S. Bonsma and F. Zickfeld. Spanning trees with many leaves in graphs without diamonds and blossoms. In E. S. Laber, C.F. Bornstein, L. T. Nogueira, and L. Faria, editors, *Latin American Symposium on Theoretical Informatics (LATIN)*, volume 4957 of *LNCS*, pages 531–543. Springer, 2008. (Cited on page 127.)
- [17] D. J. Brueni and L. S. Heath. The PMU placement problem. *SIAM Journal of Discrete Mathematics*, 19:744–761, 2006. (Cited on pages 167, 168 and 180.)
- [18] R. Carr, T. Fujito, G. Konjevod, and O. Parekh. A $2\frac{1}{10}$ approximation algorithm for a generalization of the weighted edge-dominating set problem. *Journal of Combinatorial Optimization*, 5:317–326, 2001. (Cited on page 48.)
- [19] M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003. (Cited on page 184.)
- [20] J. Chen, H. Fernau, Y. A. Kanj, and G. Xia. Parametric duality and kernelization: lower bounds and upper bounds on kernel size. *SIAM Journal on Computing*, 37:1077–1108, 2007. (Cited on pages 185 and 207.)

- [21] J. Chen, F.V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008. (Cited on page 211.)
- [22] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. In T. Ibaraki, N. Katoh, and H. Ono, editors, *International Symposium on Algorithms and Computation (ISAAC)*, volume 2906 of *Lecture Notes in Computer Science*, pages 148–157, 2003. (Cited on page 142.)
- [23] J. Chen, Y. Liu, S. Lu, B. O’Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In R.E. Ladner and C. Dwork, editors, *Symposium on Theory of Computing (STOC)*, pages 177–186. ACM, 2008. (Cited on pages 184 and 207.)
- [24] B. Chor, M. Fellows, and D. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 257–269, 2004. (Cited on pages 147 and 148.)
- [25] E. J. Cockayne, P. J. P. Grobler, S. T. Hedetniemi, and A. A. McRae. What makes an irredundant set maximal? *J. Combin. Math. Combin. Comput.*, 25:213–224, 1997. (Cited on page 145.)
- [26] E. J. Cockayne, S. T. Hedetniemi, and D. J. Miller. Properties of hereditary hypergraphs and middle graphs. *Canad. Math. Bull.*, 21(4):461–468, 1978. (Cited on page 145.)
- [27] E. J. Cockayne and C. M. Mynhardt. Irredundance and maximum degree in graphs. *Combin. Proc. Comput.*, 6:153–157, 1997. (Cited on page 145.)
- [28] N. Cohen, F.V. Fomin, G. Gutin, E. J. Kim, S. Saurabh, and A. Yeo. An algorithm for finding k -vertex out-trees and its application to k -internal out-branching problem. In Hung Q. Ngo, editor, *Computing and Combinatorics (COCOON)*, volume 5609 of *LNCS*, pages 37–46. Springer, 2009. (Cited on page 90.)
- [29] S. Cook. The complexity of theorem proving procedures. In *Symposium on Theory of Computing (STOC)*, pages 151–158, 1971. (Cited on page 15.)
- [30] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961, 2009. (Cited on page 18.)
- [31] M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S.E. Nikolettseas, and W. Thomas, editors, *International Colloquium on Automata, Languages and Programming I (ICALP)*, volume 5555 of *LNCS*, pages 304–315. Springer, 2009. (Cited on page 18.)

- [32] J. Daligault, G. Gutin, E. J. Kim, and A. Yeo. FPT algorithms and kernels for the directed k -leaf problem. *Journal of Computer and System Sciences*, 2009. <http://dx.doi.org/10.1016/j.jcss.2009.06.005>. (Cited on pages 111, 112, 127, 128 and 141.)
- [33] J. Daligault and S. Thomassé. On finding directed trees with many leaves. In J. Chen and F.V. Fomin, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5917 of *LNCS*, pages 86–97. Springer, 2009. (Cited on page 111.)
- [34] F. Dehne, M. Fellows, H. Fernau, E. Prieto, and F. Rosamond. NONBLOCKER: parameterized algorithmics for MINIMUM DOMINATING SET. In J. Štuller, J. Wiedermann, G. Tel, J. Pokorný, and M. Bielikova, editors, *Software Seminar (SOFSEM)*, volume 3831 of *Lecture Notes in Computer Science*, pages 237–245. Springer, 2006. (Cited on page 42.)
- [35] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. (Cited on pages 19, 20 and 36.)
- [36] R. G. Downey, M. R. Fellows, and V. Raman. The complexity of irredundant sets parameterized by size. *Discrete Applied Mathematics*, 100:155–167, 2000. (Cited on page 146.)
- [37] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms & Applications*, 7:131–140, 2003. (Cited on page 90.)
- [38] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Symp. Discrete Algorithms (SODA)*, pages 781–790. ACM and SIAM, January 2004. (Cited on page 53.)
- [39] G. Erdélyi, H. Fernau, J. Goldsmith, N. Mattei, D. Raible, and J. Rothe. The complexity of probabilistic lobbying. In F. Rossi and A. Tsoukiàs, editors, *Algorithmic Decision Theory (ADT)*, volume 5783 of *LNCS*, pages 86–97. Springer, 2009. (Cited on page 9.)
- [40] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In H. Broersma, M. Johnson, and S. Szeider, editors, *Algorithms and Complexity in Durham (ACiD)*, volume 4 of *Texts in Algorithmics*, pages 1–41. King’s College Publications, 2005. (Cited on pages 127 and 143.)
- [41] O. Favaron. Two relations between the parameters of independence and irredundance. *Discrete Mathematics*, 70(1):17–20, 1988. (Cited on page 145.)
- [42] O. Favaron. A note on the irredundance number after vertex deletion. *Discrete Mathematics*, 121(1-3):51–54, 1993. (Cited on page 145.)

- [43] O. Favaron, T. W. Haynes, S. T. Hedetniemi, M. A. Henning, and D. J. Knisley. Total irredundance in graphs. *Discrete Mathematics*, 256(1-2):115–127, 2002. (Cited on page 145.)
- [44] M. R. Fellows. Blow-ups, win/win’s, and crown rules: Some new directions in fpt. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 2880 of *LNCS*, pages 1–12. Springer, 2003. (Cited on page 147.)
- [45] M. R. Fellows, G. Fricke, S. T. Hedetniemi, and D. P. Jacobs. The private neighbor cube. *SIAM J. Discrete Math.*, 7(1):41–47, 1994. (Cited on page 145.)
- [46] M.R. Fellows, D. Lokshtanov, N. Misra, F.A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In S.H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation, International Symposium (ISAAC)*, volume 5369 of *LNCS*, pages 294–305. Springer, 2008. (Cited on page 18.)
- [47] H. Fernau. Edge dominating set: efficient enumeration-based exact algorithms. In H. L. Bodlaender and M. Langston, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2006. (Cited on page 48.)
- [48] H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In S. Albers and J.Y. Marion, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 09001 of *Dagstuhl Seminar Proceedings*, pages 421–432. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2009. (Cited on page 9.)
- [49] H. Fernau, S. Gaspers, D. Kratsch, M. Liedloff, and D. Raible. Exact exponential-time algorithms for finding bicliques in a graph. In S. Cafieri, A. Mucherino, G. Nannicini, F. Tarissan, and L. Liberti, editors, *Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 205–209, 2009. (Cited on page 9.)
- [50] H. Fernau, S. Gaspers, and D. Raible. Exact and parameterized algorithms for max internal spanning tree. In C. Paul and M. Habib, editors, *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 5911 of *LNCS*, pages 100–111, 2009. (Cited on page 9.)
- [51] H. Fernau, A. Langer, M. Liedloff, J. Kneis, D. Kratsch, D. Raible, and P. Rossmanith. An exact algorithm for the maximum leaf spanning tree problem. In J. Chen and F.V. Fomin, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5917 of *LNCS*, pages 161–172. Springer, 2009. (Cited on pages 9, 18, 112, 114, 124, 127, 128, 129, 141 and 168.)

- [52] H. Fernau and D. F. Manlove. Vertex and edge covers with clustering properties: Complexity and algorithms. *Journal on Discrete Algorithms*, 7:149–167, 2009. (Cited on page 42.)
- [53] H. Fernau and D. Raible. Alliances in graphs: a complexity-theoretic study. In J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, F. Plášil, and M. Bieliková, editors, *Software Seminar (SOFSEM), Proceedings Vol. II*, pages 61–70. Institute of Computer Science ASCR, Prague, 2007. (Cited on page 9.)
- [54] H. Fernau and D. Raible. Exact algorithms for maximum acyclic subgraph on a superclass of cubic graphs. In S.-I. Nakano and Md. S. Rahman, editors, *Workshop on Algorithms and Computation (WALCOM)*, volume 4921 of *Lecture Notes in Computer Science*, pages 144–156. Springer, 2008. (Cited on pages 9 and 142.)
- [55] H. Fernau and D. Raible. A parameterized perspective on packing paths of length two. In B. Yang, D.-Z. Du, and C. An Wang, editors, *Combinatorial Optimization and Applications (COCOA)*, volume 5165 of *Lecture Notes in Computer Science*, pages 54–63. Springer, 2008. (Cited on page 9.)
- [56] H. Fernau and D. Raible. A parameterized perspective on packing paths of length two. *Journal of Combinatorial Optimization*, 2008. <http://dx.doi.org/10.1007/s10878-009-9230-0>. (Cited on page 9.)
- [57] H. Fernau and D. Raible. Packing paths: Recycling saves time. In S. Cafieri, A. Mucherino, G. Nannicini, F. Tarissan, and L. Liberti, editors, *Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 79–83, 2009. (Cited on page 9.)
- [58] H. Fernau and D. Raible. Searching trees: an essay. In J. Chen and S. B. Cooper, editors, *Theory and Applications of Models of Computation (TAMC)*, volume 5532 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2009. (Cited on page 10.)
- [59] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume Supplement Volume A, pages 209–258. Kluwer Academic Publishers, 1999. Also AT&T Technical Report No. 99.2.2. (Cited on pages 183, 184 and 185.)
- [60] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. In J. Chen and S. B. Cooper, editors, *Theory and Applications of Models of Computation (TAMC)*, volume 5532 of *Lecture Notes in Computer Science*, pages 221–230. Springer, 2009. (Cited on page 18.)
- [61] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008. (Cited on page 24.)

- [62] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Text in Theoretical Computer Science. Springer, 2006. (Cited on page 20.)
- [63] F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback set in time $\mathcal{O}(1.7548^n)$. In H. L. Bodlaender and M. Langston, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *Lecture Notes in Computer Science*, pages 184–191. Springer, 2006. (Cited on page 185.)
- [64] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008. (Cited on pages 18 and 36.)
- [65] F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for maximum internal spanning tree. *CoRR*, abs/0907.3208, 2009. <http://arxiv.org/abs/0907.3208>. (Cited on page 90.)
- [66] F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for maximum internal spanning tree. In Y. Dong, D.-Z. Du, and O.H. Ibarra, editors, *Algorithms and Computation, International Symposium (ISAAC)*, volume 5878 of *LNCS*, pages 275–282. Springer, 2009. (Cited on page 90.)
- [67] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: domination – a case study. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *International Colloquium on Automata, Languages and Programming I (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 191–203. Springer, 2005. (Cited on pages 24 and 36.)
- [68] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. Technical Report 307, Department of Informatics, University of Bergen, 2005. (Cited on page 180.)
- [69] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5), 2009. (Cited on page 146.)
- [70] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1), 2008. (Cited on page 36.)
- [71] F. V. Fomin, K. Iwama, D. Kratsch, P. Kaski, M. Koivisto, L. Kowalik, Y. Okamoto, J. van Rooij, and R. Williams. 08431 open problems – moderately exponential time algorithms. In F. V. Fomin, K. Iwama, and D. Kratsch, editors, *Moderately Exponential Time Algorithms*, number 08431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. (Cited on page 146.)

Bibliography

- [72] F.V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O^*(2^{0.288})$ independent set algorithm. In *Symposium on Discrete Algorithms (SODA)*, pages 18–25. ACM Press, 2006. (Cited on page 36.)
- [73] F.V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n . *Algorithmica*, 52(2):153–166, 2008. (Cited on pages 24, 36, 112, 141 and 168.)
- [74] F.V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009. (Cited on pages 32, 33, 65, 118, 128, 167, 168 and 185.)
- [75] M. Fürer, S. Gaspers, and S.P. Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. In J. Chen and F.V. Fomin, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5917 of *LNCS*, pages 173–184. Springer, 2009. (Cited on page 18.)
- [76] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32(4):826–834, 1977. (Cited on page 169.)
- [77] M. R. Garey and D. S. Johnson. *Computers and Intractability*. New York: Freeman, 1979. (Cited on page 15.)
- [78] S. Gaspers. *Exponential Time Algorithms: Structures, Measures, and Bounds*. PhD thesis, Department of Computer and Information Science, University of Bergen, Norway, 2008. (Cited on pages 18, 53 and 54.)
- [79] S. Gaspers and M. Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 4271 of *LNCS*, pages 78–89, 2006. (Cited on page 36.)
- [80] S. Gaspers and G. B. Sorkin. A universally fastest algorithms for Max 2-Sat, Max 2-CSP, and everything in between. In *Symposium on Discrete Algorithms (SODA)*, pages 606–615. ACM Press, 2009. (Cited on page 61.)
- [81] F.W. Glover and G.A. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Springer, 2003. (Cited on page 16.)
- [82] R. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Reading, MA: Addison-Wesley, 3. edition, 1989. (Cited on pages 23 and 24.)
- [83] J. Gramm. *Fixed-Parameter Algorithms for the Consensus Analysis of Genomic Data*. Dissertation, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2003. (Cited on page 24.)

- [84] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Applied Mathematics*, 130:139–155, 2003. (Cited on pages 61, 62 and 87.)
- [85] J. Gramm and R. Niedermeier. Faster exact solutions for MAX2SAT. In *Algorithms and Complexity, 4th Italian Conference (CIAC)*, volume 1767 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2000. (Cited on pages 62 and 87.)
- [86] J. Guo, R. Niedermeier, and D. Raible. Improved algorithms and complexity results for power domination in graphs. In *Fundamentals of Computation Theory, International Symposium (FCT)*, volume 3623 of *Lecture Notes in Computer Science*, pages 172–184. Springer, 2005. (Cited on page 168.)
- [87] G. Gutin and A. Yeo. Some parameterized problems on digraphs. *The Computer Journal*, 51(3):363–371, 2008. (Cited on page 185.)
- [88] T. W. Haynes, S. Mitchell Hedetniemi, S. T. Hedetniemi, and M. A. Henning. Domination in graphs applied to electric power networks. *SIAM Journal on Discrete Mathematics*, 15(4):519–529, 2002. (Cited on pages 168 and 172.)
- [89] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*, volume 208 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, 1998. (Cited on page 145.)
- [90] S. T. Hedetniemi, R. Laskar, and J. Pfaff. Irredundance in graphs: a survey. *Congr. Numer.*, 48:183–193, 1985. (Cited on page 145.)
- [91] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10:196–210, 1962. (Cited on pages 17 and 89.)
- [92] T. Hofmeister. An approximation algorithm for MAX-2-SAT with cardinality constraint. In G. Di Battista and U. Zwick, editors, *European Symposium on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003. (Cited on page 62.)
- [93] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. (Cited on page 17.)
- [94] J. Hromkovič. *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, 2001. (Cited on pages 15 and 17.)
- [95] K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In Guohui Lin, editor, *Computing and Combinatorics (COCOON)*, LNCS, pages 108–117. Springer, 2007. (Cited on pages 90 and 93.)

- [96] R. M. Karp. Dynamic programming meets the principle of inclusion-exclusion. *Information Processing Letters*, 1(2):49–51, 1982. (Cited on page 90.)
- [97] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, 1972. (Cited on pages 15 and 183.)
- [98] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2008. (Cited on pages 111, 115, 124, 127, 128, 131, 132, 134 and 141.)
- [99] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Algorithms based on the treewidth of sparse graphs. In D. Kratsch, editor, *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 3787 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 2005. (Cited on page 87.)
- [100] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Parameterized power domination complexity. *Information Processing Letters*, 98(4):145–149, 2006. (Cited on page 168.)
- [101] S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 Annual Conference (ACM77)*, pages 294–300. Association for Computing Machinery, 1977. (Cited on page 90.)
- [102] A. Kojevnikov and A. S. Kulikov. A new approach to proving upper bounds for MAX-2-SAT. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–17. ACM Press, 2006. (Cited on pages 61, 62, 64, 65, 80, 81 and 87.)
- [103] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S.E. Nikolettseas, and W. Thomas, editors, *International Colloquium on Automata, Languages and Programming I (ICALP)*, volume 5555 of *LNCS*, pages 653–664. Springer, 2009. (Cited on pages 112 and 127.)
- [104] A. S. Kulikov and K. Kutzkov. New bounds for max-sat by clause learning. In V. Diekert, M. V. Volkov, and editors A. Voronkov, editors, *Computer Science — Theory and Applications, International Symposium on Computer Science in Russia (CSR)*, volume 4649 of *LNCS*, pages 194–204. Springer, 2007. (Cited on pages 61, 62, 73, 81, 82 and 87.)
- [105] A. S. Kulikov and K. Kutzkov. New upper bounds for the problem of maximal satisfiability. *Journal of Discrete Mathematics and Applications*, 19(2):155–172, 2009. (Cited on pages 61 and 87.)
- [106] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999. (Cited on page 24.)

- [107] O. Kullmann. *Fundamentals of Branching Heuristics*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 205–244. IOS Press, 2009. (Cited on page 24.)
- [108] R. Laskar and J. Pfaff. Domination and irredundance in graphs. Technical Report Techn. Rep. 434, Clemson Univ., Dept. of Math. SC., 1983. (Cited on page 145.)
- [109] E.L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5(3), 1976. (Cited on page 17.)
- [110] M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In William Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization (IPCO)*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002. (Cited on page 62.)
- [111] C.-S. Liao and D.-T. Lee. Power domination problem in graphs. In *International Conference on Computing and Combinatorics (COCOON)*, volume 3595 of *Lecture Notes in Computer Science*, pages 818–828. Springer, 2005. (Cited on page 168.)
- [112] M. Liedloff. *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations*. PhD thesis, LITA, University of Metz (France), 2007. (Cited on page 18.)
- [113] H.-I. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms*, 29:132–141, 1998. (Cited on pages 111 and 127.)
- [114] M. Mahajan, V. Raman, and Sikdar S. Parameterizing MAXNP problems above guaranteed values. In H. L. Bodlaender and M. Langston, editors, *International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2006. (Cited on page 205.)
- [115] D. F. Manlove. *Minimaximal and maximinimal optimisation problems: a partial order-based approach*. PhD thesis, University of Glasgow, Computing Science, 1998. (Cited on page 48.)
- [116] K. Mehlhorn. *Graph algorithms and NP-completeness*. Heidelberg: Springer, 1984. (Cited on page 22.)
- [117] Z. Michalewicz and D.B. Fogel. *How to solve it: Modern Heuristics*, volume 57. Springer, 2nd edition, 2004. (Cited on page 16.)
- [118] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. (Cited on page 17.)

- [119] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems*, 43:234–253, 2008. (Cited on page 42.)
- [120] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985. (Cited on page 17.)
- [121] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (Cited on page 17.)
- [122] I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. *ACM SIGPLAN Notices*, 12, 1973. (Cited on page 184.)
- [123] J. Nederlof. Fast polynomial-space algorithms using mobius inversion: Improving on steiner tree and related problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S.E. Nikolettseas, and W. Thomas, editors, *International Colloquium on Automata, Languages and Programming I (ICALP)*, volume 5555 of *LNCS*, pages 713–725. Springer, 2009. (Cited on page 110.)
- [124] A. Newman. The maximum acyclic subgraph problem and degree-3 graphs. In M. X. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization (APPROX-RANDOM)*, volume 2129 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2001. (Cited on pages 184, 185, 186, 187, 188, 193 and 203.)
- [125] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. (Cited on page 20.)
- [126] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:63–88, 2000. (Cited on page 87.)
- [127] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. (Cited on page 15.)
- [128] J. Plesník. Equivalence between the minimum covering problem and the maximum matching problem. *Discrete Mathematics*, 49:315–317, 1984. (Cited on page 34.)
- [129] S. Poljak and Z. Tuza. Maximum cuts and largest bipartite subgraphs. In Cook, László Lovász, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 181–244, 1995. (Cited on page 61.)
- [130] E. Prieto. *Systematic Kernelization in FPT Algorithm Design*. PhD thesis, The University of Newcastle, Australia, 2005. (Cited on page 90.)
- [131] E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing FPT-algorithms—the case of k -internal spanning tree. In *Proceedings of Workshop on Algorithms and Data Structures (WADS)*, volume 2748 of *Lecture Notes in Computer Science*, pages 465–483. Springer, 2003. (Cited on pages 90 and 93.)

- [132] E. Prieto and C. Sloper. Reducing to independent set structure – the case of k -internal spanning tree. *Nordic Journal of Computing*, 12(3):308–318, 2005. (Cited on page 90.)
- [133] D. Raible and H. Fernau. A new upper bound for MAX-2-SAT: A graph-theoretic approach. In E. Ochmanski and J. Tyszkiewicz, editors, *Mathematical Foundations of Computer Science (MFCS)*, volume 5162 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2008. (Cited on pages 10 and 36.)
- [134] D. Raible and H. Fernau. Power domination in $O^*(1.7548^n)$ using reference search trees. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation, International Symposium (ISAAC)*, volume 5369 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2008. (Cited on pages 10, 24, 36 and 206.)
- [135] D. Raible and H. Fernau. An amortized search tree analysis for k -leaf spanning tree. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, editors, *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 5901 of *LNCS*, pages 672–684. Springer, 2010. (Cited on pages 111 and 114.)
- [136] D. Raible and H. Fernau. A faster exact algorithm for the directed maximum leaf spanning tree problem. In *Computer Science Symposium in Russia (CSR)*, 2010. to appear. (Cited on page 10.)
- [137] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006. (Cited on pages 112, 145 and 184.)
- [138] V. Raman and S. Saurabh. Improved fixed parameter tractable algorithms for two “edge” problems: MAXCUT and MAXDAG. *Information Processing Letters*, 104(2):65–72, 2007. (Cited on pages 184, 206 and 207.)
- [139] V. Raman, S. Saurabh, and S. Sikdar. Improved exact exponential algorithms for vertex bipartization and other problems. In M. Coppo et al., editors, *Italian Conference on Theoretical Computer Science (ICTCS)*, volume 3701 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2005. (Cited on page 47.)
- [140] I. Razgon. Computing minimum directed feedback vertex set in $O(1.9977^n)$. In G.F. Italiano, E. Moggi, and L. Laura, editors, *Theoretical Computer Science, 10th Italian Conference (ICTCS)*, pages 70–81. World Scientific, 2007. (Cited on page 184.)
- [141] G. Salamon. Approximation algorithms for the maximum internal spanning tree problem. In L. Kucera and A. Kucera, editors, *Mathematical Foundations of Computer (MFCS)*, volume 4708 of *LNCS*, pages 90–102. Springer, 2007. (Cited on page 90.)

Bibliography

- [142] G. Salamon and G. Wiener. On finding spanning trees with few leaves. *Information Processing Letters*, 105(5):164–169, 2008. (Cited on page 90.)
- [143] A.D. Scott and G.B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3-4):260–287, 2007. (Cited on pages 61 and 67.)
- [144] C. Sloper. *Techniques in Parameterized Algorithm Design*. PhD thesis, Department of Computer and Information Science, University of Bergen, Norway, 2005. (Cited on page 211.)
- [145] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *European Symposium on Algorithms (ESA)*, volume 1461 of *LNCS*, pages 441–452. Springer, 1998. (Cited on pages 111 and 127.)
- [146] E. Speckenmeyer. On feedback vertex sets and nonseparating independent sets in cubic graphs. *Journal of Graph Theory*, 3:405–412, 1988. (Cited on page 168.)
- [147] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems Man Cybernet.*, 11(2):109–125, 1981. (Cited on page 183.)
- [148] R.E. Tarjan and A.E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977. (Cited on page 17.)
- [149] J. A. Telle. *Vertex Partitioning Problems: Characterization, Complexity and Algorithms on Partial k -Trees*. PhD thesis, Department of Computer Science, University of Oregon, USA, 1994. (Cited on pages 152 and 160.)
- [150] M. T. Thai, F. Wang, D. Liu, S. Zhu, and D.-Z. Du. Connected dominating sets in wireless networks different transmission ranges. *IEEE Trans. Mobile Computing*, 6:1–10, 2007. (Cited on page 127.)
- [151] J. M. M. van Rooij and H. L. Bodlaender. Design by measure and conquer, a faster exact algorithm for dominating set. In Susanne Albers and Pascal Weil, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 657–668. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. (Cited on pages 24, 36 and 168.)
- [152] J. M. M. van Rooij and H. L. Bodlaender. Exact algorithms for edge domination. In M. Grohe and R. Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop (IWPEC)*, volume 5018 of *LNCS*, pages 214–225. Springer, 2008. (Cited on page 48.)

- [153] J. M. M. van Rooij, J. Nederlof, and T.C. van Dijk. Inclusion/exclusion meets measure and conquer. In A. Fiat and P. Sanders, editors, *European Symposium on Algorithms (ESA)*, volume 5757 of *LNCS*, pages 554–565. Springer, 2009. (Cited on pages 18, 36, 42 and 168.)
- [154] V.V. Vazirani. *Approximation Algorithms*. Springer, 2001. (Cited on page 16.)
- [155] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007. (Cited on pages 110 and 142.)
- [156] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. (Cited on page 61.)
- [157] G. Xu, L. Kang, E. Shan, and M. Zhao. Power domination in block graphs. *Theoretical Computer Science*, 359(1–3):299–305, 2006. (Cited on page 168.)
- [158] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal of Applied Mathematics*, 38(3):364–372, June 1980. (Cited on page 48.)

Bibliography

LEBENS LAUF

PERSÖNLICHE ANGABEN

Name: Daniel Binkele-Raible

Staatsangehörigkeit: deutsch

Geburtsdatum: 08.09.1976

Geburtsort: Villingen-Schwenningen

SCHULAUSBILDUNG

1983–1987	Grundschule (Karlschule) in VS-Schwenningen
1987–1993	Progymnasium St. Ursula in VS-Villingen
1993–1996	Gymnasium am Deutenberg in VS-Schwenningen

ZIVILDIENTST

1996–1997	Zivildienst im Mobilen Sozialen Hilfsdienst der Arbeit- erwohlfahrt, Schwarzwald-Baar
-----------	--

BERUFSAUSBILDUNG

1997–1999	Berufsausbildung zum 'Industrietechnologen Fachrich- tung Datentechnik Schwerpunkt Wirtschaft' bei der Siemens AG, München
-----------	--

HOCHSCHULAUSBILDUNG

Okt. 1999 – Sept. 2001	Vordiplom in Informatik, Universität Tübingen
Okt. 2001 – Juli 2002	Hauptstudium der Informatik, Freie Universität Berlin
Aug. 2002 – Mai 2006	Hauptstudium der Informatik, Universität Tübingen
Mai 2006 – Okt. 2009	Doktorand/Wissenschaftlicher Mitarbeiter, Arbeits- bereich Theoretische Informatik, Universität Trier, Doktorvater: Prof. Dr. Henning Fernau