Hybrid AI for Process Management

Improving Similarity Assessment in Process-Oriented Case-Based Reasoning via Deep Learning

Vom Fachbereich IV der Universität Trier zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Dr. rer. nat.) genehmigte Dissertation

von

Maximilian Hoffmann

Trier, 2025

Betreuer:	Prof. Dr. Ralph Bergmann
1. Berichterstatter:	Prof. Dr. Ralph Bergmann
2. Berichterstatter:	Prof. Dr. David B. Leake
Datum der Disputation:	26.05.2025

Abstract

Case-Based Reasoning (CBR) is a symbolic Artificial Intelligence (AI) approach that has been successfully applied across various domains, including medical diagnosis, product configuration, and customer support, to solve problems based on experiential knowledge and analogy. A key aspect of CBR is its problem-solving procedure, where new solutions are created by referencing similar experiences, which makes CBR explainable and effective even with small amounts of data. However, one of the most significant challenges in CBR lies in defining and computing meaningful similarities between new and past problems, which heavily relies on domain-specific knowledge. This knowledge, typically only available through human experts, must be manually acquired, leading to what is commonly known as the knowledge-acquisition bottleneck. One way to mitigate the knowledge-acquisition bottleneck is through a hybrid approach that combines the symbolic reasoning strengths of CBR with the learning capabilities of Deep Learning (DL), a subsymbolic AI method. DL, which utilizes deep neural networks, has gained immense popularity due to its ability to automatically learn from raw data to solve complex AI problems such as object detection, question answering, and machine translation. While DL minimizes manual knowledge acquisition by automatically training models from data, it comes with its own limitations, such as requiring large datasets, and being difficult to explain, often functioning as a ”black box”. By bringing together the symbolic nature of CBR and the data-driven learning abilities of DL, a neuro-symbolic, hybrid AI approach can potentially overcome the limitations of both methods, resulting in systems that are both explainable and capable of learning from data.

The focus of this thesis is on integrating DL into the core task of similarity assessment within CBR, specifically in the domain of process management. Processes are fundamental to numerous industries and sectors, with process management techniques, particularly Business Process Management (BPM), being widely applied to optimize organizational workflows. Process-Oriented Case-Based Reasoning (POCBR) extends traditional CBR to handle procedural data, enabling applications such as adaptive manufacturing, where past processes are analyzed to find alternative solutions when problems arise. However, applying CBR to process management introduces additional complexity, as procedural cases are typically represented as semantically annotated graphs, increasing the knowledge-acquisition effort for both case modeling and similarity assessment. The key contributions of this thesis are as follows: It presents a method for preparing procedural cases, represented as semantic graphs, to be used as input for neural networks. Handling such complex, structured data represents a significant challenge, particularly given the scarcity of available process data in most organizations. To overcome the issue of data scarcity, the thesis proposes data augmentation techniques to artificially expand the process datasets, enabling more effective training of DL models. Moreover, it explores several deep learning architectures and training setups for learning similarity measures between procedural cases in POCBR applications. This includes the use of experience-based Hyperparameter Optimization (HPO) methods to fine-tune the deep learning models. Additionally, the thesis addresses the computational challenges posed by graph-based similarity assessments in CBR. The traditional method of determining similarity through subgraph isomorphism checks, which compare nodes and edges across graphs, is computationally expensive. To alleviate this issue, the hybrid approach seeks to use DL models to approximate these similarity calculations more efficiently, thus reducing the computational complexity involved in graph matching.

The experimental evaluations of the corresponding contributions provide consistent results that indicate the benefits of using DL-based similarity measures and case retrieval methods in POCBR applications. The comparison with existing methods, e. g., based on subgraph isomorphism, shows several advantages but also some disadvantages of the compared methods. In summary, the methods and contributions outlined in this work enable more efficient and robust applications of hybrid CBR and DL in process management applications.

Keywords: Process-Oriented Case-Based Reasoning · Similarity Learning · Graph Neural Networks · Transfer Learning · Hyperparameter Optimization · Learning to Rank · Augmentation

Acknowledgements

Productive research is all about collaboration, sharing experiences and supporting each other. As my journey as a PhD student slowly reaches its end, I want to thank those people who have supported me throughout the work on this thesis.

Ralph and David are the supervisors of this thesis and helped me a lot with their constructive feedback and guidance. I have been working together with Ralph for almost eight years now. Starting as a research student assistant in his group under the supervision of Christian and Gilbert, I took my first steps into the research world. While finishing my Master's studies, Ralph offered me my current position, probably the most important reason why I decided on starting as a PhD student. Throughout the last years, he continuously supported me with feedback, discussions, and other resources. I know David from participating at various ICCBR conferences in the last few years. Due to his reputation in the community and our shared interest in deep learning and case-based reasoning, I decided he is a great fit as my second supervisor. Gladly, he agreed and provided valuable feedback for this thesis. I want to thank both a lot.

My colleagues at the Department of Artificial Intelligence and Intelligent Information Systems also deserve to be mentioned here. We work together daily, and I am happy to be part of the group. I always felt support, not only during working hours but also afterward. Likewise, I am also a member of the ProCAKE core development team that steers the development of ProCAKE. I want to thank Lisa, Lukas, Lukas, Alexander, and Christian for the collaboration. Additionally, I would like to thank Nicolas, who supported me as a student research assistant. I wish him all the best for his own professional career. Special thanks also go to the people who provided feedback on the first draft of the thesis or helped with proofreading: Mirko, Maik, and Johanna.

Every endeavor that spans over a longer period of time, such as a PhD thesis, is not possible without supportive friends and family. I want to thank my friends and family very much. Particularly, I thank my wonderful girlfriend Denise. She supported me from the beginning to the end, had to go without me when I was working longer, and sometimes even had to bear with me talking about the contents of my work.

Maximilian Hoffmann
University of Trier
January 2025

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BPM	Business Process Management
CBR	Case-Based Reasoning
CNN	Convolutional Neural Network
DGL	Deep Graph Library
DL	Deep Learning
DSR	Design Science Research
GAN	Generative Adversarial Network
GEM	Graph Embedding Model
GMN	Graph Matching Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
HPO	Hyperparameter Optimization
IML	Informed Machine Learning
IoT	Internet of Things
LLM	Large Language Model
LSTM	Long Short-Term Memory
LTR	Learning To Rank
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MPNN	Message Passing Neural Network
MSE	Mean Squared Error
NLP	Natural Language Processing
PAIS	Process-Aware Information System
POCBR	Process-Oriented Case-Based Reasoning
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
sGEM	Semantic Graph Embedding Model
sGMN	Semantic Graph Matching Network
SNN	Siamese Neural Network
TL	Transfer Learning
XAI	eXplainable Artificial Intelligence

Part I

Preamble

Chapter 1

Introduction

Case-Based Reasoning (CBR) [1, 14, 129] is a symbolic *Artificial Intelligence (AI)* method [51] that was first discussed at the end of the 1980s [85]. Over the years, it has been applied to many different domains and use cases such as medical diagnosis [72], product configuration [149], and customer support [6]. CBR has proven to be a successful methodology for problem-solving based on experiential knowledge and analogy. The way problems in CBR are solved is closely inspired by the human behavior of solving problems, which is, by comparing them to already experienced situations and applying successful solutions from the past to the current situation. Central to this procedure is the concept of experiences, so-called *cases*, and their management [14]. Each individual case in CBR contains knowledge about a specific problem-solving situation, usually described by a problem and a solution. When a new problem is encountered, the solution for the new situation is derived from the solutions of useful, experienced cases. This has the advantage of a straightforward *explainability*, as the new solution can directly be explained with the case it is derived from [139]. Additionally, the method usually works well with *small amounts of data* since there is no need to generalize the cases into a more generic model like in most *Machine Learning (ML)* methods. However, an important challenge of CBR systems arises when taking a closer look at how experienced and new problems are compared, i. e., by computing pairwise *similarities*. The notion of similarity is central in CBR applications, and their main usage is the approximation of utility for the problem-solving procedure [14, 128]. The principle that the solution of one problem is likely to be useful for solving other problems if these problems are similar, is followed. Concrete similarity values are determined by similarity measures. The definition of these measures can be very challenging, as they need to consider domain-specific semantic knowledge to provide similarities that actually approximate the utility of an experienced case for solving the current problem. The required knowledge is usually only available from human experts, and it has to be manually acquired in a resource-intensive process, leading to a *knowledge-acquisition bottleneck* [58]. This bottleneck is also present in other parts of CBR applications, such as the modeling of the cases, where experiences are commonly not formalized and accessible but rather only exist as part of the experience of human experts. Consequently, solutions for reducing the knowledge-acquisition bottleneck are of high interest in the CBR community.

One solution approach is the integration of automatic learning capabilities of research disciplines such as *Deep Learning (DL)* [92]. DL is a sub-symbolic AI method [51] from the category of ML, characterized by the usage of *deep neural network models* (see [3] for an introduction). While the first concepts of neural networks as structural models of human brains date back to the 1940s, their current rise in popularity has started in the late 2000s due to significant advances in computation capabilities. With growing amounts of storage capacity and data as well as powerful computation hardware, DL models achieved state-of-the-art results in many ML and AI problems such as object detection [175], question answering [20], and machine translation [9]. DL applications mostly prevent the knowledge-acquisition bottleneck by utilizing models that are *automatically learned* from raw data for predicting solutions to problems. Therefore, neural networks are fed with pairs of input data (problem) and output data (solution) in a training procedure. The neural network automatically learns to adjust its trainable parameters such that the inputs lead to the correct outputs. Once the model is trained, it can be used to compute outputs for new, unseen input data, thus enabling problem-solving. Nevertheless, DL methods also come with their own challenges: They require large, high-quality datasets to train effectively and to generalize well to unseen data. Collecting and labeling such datasets is time-consuming and costly in most domains. Additionally, DL models often operate as “black boxes”, making their decision-making processes difficult to interpret. This lack of explainability is problematic in critical applications where understanding a model’s predictions is essential for trust and regulatory compliance. Furthermore, there is a multitude of available model architectures and training methods, thus, complicating model selection and configuration. This complexity demands significant expertise and experimentation, leading to a trial-and-error approach that consumes plenty of resources.

Bringing together symbolic and sub-symbolic AI methods in a hybrid setup is a prominent topic in the research community. Referred to as neuro-symbolic AI[1] [51, 138], the core idea is to combine the strengths of each of the fields to overcome the respective weaknesses. The main premise is to combine the neural ability to train robust, expressive, and performant DL models from raw data with the explainability, correctness, and focus on structured human expert knowledge of symbolic approaches. This thesis investigates one example of such a hybrid approach, i. e., the combination of DL and CBR [87, 88], with the main goal of diminishing existing knowledge-acquisition bottlenecks of CBR applications. The hybrid approach specifically focuses on the central CBR task of similarity assessment.

The examined use case and domain of the thesis is *process management*, as processes are in widespread use with a large impact throughout society, businesses, and academia. This impact is noticeable due to prominent research disciplines like *Business Process Management (BPM)* [38] that solely focus on methods for improving the processes within all kinds of organizations. BPM covers different aspects, such as process discovery [153] and redesign [127], and manages processes along their lifecycle [38, pp. 16-27]. Typical BPM tasks range from more traditional ones such as supply chain management [150] and customer relationship management [118] to more modern ones such as smart homes [145] and adaptive manufacturing [103]. The field of CBR is also concerned with research on BPM as part of *Process-Oriented Case-Based Reasoning (POCBR)* [15, 112]. POCBR applications use the CBR methodology [159] to solve problems involving procedural data. For instance, a POCBR application could be used to provide adaptive and flexible manufacturing [102]. If some kind of problem is detected with the executed manufacturing process, e. g., a machine failure, the current process can be used to find similar manufacturing processes with the same goal but without using the defective machine. To implement such applications, existing CBR approaches can only rarely be used out-of-the-box. Instead, the focus on procedural data requires specific considerations. These considerations ultimately increase the involved complexity and exacerbate the knowledge-acquisition bottleneck, making the aforementioned hybrid approaches even more worthwhile.

Conceptualizing and implementing a hybrid approach of DL and POCBR methods poses several challenges: Procedural cases in POCBR are commonly represented as semantically annotated graphs [15], with types and semantic annotations for each node and each edge and a

[1] Often used interchangeably with the terms hybrid AI, *Informed Machine Learning (IML)* [133], and System 1 and System 2 DL [13].

complex topology. This poses a challenge for the knowledge-acquisition procedure. An analogous challenge is present for the specification of similarity knowledge and the associated measures that operate on the process data. The effort for knowledge-acquisition is commonly high for such measures, as their definition has to be based on some ground-truth that is usually determined by human domain experts [58]. Further, most similarity measures for semantic graphs rely on some form of subgraph isomorphism check. This means that the similarity value is determined w. r. t. the number of nodes and edges with a one-to-one correspondence in both graphs. Checking this is computationally expensive, usually with polynomial or exponential complexity [15, 21, 120]. There are also challenges for the DL side: The complex information of POCBR cases has to be processed by neural networks, which requires a specialized setup of input data preparation, neural network architecture, and training method. A particular challenge is the general requirement of large amounts of data to train DL models that is conflicting with the small amounts of training data of CBR applications. POCBR data is usually even more scarce, as processes are commonly modeled manually and are not available in large quantities from organizations. In addition, the general challenge of adequate configuration and parameterization of the used DL setup is also present when using DL for process data. Due to the specialization of the setup, the importance of useful configurations and hyperparameters is increased.

This thesis addresses the aforementioned challenges with the following contributions:

- Comprehensive preparation method of procedural cases represented as semantic graphs to be used in neural network training
- Several DL architectures, training setups, and an experience-based *Hyperparameter Optimization (HPO)* method for learning similarity measures between procedural cases to be used in POCBR applications
- Data augmentation methods for enlarging the existing number of cases of POCBR applications to conduct effective neural network training

The thesis is a compilation thesis and follows the principles of *Design Science Research (DSR)* [62, 63, 123]. This means that the contributions are structured in three artifacts with separate research questions and contributing publications. The general outline of the thesis is structured into three parts, i. e., the *preamble* in Part I, the individual *publications* in Part II, and the *appendix* in Part III. The preamble introduces, summarizes, and puts the published papers into their outer and inner context. Thereby, the scientific landscape presents foundations and related approaches that are important to embed the contributions of this thesis in the scientific context (see Ch. 2). In Ch. 3, the research questions are raised and the contributions of the thesis, consolidated into three artifacts, are explained. The chapter also focuses on the relations among the original publications and the artifacts as part of the proposed approach. Eventually, the preamble is completed by concluding the key findings of the thesis w. r. t. the research questions and presenting directions for future research in Ch. 4. Part II is composed of a collection of the five published papers (see Ch. 5 to 9) that are part of this compilation thesis. The appendix contains additional information, i. e., a CV of the author of this thesis (see App. A) and a complete list of his publications (see App. B).

Chapter 2

Scientific Landscape

This thesis examines the combination of methods and technologies from POCBR and DL in a hybrid approach. More specifically, the essential CBR task of similarity assessment between procedural cases is addressed. The main research fields and related work within the scientific landscape surrounding this topic are introduced in this chapter.

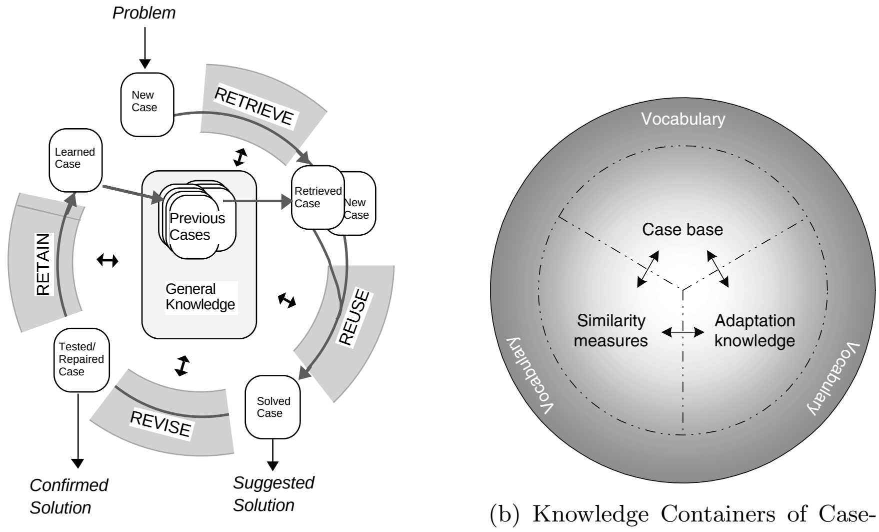
2.1 Case-Based Reasoning

CBR [1, 14, 129] is a problem-solving methodology, originating in the 1980s, that leverages experiences and domain knowledge in the problem-solving procedure. The approach is an example of a symbolic AI method [51] and has many successful applications in practice and academia, e. g., customer support [6], manufacturing [103], argumentation [93], aquaculture [110], and many more.

2.1.1 CBR Cycle and Knowledge Containers

At the core of CBR applications is a four-phased cycle, called *4R cycle* [1], that provides a common interface for problem-solving (see Fig. 2.1a): In the *retrieve* phase, a case base of experienced problem-solution-pairs, called cases, is queried with the current problem to find similar problems with a potential solution. The retrieved cases are then *reuse*d with the goal of solving the current problem with the found solutions. Reusing these cases often goes along with modifications to the retrieved solutions (called case adaptation [57]), as the solutions often do not fit well enough to solve the current problem sufficiently. If a solution to the current problem is proposed, it is evaluated and *revise*d if needed, e. g., by a human expert or in a simulation. The fourth (and last) phase is the *retain* phase, where it is decided for the newly created case of current problem and revised solution whether it is added to the case base or not.

The phases of this cycle rely on several types of knowledge, that are commonly referred to as *knowledge containers*, introduced by Richter and Weber [129] (see Fig. 2.1b). The *vocabulary* contains information about the domain that the CBR application is used in and, thus, is essential to model all kinds of knowledge in the system. The *case base* container consists of experiential knowledge about the problem-solution-pairs that are modeled with the definitions and rules given by the vocabulary. To determine similarities between these cases, similarity measures and all related information such as value ranges, taxonomies, and enumerations are described by the *similarity* container. The *adaptation* container holds information about adaptations of cases in the problem-solving procedure, e. g., rules that modify case attributes according to the query parameters.



(a) 4R Cycle. Source: [1]

(b) Knowledge Containers of Case-Based Reasoning. Source: [132]

Figure 2.1: 4R Cycle and Knowledge Containers of Case-Based Reasoning.

2.1.2 Similarities and Retrieval

In all phases of the CBR cycle, as well as in the CBR methodology as a whole, *similarities* and the similarity knowledge container play a crucial role. The main usage of similarities is the approximation of utility [14, 128] in the sense that the solution of one problem is likely useful for other problems if these problems are similar. This means that similarities are used to identify experiences that are most likely to be useful for solving the current problem. Since these are rather unspecific statements, there are various conceptualizations and concrete implementations of similarity assessment in CBR applications for different types of cases, different types of retrieval algorithms, different kinds of domains, and so on (see Bergmann [14] for an overview). Nevertheless, there are also methods that can be applied in arbitrary similarity assessment procedures. One is the *local-global principle*, introduced by Richter [128], that is also considered in the contributions of this thesis. The principle states to break down the similarity computation between two entities into multiple lower-level similarity computations of individual parts of these entities. For instance, assume these entities to be cases, then the similarity between a pair of cases could be the average similarity value of all individual similarities of the pairs of case attributes. This allows specific similarity measures for parts of the case representation, ultimately, resulting in more reasonable similarities. The local-global principle is intuitive and applicable to many parts of similarity assessment in CBR.

Especially in the *retrieve phase* of the 4R cycle [1], similarity assessment plays a key role. A retrieval procedure is used to find the most similar cases from a case base w. r. t. a given query. These cases are determined by measuring the pairwise similarity between the query and each case of the case base. The cases are then ranked in descending order given the computed similarities. As the performance of a retrieval is determined by the used similarity measure and the performance of the similarity measure mainly by the complexity of the underlying case representation, performance bottlenecks can arise for complex domains and case representations such as process-oriented cases [84, 120, 173]. One way to tackle these issues is the use of multi-stage retrieval methods such as *MAC/FAC* (“Many are called, but few are chosen”), introduced by Forbus, Gentner, and Law [46]. MAC/FAC uses two retrieval stages, where the first stage

(MAC) prefilters the case base with a performant, usually knowledge-light, similarity measure. The remaining cases are then evaluated in the FAC retrieval stage with a possibly slower, usually knowledge-intensive, similarity measure to determine the final retrieval results. The goal is to use a MAC similarity measure that yields comparable results w. r. t. the FAC similarity measure, but has a better performance. Consequently, the method usually introduces a trade-off between performance and quality of the retrieval that is influenced by the selection of the MAC similarity measure.

Similarities are not only used in the retrieve phase of CBR applications, but rather throughout all phases of the CBR cycle. For instance, in the reuse phase, similarities play a key role in learning and applying adaptation knowledge and validating its effects on the adapted case [115, Sect. 2.4.5]. The goal, analogous to the retrieve phase, is to approximate the utility that a retrieved and adapted case provides for solving the current problem situation. The retain phase also uses the same principles as the aforementioned phases w. r. t. similarities [135]. That is, similarities are used to assess the utility of a retrieved, adapted, and revised case as a new member of the case base for efficient long-term problem-solving. A case could be a useful member if, for instance, it contains knowledge that is currently not present in the case base. In terms of similarities, this means that a possibly low similarity to the cases in the case base could indicate a large information gain of adding the case to the case base, and vice versa.

The aforementioned scenarios are only examples where similarities can be and are used in CBR applications. Anyway, they play a crucial role and highlight the importance of novel approaches for optimizing similarity assessment. One example of solutions with an ever-growing popularity use DL methods for similarity assessment [88]. An overview is given in Sect. 2.3.

2.1.3 Process-Oriented Case-Based Reasoning

Due to specific requirements, unique types of knowledge, and individual challenges, subfields of CBR have emerged. One of these subfields is POCBR [15, 112] which is the application context of this thesis. POCBR bridges the gap between CBR and process management. More precisely, the CBR methodology [159] is applied to applications and problems involving process and workflow data, so called *Process-Aware Information Systems (PAISs)* [38], with the goal of optimizing them. The application of CBR to procedural data has been pursued since the early days of CBR research, e. g., Hammond [55] who uses case-based planning for recipe generation. But only since the 2010s it is common to refer to this branch of CBR as POCBR. Compared to classical CBR, POCBR shows differences in the utilized case and domain representations and the similarity assessment.

An example of a POCBR case is given in Fig. 2.2. The figure depicts an excerpt of a cooking recipe represented as a semantically annotated graph [15], which is one possible and also the most common case representation in POCBR. The graph has multiple types of nodes and edges with interrelations that give the topological structure of the recipe, i. e., a baguette is first coated with mayonnaise and then layered with a slice of Gouda cheese to form a sandwich dish. The semantic information is annotated for all nodes and edges, with only one example shown for the node of the task coat. The information states that the duration of this cooking step is two minutes and that the list of auxiliaries contains a spoon and a knife.

The example shows that the effort for knowledge-acquisition is commonly high for such measures, as their definition has to be based on some ground-truth that is usually determined by human domain experts [58]. For instance, a chef could be given several sandwich recipes to decide on which pairs are similar to each other or which cooking steps or ingredients are suitable replacements. But even after this effort and with this information, a concrete similarity measure is challenging to define due to the needed transformation of the expert knowledge to similarities

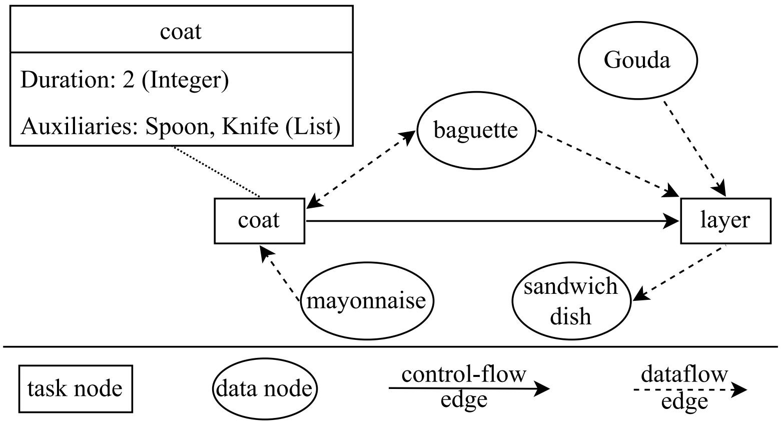


Figure 2.2: Exemplary Cooking Recipe Represented as a Semantically-Annotated Graph. Source: Adapted from [67]

between semantic annotations, similarities between different types of nodes and edges, similarity assessments of the graph structure, and so on. The costs are even higher if, for instance, experienced experts are not available or documented knowledge is outdated.

The second challenge is the computational performance of these measures. Most similarity measures for semantic graphs rely on some form of subgraph isomorphism check. This means that the similarity value is determined w. r. t. the number of nodes and edges with a one-to-one correspondence in both graphs. There are different ways of finding these correspondences. A widely used method is to use an A*-based search [15, 173] where pairs of nodes and edges are mapped onto each with the goal of finding the maximum similarity for the entire set of nodes and edges of both graphs. The algorithm is iterative, and pairs of nodes and edges are mapped one by one until all nodes of one graph are mapped. There are different heuristics [173] to guide the search process towards an optimal solution, with the common strategy of selecting the next node pair as the one with the maximum pairwise similarity. An A* search or other subgraph isomorphism checks usually have the advantage of a high explainability due to the clear correspondences, i. e., mappings, that lead to the final similarity value. On the other hand, these measures are computationally expensive, usually with polynomial or exponential complexity [15, 21, 120] and can lead to performance problems.

Those two challenges are present in many POCBR applications. Improving in these regards motivates approaches using DL in POCBR applications (see Sect. 2.4.1), as DL can generally provide self-learning with a reduced manual acquisition effort and increased performance (e. g., [67, 71]).

2.1.4 Software Libraries

The implementation of CBR approaches in software systems is usually done with libraries that provide key functionalities of these approaches, such as data representations and retrieval algorithms. In a recent paper, Schultheis, Zeyen, and Bergmann [144] give an overview and compare CBR software libraries used in research and practice. The libraries are analyzed w. r. t. the supported knowledge containers, the phases of the CBR cycle, the provided interfaces, and other specific features. It is concluded that no library covers all aspects of CBR applications, and most of the compared libraries are specifically tailored towards certain use cases.

One of the compared frameworks is ProCAKE [16], which is developed at the University of Trier[1]. ProCAKE focuses on creating structural and procedural CBR applications, making it the de-facto standard library for POCBR. ProCAKE supports integrated process and knowledge management by offering a domain-independent platform with several data types for knowledge modeling, various syntactic and semantic similarity measures, and multiple retrieval algorithms like k-NN and MAC/FAC [46]. ProCAKE is implemented in Java and configured via XML, making it adaptable for diverse CBR applications.

2.2 Deep Learning

DL [92] is a subfield of ML and AI, characterized by the usage of *deep neural network models*. Neural networks are not a new technology, and the first ideas date back to the 1940s in an effort to model the structure of human brains. With growing amounts of storage capacity and data as well as powerful computation hardware, DL models achieved state-of-the-art results in many ML and AI problems such as object detection [175], question answering [20], and machine translation [9]. The main benefit of using DL approaches over symbolic AI methods is their *automatic learning capability* [51, 92]. It allows training models to solve a specific problem only from data, with little manual engineering effort. Further, the model can be improved by repeating the training procedure if new data is available.

2.2.1 Training Deep Neural Networks

Deep neural networks learn representations of data with multiple levels of abstraction. These levels are modeled by interconnected layers of neurons, each performing a simple numeric transformation of input into output data. Starting with the input data to the network itself, e. g., a sentence, the neural network transforms the input layer-wise into its final representation, e. g., the translation of this sentence.

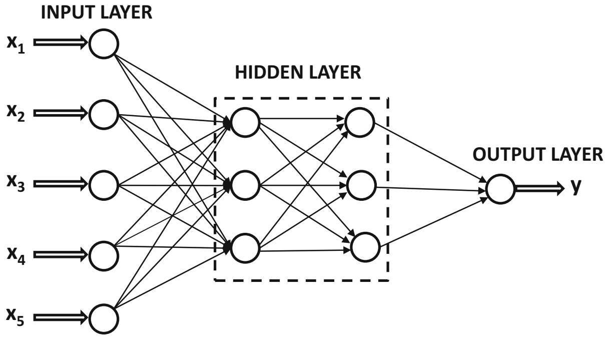


Figure 2.3: Basic Architecture of a Deep Neural Network. Source: [3, p. 19]

This basic architecture of a so-called feed-forward neural network of input layer, hidden layers, and output layer is depicted in Fig. 2.3 and is generally present in all neural networks. In this example, the neural network is fed with five input features (x_1 to x_5) to predict a single output value (y). The data flow starts at the input layer, goes through the hidden layers, and ends in the output layer, with each neuron having connections to all neurons of the preceding layer and all neurons of the following layer. The resulting topology of the neural network forms a

[1] https://procake.uni-trier.de/

mathematical function with the inputs as the arguments and the outputs as the value of the function. Each connection between neurons is a trainable weight and acts as a parameter of the function. This means that these weights can be adjusted in a training procedure to compute output values that match the distribution of the training examples.

The *training* of a neural network and the mathematical function it represents, respectively, is performed by a combination of *Stochastic Gradient Descent (SGD)* and backpropagation. Starting from a pair of input data and the corresponding label, the first step is to wrap the function in a loss term. The loss term represents the error that is involved in the computation of the neural network's output values, i. e., predictions, w. r. t. to the label. A higher loss stands for worse predictions, and vice versa. To steer the optimization, derivatives of the error function are computed and then backpropagation is applied repeatedly from the output layer backward to the input layer. Based on these derivatives for all layers and neurons in the neural network, the model can be optimized by SGD. Doing multiple iterations of feeding the network with training data, computing the loss term, and optimizing the network parameters based on the loss term eventually leads to a trained model with sufficient prediction accuracy.

The setup of a robust and efficient training pipeline is a common challenge for DL applications, especially regarding the amount of training data and the model complexity. On the one hand, the training should be fast to minimize computation time for new versions of a model or during experimentation. On the other hand, the training should be robust and lead to a well-trained model with good validation results. This tradeoff is addressed with many different techniques and approaches, such as regularization methods (e. g., [148]), monitoring of the training progress (e. g., [113]), and advanced optimizers (e. g., [83]). Another common problem of DL training is an insufficient amount of training data. A popular approach that addresses this problem is data augmentation [40]. Training deep learning models effectively usually requires vast amounts of data to capture the variability and complexity of real-world scenarios. However, acquiring such extensive datasets can be challenging and expensive. Data augmentation helps mitigate this issue by artificially expanding the size and diversity of the training data. Data augmentation techniques are especially prominent for image processing models that apply rotations, translations, flips, and color adjustments, to increase the amount of training data. The new, unique training examples enhance the model's ability to generalize and improve its performance on unseen data, ultimately leading to more robust and accurate models.

Another method to handle problems regarding training data is *Transfer Learning (TL)* [86, 152] with the core idea of reusing existing trained (parts of) neural networks in subsequent training procedures. In most cases of TL, an existing pretrained model is fine-tuned to perform another, usually related, task w. r. t. the original model. For instance, an existing model that was trained on general text documents for the task of generating new text could be fine-tuned with legal documents to generate more useful texts in legal application scenarios. Due to the integration of an already trained neural network, the training complexity can be reduced significantly. TL also allows models trained on large amounts of data to be more accessible to researchers and companies with limited computation budgets, ultimately increasing expressiveness and robustness.

To use concepts and ideas of new model architectures or training paradigms in practice, several software libraries enable the definition and the training of neural networks. Erickson *et al.* [41] list several libraries, of which the following text introduces the most widely used ones: *TensorFlow* [2], developed by Google, provides a flexible and comprehensive ecosystem around DL, supporting both research and production environments. *PyTorch* [121], created by Facebook's AI research lab, uses a dynamic computation graph and focuses on ease of use, making it a popular choice among researchers. *Keras* [30], which can run on top of multiple computation frameworks, provides a high-level API for quick prototyping and experimentation. Therefore, it is likewise suited for both less-experienced and experienced users. Additionally, *MXNet* [24], backed by Apache, is known for its scalability and efficiency, particularly in distributed computing environments. All of these libraries share the goal of making the implementation and, thus, the use of DL models in software systems as straightforward as possible.

2.2.2 Deep Neural Network Architectures for Sequential and Grid-Structured Data

An important consideration in practice and an active topic in research is the most suitable *neural network architecture* for a specific application context and task. While there are many important factors that influence this decision, it is common to differentiate between different architecture classes based on the dependencies between the input values [3]. The neural networks in these classes contain specific hidden layers that consider the dependencies between the input values. The following text introduces *Recurrent Neural Networks (RNNs)* as an exemplary neural network architecture for sequential data and *Convolutional Neural Networks (CNNs)* as a possible neural network architecture for grid-structured data.

Recurrent Neural Networks

RNNs [3, ch. 7] are used to process data with sequential dependencies between input features. Such dependencies occur in many kinds of data but, most notably, in time series data, e. g., sequences of sensor readings over time, and text data, e. g., sentences in news articles. RNNs aim to directly reflect the sequential dependencies of the input data in the architecture of the neural network, and especially, of the hidden layers. In contrast to standard feed-forward neural networks (see an example in Fig. 2.3), RNNs implement information flow between neurons of the same layer. This is achieved by "unrolling" the sequence in time and passing a state between features of the time series. This means that each feature of the time series is processed by trainable neural network components and, in addition, the processed input feature is combined with the state of the sequence that is passed from the preceding feature in the time series. The result is either a new sequence of processed feature vectors, i. e., sequence-to-sequence model, or a single vector of processed features, i. e., sequence-to-vector model, depending on the use case. There are different types of RNNs with variations of how the time series is processed, e. g., the *Long Short-Term Memory (LSTM)* [64] and the *Gated Recurrent Unit (GRU)* [29]. A common drawback of RNNs are performance limitations due to the sequential processing order of the data. As the state of the sequence is passed from one feature to the next, the features can only be processed sequentially and not in parallel, which limits the capabilities of modern, parallelized computing infrastructures for neural networks.

Convolutional Neural Networks

CNNs [3, ch. 8] are commonly used to process grid-structured input features with two-dimensional dependencies. The most prominent examples are images that consist of individual pixels as input features with strong relations between neighboring pixels. The name CNN comes from the convolutions that are used to process the two-dimensional input features. Convolutions are operations that multiply the color values of a small grid of neighboring pixels with a filter matrix of the same size and then reduce the result to a single output value. In the context of neural networks, these filter matrices are composed of trainable neurons to convolve images according to the training goal. The convolutions exploit the two-dimensional dependencies between neighboring pixels and combine their information, extracting features of the image layer-by-layer. The typical setup of a CNN is to have a sequence of convolutional layers that gradually reduce the size of the input data in the beginning, followed by a feed-forward neural network to produce the desired prediction.

There are numerous variants of CNNs with different strengths and weaknesses. A comprehensive overview of different CNNs and the progress of neural image processing over the last few years is evident from the annual ImageNet large-scale visual recognition challenge.[2]

2.2.3 Graph Neural Networks

Graph Neural Networks (GNNs) (see a comprehensive introduction in [11, 157, 166]) are a special class of neural network architecture, as they are applied to graph-structured data and combine a multitude of other architectures such as RNNs and CNNs. Similar to the aforementioned classes, GNNs also deal with dependencies between input features. These dependencies arise from the graph structure of nodes being connected by edges, where each node and edge is characterized by features. In general, GNNs consider these dependencies by implementing an information flow between nodes along the edges. This means that each node has a representation that is updated iteratively via information propagation from other nodes that are connected via an edge.

Literature Surveys

There are several *surveys* that categorize GNN literature and the respective variants according to several criteria. In the following, three of these surveys with different views on the comparison of GNN literature are examined: Wu *et al.* [168] present a taxonomy of four classes of GNNs, i. e., recurrent GNNs, convolutional GNNs, graph autoencoders, and spatio-temporal GNNs. They analyze applications of these classes of GNNs based on the addressed task, i. e., whether it is node-level, edge-level, or graph-level, and the used training procedure, i. e., semi-supervised, supervised, or unsupervised. Besides the theory behind the individual approaches, available open-source implementations as well as benchmark datasets are also discussed.

Zhou *et al.* [176] also discuss several methods of GNNs. They categorize the methods based on the types of graphs the methods support, the used training method, and the method of information propagation. Compared to Wu *et al.* [168], the comparison is on a more technical level that is focused on the internal aspects of the GNN methods. In addition, application scenarios of the approaches such as biology, chemistry, and *Natural Language Processing (NLP)* are discussed.

At the heart of the survey of Waikhom and Patgiri [157] is the learning task of the compared GNN methods. They comprehensibly distinguish between supervised, semi-supervised, self-supervised, and few-shot learning. Additionally, they give extensive foundations on the topic and also recommendations on how to approach future learning tasks with GNNs. Besides these generic surveys about GNNs, there are also papers specific to one domain or use case, e. g., explainability [172], recommender systems [167], and traffic forecasting [77].

The Graph Embedding Model and the Graph Matching Network

Among the various GNN architectures discussed in the literature, the work by Li *et al.* [97] serves as the foundational model for the used GNNs in this thesis. Their approach is particularly relevant because they explicitly describe and assess it in the context of similarity-based search, which closely aligns with similarity assessment and retrieval tasks in POCBR. The neural network architecture follows the basic structure of a *Message Passing Neural Network (MPNN)* [54] and consists of four key components, which are connected sequentially (as depicted in Figure 2.4).

These components are:

[2] https://www.image-net.org/challenges/LSVRC/index.php

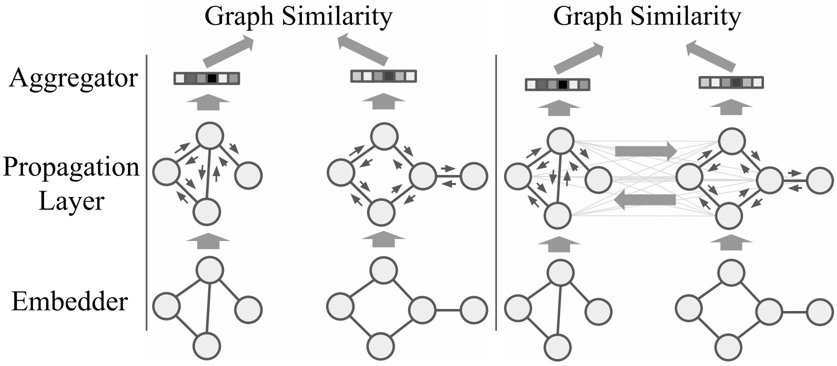


Figure 2.4: Graph Embedding Model (GEM; left) and Graph Matching Network (GMN; right). Source: [142]

1. Embedder: The embedder transforms raw graph input data into an initial vector representation. It generates a single embedding vector for each node and edge in the graph.
2. Propagation Layer: During propagation, node information represented as vectors is iteratively merged based on the graph's edge structure. Node embedding vectors are updated by element-wise summations, incorporating information from incoming edges and connected nodes. This process captures local neighborhood information for each node.
3. Aggregator: After multiple rounds of information propagation, the aggregator combines the embeddings of all nodes within each graph. The result is a single whole-graph embedding.
4. Graph Similarity Component: This final component computes a scalar similarity value using the embedding vectors of both graphs. It employs a vector similarity measure, such as cosine similarity.

The key distinction between the *Graph Embedding Model (GEM)* and the *Graph Matching Network (GMN)* lies in their propagation strategy. The GEM employs an isolated propagation approach, propagating information only within a single graph. In contrast, the GMN incorporates an attention-based cross-graph matching component, allowing information to propagate across both graphs during the early stages of similarity assessment. Please note that the main purpose of both models is to create embeddings of input graphs and both models can, thus, also be used without the graph similarity component.

Software Libraries

Usually, research papers that propose new GNN architectures or training paradigms also come with an open-source *implementation* of their approach. To reuse common components for GNN implementations and to accelerate the development process, various ecosystems of software libraries exist: The *Deep Graph Library (DGL)* [158] is a Python library specifically designed for DL with graphs and GNNs. It is agnostic of the underlying computation library, e. g., TensorFlow [2] or PyTorch [121] and comes with various extensions. PyTorch Geometric [45] extends the PyTorch ecosystem by graph learning functionality. Several research papers are already implemented into the library to simplify their usage. TensorFlow GNN [43] was only recently released and focuses on making the TensorFlow ecosystem more accessible for GNN tasks. The library also has built-in implementations of research papers and reuses several concepts of the older library graph_nets [11].

2.2.4 Hyperparameter Optimization

The way of conceptualizing, implementing, and training a DL model involves numerous configuration possibilities that influence the outcome, so-called *hyperparameters*. Most hyperparameters are part of the training procedure or the model architecture, e. g., the learning rate of SGD, the number of neurons in each hidden layer, and, on a higher level, even the selected neural network architecture itself. Finding the right configuration is very challenging and, thus, it is common to automatically *optimize these hyperparameters* [44] as part of the training procedure. The process is built around a simple equation:

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}) \tag{2.1}$$

The optimal hyperparameter vector $\lambda^* \in \Lambda$ is searched for by selecting hyperparameter vectors $\lambda \in \Lambda$ that minimize the loss $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D})$. The loss value results from training $\mathcal{A}_\lambda$ and validating its performance on the dataset $\mathcal{D}$.

Different HPO methods provide different ways of finding the optimal hyperparameter vector (argmin). This is necessary as the search space of all possible hyperparameter vectors grows exponentially with the number of hyperparameters, making exhaustive search infeasible in practice. Besides basic strategies, such as randomly selecting different hyperparameter vectors [18], there are also much more elaborate methods [42].

2.3 Hybrid Approaches of Case-Based Reasoning and Deep Learning

Using *hybrid approaches of DL and CBR* is a prominent research topic in the community with many approaches, including the approach proposed in this thesis. These approaches utilize DL methods to improve CBR applications and the complementary direction is pursued where the CBR methodology is used to improve on DL shortcomings [87, 88]. Thereby, both research directions aim to combine the strengths of each of the fields to overcome the respective weaknesses based on the concept of neuro-symbolic AI [51, 138]. The resulting system should be robust and expressive (DL properties) as well as explainable and correct (CBR properties). The challenge is to achieve this while overcoming the respective challenges [88] of, for instance, the need for large amounts of training data in DL and the knowledge-acquisition costs of CBR. The remainder of this section discusses approaches from the intersection of CBR and DL research regarding, among others, similarity learning, case adaptation, and explainability.

Several works [5, 6, 7] examine the combination of DL and CBR in NLP tasks. The main artifact of the work is the *Deep Knowledge Acquisition Framework (DeepKAF)* that targets similarity computation and retrieval of CBR applications. Specifically, the framework replaces conventional similarity measures for textual data with automatically learned neural networks, i. e., autoencoders, word embeddings, and *Siamese Neural Networks (SNNs)*. The models are also periodically retrained based on user feedback. Another branch of work [108, 109, 110] also deals with the topic of *similarity learning* by neural networks to support CBR applications. The domain these methods are applied to is decision support for aquaculture sites. The decision support is based on SNNs that learn similarities between different aquaculture sites. Martin *et al.* [107] employ convolutional SNNs to learn similarity measures for image data. The paper thereby combines CNNs and SNNs to process image data effectively and to learn a similarity measure from the data. These methods broadly fall into the category of feature extraction with DL methods, which is also popular in POCBR research. Other examples of this category are Wilkerson, Leake, and Crandall [163], who aim to improve on the retrieval of cases by not only using manually engineered but also DL-extracted features, and Marie *et al.* [105], who use CNNs for feature extraction in the medical domain.

Other approaches examine DL methods as a means for *case adaptation*. Liao, Liu, and Chao [99] base their method on the case difference heuristic [57] that determines the difference of the solution of two cases by the difference of the problem descriptions of these two cases. In their approach, a DL model predicts the solution difference given the problem difference of a pair of cases. The approach is evaluated on a regression task with rather simple numeric features. Other groups of researchers [90, 91, 171] also use the case difference heuristic in combination with DL to perform case adaptation. Leake, Ye, and Crandall [91] and Leake *et al.* [90] extend the work of Liao, Liu, and Chao [99] by training a neural network not only on the problem difference but also on the retrieved case to provide additional context. Ye, Leake, and Crandall [170] take a very similar approach but tackle classification problems instead of regression problems.

eXplainable Artificial Intelligence (XAI) is another prominent branch of work at the intersection of CBR and DL. This time, CBR is used to improve DL applications that usually lack explanations for their predictions (see, for instance, [10, 48, 52, 80] and several workshops at the International Conference on CBR on this topic). For instance, Keane and Kenny [80] discuss twin systems of neural networks and CBR methods that aim to improve the explainability of the neural networks with post-hoc explanations based on the CBR component. Another example is the work of Barnett *et al.* [10], who present an interpretable classification method of mammography images that uses CNNs to extract features from the images and a CBR component to interpret these features in a way that experts can validate the predictions. They notice that explainability is key in this domain, as the goal is to assist clinical experts in their decision-making process of the patient's treatment. It is also worth mentioning that the approach does not use specific cases, but rather learned prototypes that resemble cases in their purpose.

2.4 Deep-Learning-Based Approaches for Process and Workflow Management

DL-based approaches for process- and workflow management are discussed for the research areas of POCBR and BPM. Both disciplines deal with procedural data at their core but, apart from that, most research is done independently.

2.4.1 Deep Learning Methods in Process-Oriented Case-Based Reasoning Applications

The motivations of using DL methods in POCBR are largely the same as for CBR: trainable models combined with trustworthy, symbolic expert knowledge. The typical challenges of POCBR applications that are addressed by DL methods, however, can be more "serious" compared to those of CBR applications. For instance, DL models can be used for feature engineering or -extraction (e. g., [90, 105, 137]) of cases from a case base. Since the complexity of typical POCBR case representations in the form of semantically annotated graphs [15] is high, the motivation to use automatic learning methods instead of human experts for engineering features is even higher [67, 84]. The same observation can be made when learning similarity measures or adapting cases. Despite large potential but because the number of POCBR researchers is limited, related work that uses DL methods in POCBR applications is rather scarce. The discussed approaches aim to improve retrieval, case adaptation, or similarity assessment by incorporating DL methods.

Leonardi, Montani, and Striani [95] investigate classification approaches for medical process traces. They compare two metrics for similarity assessment, a semantic measure and one based on a DL model. The DL approach uses an RNN to extract features from the process traces and computes a vector space similarity between these feature vectors as a similarity value. The approach treats the traces as sequences of activities and only uses the activity labels as features, keeping the complexity low. Another use case of *feature extraction* for similarity assessment is the work of Klein, Malburg, and Bergmann [84]. They employ a general-purpose embedding framework called StarSpace [165] for unsupervised learning of vector representations used as case features. The framework leverages the structural properties of semantic graphs, including the relationships between task and data nodes. The approach is evaluated as a similarity measure in retrieval scenarios and is well-suited for retrievals with multiple stages, such as MAC/FAC [46, 82], where a fast but inaccurate retriever is paired with a slow but accurate one. Lenz *et al.* [94] also use DL models, specifically NLP methods, to transform complex textual features into embeddings in the domain of argumentation. The embeddings are used in similarity measures, incorporating the specific characteristics and features of the underlying arguments.

DL methods are also discussed in the context of *case adaptation*, which is relevant to the reuse phase of CBR applications [1]. One of the more recent publications in the domain of argumentation [93] extends the initial approach and includes *Large Language Models (LLMs)* as a means for the adaptation of argument graphs. Brand *et al.* [19] also investigate case adaptation and present a conceptual approach for adapting semantic processes with a GNN that is used by a *Reinforcement Learning (RL)* agent. Inspired by change patterns [160] for processes, the agent learns to delete, insert, and replace parts of a process to contribute to increasing the query fulfillment of a retrieved case. The approach can either be used to learn or create a base of adaptation knowledge, or to adapt cases directly. Nevertheless, its effectiveness has yet to be examined in an empirical evaluation.

2.5 Similarity Learning and Deep Learning in Business Process Management

Although the thesis is set in the context of the POCBR research field, related work is not limited to POCBR and CBR approaches. As POCBR applies CBR methods to BPM, approaches from BPM literature that use DL methods or deal with similarity learning are related to this thesis.

Similarity learning between processes is often denoted as *business process matching* [23, 162] in this research community, i. e., finding correspondences between two processes to determine their consistency. For further reading, a comprehensive overview of similarity measures for processes is presented by Schoknecht *et al.* [140]. Sonntag *et al.* [147] introduce an ML approach for NLP-based similarity measures between process models. In this method, the node labels of both processes are matched, and each node consists of multiple tokenized words. Learning is done by using a local search to find the optimal hyperparameter vector. The approach of Niesen *et al.* [119] also uses methods from NLP in business process matching. Specifically, they match in multiple stages: 1) Identical node labels are matched. 2) Identical lemmatized node labels are matched. 3) The vector space model is used to extract statistics from the textual node labels to form a vector representation of the process nodes, and close representations are matched.

Another discussed use case for business process matching is identifying *merge possibilities* between multiple process models. Morrison *et al.* [114] aim to generalize processes where large parts of multiple processes can be matched. Dijkman *et al.* [36] approach refactoring possibilities in process repositories by matching processes onto each other. Rosa *et al.* [130] pursue a similar goal and use graph edit distances [50] to determine which parts of the processes can be matched. Kacimi, Tari, and Kheddouci [78] split up processes into multiple sequences of tasks, called paths. These paths are transformed to a signature by encoding each unique node (XOR, AND, etc.) and concatenating these encodings along the path. Similarities between these signatures based on simple string similarity measures are then aggregated to similarities for the processes. More modern approaches such as Sungkono *et al.* [151] also merge processes by computing textual similarities with NLP transformers such as BERT [34].

There are other approaches that use DL in BPM research, e. g., for general-purpose representation learning [125], process mining [56], predictive process monitoring and prediction [117, 126], and process generation [39]. However, their applicability to POCBR is limited due to numerous assumptions on the complexity of the processes that mostly only consider simple labeled graphs. GNNs are also in use for BPM tasks (e. g., [49, 59, 155]) but the limitations to simple processes or graphs also apply there.

Chapter 3

Contributions

Building on the foundational concepts and existing literature discussed in the previous chapter, this chapter contains the core of this thesis, i. e., the formulation of research questions and the contributions that emerge from the thesis. There are three research questions, addressing the goal of hybrid similarity assessment in POCBR applications, that are specifically derived from three formulated challenges of hybrid DL and POCBR approaches (see Sect. 3.1). Following the research questions is the presentation of the contributions of the thesis, with an introduction of the research methodology and a presentation of an overview (see Sect. 3.2). Following is a more detailed explanation of the three developed research artifacts (see Sect. 3.3 – 3.5). The chapter is closed with a brief overview of the implemented software tools (see Sect. 3.6) and a summary of all conducted experimental evaluations (see Sect. 3.7).

3.1 Research Questions

The presentation of the scientific landscape reveals *challenges* that arise for hybrid applications with POCBR and DL components. The following three challenges are identified by the author based on the presented literature and his personal experience of working with POCBR and DL methods:

1. **Data Complexity**: The data required to operate a POCBR application is usually very complex. The cases are represented as semantically annotated graphs [15] with a complex topology of nodes and edges. The complex, procedural nature of the cases is also reflected in other types of knowledge used in the application, such as the modeled domain knowledge or the applied similarity measures. While this complexity with the associated knowledge-acquisition bottleneck [58] is one of the motivations for using DL models to avoid problems such as a computationally expensive similarity assessment (see Sect. 2.1.3), it also poses a challenge for the DL models themselves to include this knowledge into the neural networks: All POCBR data used as input or output data for the DL models has to be numerically encoded to enable computations by the neural network. The architecture and training method of the neural network also have to be capable of processing the complex input data effectively. Both aspects refer to a general challenge of neuro-symbolic AI [133, 138], that is, to transform, merge or consolidate symbolic and non-symbolic data and the associated representations.

2. **Data Scarcity**: Another challenge is scarce data, which can be a problem in POCBR applications. The reason for this is the focus on examples of processes as the main training

input: On the one hand, processes are usually manually modeled and, because of limited resources, examples of processes are not available in large quantities. On the other hand, process data is generally not as ubiquitous as data from other domains, such as natural language or images, which further limits the amount of available data. For instance, when looking at the experiments of all the papers presented in Sect. 2.4.1, there is an average of 1331 cases that serve as training data. This is a tiny amount compared to other well-known datasets for training DL models, e. g., 60000 cases for MNIST[1], 50000 for CIFAR-10[2] or multiple terabytes of data for training LLMs [20]. An integration of DL methods in POCBR needs to be capable of dealing with small amounts of data.

3. **DL Design and Configuration**: The main idea of this thesis is integrating DL methods in POCBR applications, with a focus on improved similarity assessment and retrieval. As stated in Sect. 2.2, the success of DL is dependent on designing a proper setup, including model architecture, training method, and hyperparameter configuration. It is important to optimize every aspect of this setup individually and to find an overall optimal setup that considers trade-offs between different facets of the examined problem. One important trade-off to consider is the relation between prediction quality of the model and its inference runtime (see Sect. 2.1.3). Both properties are important for DL-based similarity measures in POCBR applications because, on the one hand, the predicted similarities need to be precise to replace manually modeled similarity measures and to ultimately reduce knowledge-acquisition effort. On the other hand, the model inference time, i. e., the time to predict a similarity, should be as short as possible to diminish performance bottlenecks in POCBR applications existing at the current time. For a more detailed analysis and quantification of these bottlenecks, the reader is referred to Ontañón [120] and Zeyen and Bergmann [173].

The following *research questions* are investigated under consideration of the aforementioned challenges:

RQ1 How can the complex symbolic data present in POCBR applications, i. e., the case base, the domain knowledge, and the similarity knowledge, be numerically encoded to be used for training DL models?

RQ2 Which DL setups (architectures, training procedures, configurations) are suitable for training DL models for similarity assessment and case retrieval in POCBR?

RQ3 How can possible limitations caused by a limited number of labeled training cases in POCBR be mitigated when training DL models?

3.2 Research Methodology and Overview

As stated in the introduction, the thesis follows the principles of DSR [62, 63] (see Fig. 3.1). DSR is a research methodology that assumes knowledge can be produced by designing and implementing artifacts that address practical problems. Such an artifact can be any construct, model, method, or system that aims to solve an identified issue. The methodology is structured around three interconnected cycles, i. e., the *Relevance Cycle*, the *Rigor Cycle*, and the *Design Cycle*.

[1] https://www.tensorflow.org/datasets/catalog/mnist

[2] https://www.tensorflow.org/datasets/catalog/cifar10

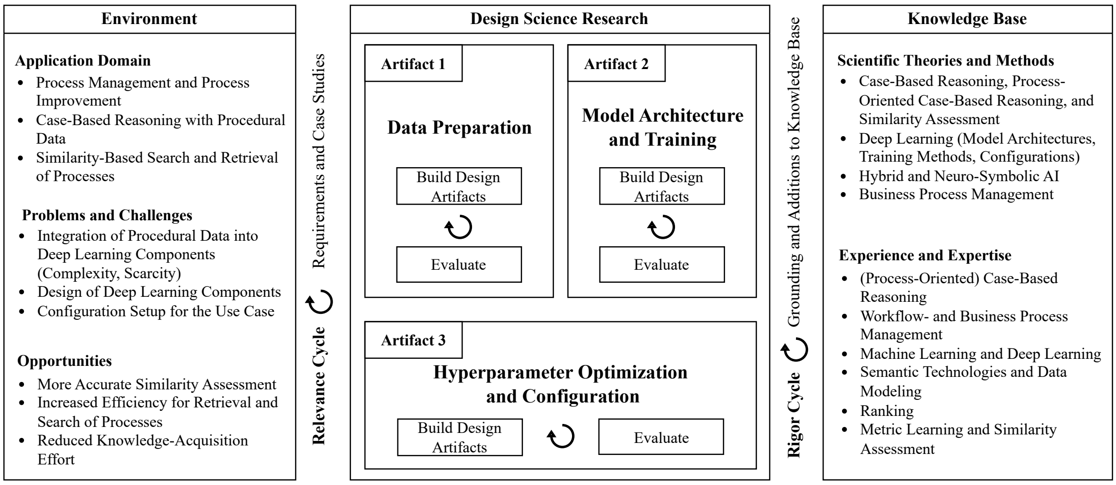

Environment
Application Domain
Process Management and Process Improvement
Case-Based Reasoning with Procedural Data
Similarity-Based Search and Retrieval of Processes
Problems and Challenges
Integration of Procedural Data into Deep Learning Components (Complexity, Scarcity)
Design of Deep Learning Components
Configuration Setup for the Use Case
Opportunities
More Accurate Similarity Assessment
Increased Efficiency for Retrieval and Search of Processes
Reduced Knowledge-Acquisition Effort
Relevance Cycle
Requirements and Case Studies
Design Science Research
Artifact 1
Data Preparation
Build Design Artifacts
Evaluate
Artifact 2
Model Architecture and Training
Build Design Artifacts
Evaluate
Artifact 3
Hyperparameter Optimization and Configuration
Build Design Artifacts
Evaluate
Rigor Cycle
Grounding and Additions to Knowledge Base
Knowledge Base
Scientific Theories and Methods
Case-Based Reasoning, Process-Oriented Case-Based Reasoning, and Similarity Assessment
Deep Learning (Model Architectures, Training Methods, Configurations)
Hybrid and Neuro-Symbolic AI
Business Process Management
Experience and Expertise
(Process-Oriented) Case-Based Reasoning
Workflow- and Business Process Management
Machine Learning and Deep Learning
Semantic Technologies and Data Modeling
Ranking
Metric Learning and Similarity Assessment

Figure 3.1: Design Science Research Methodology. Source: [62]

The Relevance Cycle focuses on the connection between the research and the real-world context, i. e., the *environment*. It ensures that the problems addressed by the research and the created artifacts are grounded in practical needs and applicable to real-world problems. The main tasks of the Relevance Cycle can be divided into tasks that precede and prepare the development of an artifact and tasks that apply and validate a developed artifact in the environment. Tasks of the former category are concerned with problem identification and definition of requirements to guide the artifact development. If an artifact is available, it is applied and evaluated in a real-world context through pilot projects, case studies, or field trials to observe its effectiveness and to gather feedback.

The Rigor Cycle ensures that the research is scientifically sound and contributes to the existing body of knowledge, i. e., the *knowledge base*. It connects the research process to the established theoretical foundations and defines methods for both designing and evaluating the artifact. It ensures that the research not only contributes a practical solution but also advances theoretical understanding. The key tasks of this cycle include the compilation of a knowledge base that provides the foundation for understanding the problem, designing the artifact, and selecting the evaluation methods, the selection of scientifically validated methods to ensure that the research adheres to the scientific state-of-the-art, and the conceptualization of the artifact's evaluation to enable the validity and reliability of the artifact.

The Design Cycle is the core process within DSR, focusing on the actual development and refinement of the artifact [123]. This cycle is iterative and, in each iteration, the researcher builds an initial solution, tests it, gathers feedback, and improves the artifact further. By emphasizing continuous iteration, the Design Cycle ensures that the artifact becomes increasingly effective and adaptable to real-world scenarios. The iterations end if the artifact solves the initially identified problems and meets the requirements. The Design Cycle acts as the connection between all three cycles in DSR, thus bringing together the current state-of-the-art (Rigor Cycle) with the real-world environments (Relevance Cycle).

As part of this thesis, three artifacts have been developed. These artifacts are interrelated and embedded into a cohesive framework that can be used for hybrid POCBR applications with DL-based similarity measures and retrieval functionality. Each of these artifacts consist of a theoretical method such as an algorithm or a neural network architecture in combination with the corresponding implementation. Each artifact has been developed according to the principles

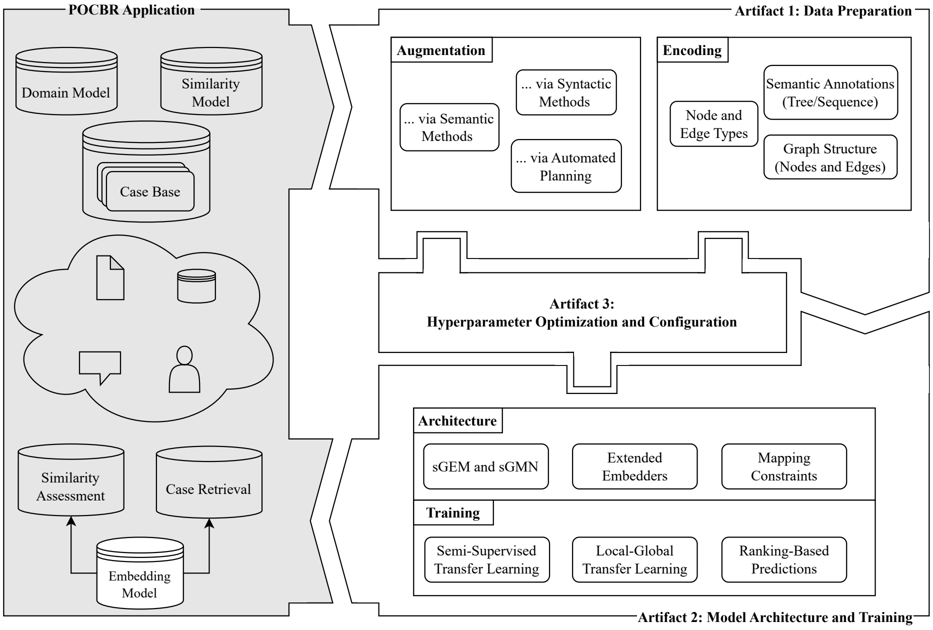

POCBR Application
Domain Model
Similarity Model
Case Base
Similarity Assessment
Case Retrieval
Embedding Model
Artifact 1: Data Preparation
Augmentation
... via Semantic Methods
... via Syntactic Methods
... via Automated Planning
Encoding
Node and Edge Types
Semantic Annotations (Tree/Sequence)
Graph Structure (Nodes and Edges)
Artifact 3: Hyperparameter Optimization and Configuration
Architecture
sGEM and sGMN
Extended Embedders
Mapping Constraints
Training
Semi-Supervised Transfer Learning
Local-Global Transfer Learning
Ranking-Based Predictions
Artifact 2: Model Architecture and Training

Figure 3.2: Overview of the Proposed Approach.

and procedures of DSR, as introduced before. Further, each of the three artifacts has one or more associated publications. Figure 3.2 visualizes this relationship. Table 3.1 gives an overview of the individual publications that are part of the thesis, with their main contributions as well as their associated artifacts.

Artifact 1 (see Sect. 3.3) consists of methods for *training data encoding* and *semantic graph augmentation* alongside their respective implementations. Training data encoding refers to the procedure of mapping training examples from an arbitrary data representation, i. e., semantic graphs in this thesis, to a numerical multidimensional vector space. This procedure is necessary since DL models can only operate on numeric data. The developed encoding scheme [67] handles the key components of processes represented as semantic graphs, that is, the structure with nodes and edges in between, the types of nodes and edges, and the semantic annotations. Especially the semantic annotations are challenging to encode due to their inherent complexity, featuring multiple types of data, tree-like compositions, and further information from the underlying domain. The second part of artifact 1 consists of augmentations of semantic graphs to enlarge the training data set [70]. Data augmentation [40] (see Sect. 2.2.1) is popular in DL research to mitigate issues with small training data sets, which is usually also a problem with POCBR data. Different categories of augmentation methods are examined with different levels of correctness. The level of correctness of an augmentation method increases if the augmented training example preserves either the syntactic structure, the semantic structure, or both aspects of the original, non-augmented case.

Given sufficiently large and numerically encoded training data, Artifact 2 (see Sect. 3.4) consists of several GNN architectures paired with corresponding training procedures and implementations for learning pairwise semantic graph similarities. During the research, several architectures that provide, among other aspects, specific layers for embedding the semantic an-

Table 3.1: Overview of Publications.

Ref./ Ch.	**Title**	**Main Contributions**	**Arti-fact(s)**	**RQ(s)**
[67] / 5	Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning	• Comprehensive encoding scheme of semantic graphs • sGEM and sGMN models for generating embeddings of semantic graphs	1, 2	1, 2
[70] / 6	Augmentation of Semantic Processes for Deep Learning Applications	• Process augmentation methods for enlarging the amount of training cases • Semi-supervised TL procedure with an unsupervised pretraining phase and a consecutive supervised training phase	1, 2	2, 3
[122] / 7	Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning Using Graph Neural Networks and Transfer Learning	• TL approach for learning local and global similarities • Different variants of node and edge embedders, using RNNs, MLPs, and attention-based neural networks	2	2
[68] / 8	Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning	• Extension of the training procedure and the architecture of sGEM and sGMN for predicting pairwise rankings • Rank aggregation algorithms to use pairwise rankings in case retrieval	2	2
[66] / 9	Improving Automated Hyperparameter Optimization with Case-Based Reasoning	• CBR-based decision-making procedure for extending existing state-of-the-art HPO methods • Continuous similarity assessment and retrieval of hyperparameter vectors	3	2

notations and types of semantic graphs [67] and an architecture for employing *Learning To Rank (LTR)* functionality [68] have been examined. To make use of the model architectures, artifact 2 also consists of methods for model training. Several training procedures are proposed for the use case of similarity learning between semantic graphs: TL for pretraining local similarity models [122], semi-supervised TL for training with augmented, unlabeled data [70], and training with ranking-based loss functions [68].

Artifact 3 (see Sect. 3.5) contains an algorithm and its implementation for doing HPO [31, 44] of the models resulting from the second artifact. This optimization is crucial to adapt the hyperparameters of the training procedure and the model architecture to the CBR application that the trained model is used in. The corresponding approach *Hyperparameter Optimization with Case-Based Reasoning (HypOCBR)* [66] extends other state-of-the-art HPO methods by a decision-making process, which aims to only consider hyperparameter vectors for an optimization run if they appear promising. This decision is based on an underlying experience-based system that compares the current hyperparameter vector to already seen hyperparameter vectors and their optimization results. The similarity measures used to compare these vectors are defined manually and incorporate the experience of DL practitioners about known-good hyperparameter configurations.

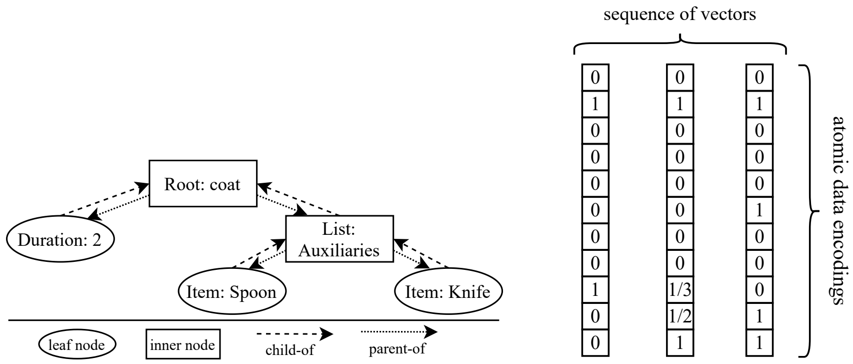
In the following sections 3.3 - 3.5, the artifacts and the developed methods and algorithms are described in more detail. Section 3.6 describes the implementation of all three artifacts in the overall context and Sect. 3.7 discusses the evaluations.

3.3 Artifact 1: Data Preparation

The starting point to address RQ1 is the data provided by POCBR applications [128, 129]. This data can be used in the process of training a DL-based similarity measure and is therefore of importance. Common literature [128, 129] differentiates four main types of knowledge, i. e., the vocabulary (domain knowledge), similarity knowledge, knowledge about the case base, and adaptation knowledge. The domain where the application is used is usually strictly specified by a domain model (also known as vocabulary [128, 129]). The domain model can be very diverse, but in POCBR, it usually defines the structure and content of the process-oriented cases, e. g., its attributes, default values, node and edge types, etc. The similarity knowledge consists of all information about how similarities between cases are determined. This knowledge ranges from simple symbolic measures of equality or inequality to more complex measures such as weighted similarities based on the local-global principle [128]. The case base represents knowledge about experienced problem-solving situations. It is usually the main type of knowledge in CBR and POCBR applications [129]. The cases can be represented in many ways, though this thesis focuses on the representation as semantic graphs [15], which is the most common in POCBR. The adaptation knowledge is used in the adaptation process of cases for solving upcoming problems. However, it is disregarded in the current approach as the main focus is on case retrieval.

3.3.1 Data Encoding

Encoding the data provided by POCBR applications is necessary to properly train a DL model. Thereby, it is important to have a *generic encoding approach*, that can be used in arbitrary application contexts and use cases without the need to manually define encodings of the specific data of each domain. From the four types of data introduced before, the proposed encoding scheme [67] predominantly utilizes the information on the case base but also information about the domain and the similarity knowledge. More specific, the encoding scheme encodes the cases in the case base by incorporating the case information as well as the information from the domain and similarity knowledge.



0	0	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	1
0	0	0
0	0	0
1	1/3	0
0	1/2	1
0	1	1

(a) Tree Encoding

(b) Sequence Encoding

Figure 3.3: Encoding of Composite Types. Source: [67]

The *encoding scheme* transforms process-oriented cases, represented as semantic graphs, to a numeric vector representation by encoding the nodes and edges that form the process structure. These nodes and edges are characterized by a type and a semantic annotation. The node and edge types are encoded as one-hot encodings since its number is limited and this type of encoding is well-established for categorical data. Encoding the semantic annotations is more complex, as they can vary greatly in different domains. For instance, the experiments in [68, 122] use a recipe domain, a data mining domain, and a manufacturing domain with a distinct structure and content of the semantic annotations but the same underlying encoding scheme.

Encoding the semantic annotations follows the idea of viewing them as a collection of atomic data attributes, e. g., string, integer, and float, and composite data attributes that combine one or many composite or atomic attributes, e. g., list and set. This structure can be considered a tree, with inner nodes being the composite parts and leaf nodes being the atomic parts. Figure 3.3 depicts an exemplary semantic annotation that shows these parts. The encoding scheme consequently encodes the nodes of this tree by a vector with two components, i. e., a type vector that represents the data class of the type, e. g., string or integer, as a one-hot encoding and a value vector that encodes the concrete value of the type, e. g., “Spoon” or “2” (see [67] for more details). This tree encoding (see Fig. 3.3a) can also be transformed into a sequence encoding by lining up all atomic vectors, i. e., leaf nodes of the tree structure, in a sequence (see Fig. 3.3b). These two encoding variants enable the use of different types of neural network layers for processing and increase the configuration options w. r. t. the model architecture of RQ2. This means that the tree encoding can be processed with layers based on GNNs. Analogously, the sequence encoding can be processed with layers designated for sequences, such as RNNs or transformer-based layers (see [122] for more details).

3.3.2 Augmentation

A prerequisite for all applications involving DL methods is a solid set of training examples, as neural networks usually need large amounts of data to train effectively. In the proposed approach, the cases of the case base serve as training examples and large case bases can be difficult to acquire in POCBR applications. *Data augmentation* methods [40] specifically address this issue by artificially expanding training datasets. Various transformations of existing training examples as well as synthetically generated training examples enlarge the existing amount of training data with the goal of improving the generalization ability of the resulting models. This improved generalization is achieved, as the model's training procedure contains a wider range of variations in the training data, thereby reducing overfitting and enhancing model performance on unseen data.

This thesis proposes specific augmentation methods for semantic graphs to enlarge the available training sets [70]. Rather than only focusing on concrete augmentation methods, augmentation in general is discussed for data represented as semantic graphs to be used for similarity learning with GNNs. Therefore, several key criteria of the augmentation methods are discussed first. These criteria comprise the importance of semantic annotations, i. e., semantic annotations are an essential aspect of the similarity computation and respective augmentations can lead to significant changes in the composition of the training set. Additionally, the importance of the underlying control-flow and data-flow is highlighted, i. e., augmentations of the control-flow and data-flow change the nature of the cases significantly. Furthermore, another criterion, denoted as *correctness*, is described to define and select augmentation methods. The importance of the correctness of the augmented data is motivated by the fact that the augmented data should be as close to the original data as possible to ensure a high quality of the augmented data as DL training data. On the other hand, this criterion acts as a mechanism to allow augmentation methods to manage the tradeoff between useful augmentation methods and their complexity and required knowledge, as a higher quality also comes with a higher knowledge demand of the augmenting methods.

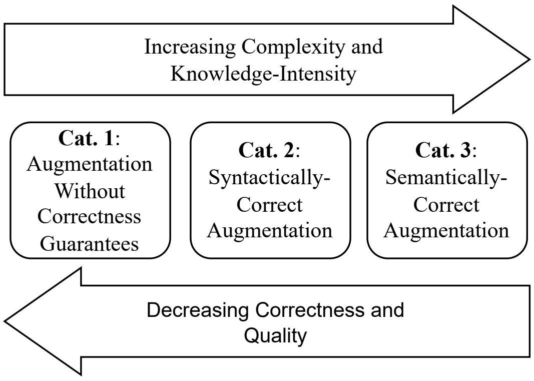

Increasing Complexity and Knowledge-Intensity
Cat. 1: Augmentation Without Correctness Guarantees
Cat. 2: Syntactically-Correct Augmentation
Cat. 3: Semantically-Correct Augmentation
Decreasing Correctness and Quality

Figure 3.4: Overview of the Categories of Augmentation Methods. Source: [70]

The correctness is measured by two factors, that is, the syntactic and the semantic correctness. *Syntactic correctness* addresses the structure of a semantic graph w. r. t. to its nodes and the edges between them. A violation of the syntactic correctness is equal to a violation of the constraints given by the definition of the semantic graph representation [15] such as, for instance, the removal of a part-of edge between a workflow node and a task node. While syntactically correct augmented graphs can ensure a valid structure, it is unclear whether their semantic annotations are also valid. To ensure this, *semantic correctness* is introduced as the conformity of an augmented process to the domain information given by the POCBR system. For instance, an augmented recipe graph (see the example in Fig. 2.2) is not semantically correct if the ingredient represented by a data node is not part of the taxonomy of ingredients of the underlying domain knowledge. Figure 3.4 depicts the relations between syntactic and semantic correctness and quality and knowledge intensity. The figure also distinguishes between three categories of augmentation methods. Augmentation methods of Cat. 1 give no correctness guarantees for the resulting augmented graphs. Methods of Cat. 2 are only syntactically but not semantically correct and methods of Cat. 3 are syntactically and semantically correct. The central premise of this distinction is that augmentation techniques that prioritize maintaining a higher level of correctness tend to be more complex and with a higher knowledge demand. Consequently, augmented data that exhibits a higher degree of correctness is typically a better reflection of the underlying domain and it is assumed to be more valuable for learning the characteristics of cases within that domain.

As part of the concept and to be used in the experiments [70], several concrete methods for each of the three categories are presented. The methods of Cat. 1 are mainly based on deletions, as they have rather low complexity, are easy to implement, and require no further knowledge apart from the case to augment itself. The augmentation methods based on deletion consider deleting nodes, edges, and (parts of) semantic annotations. For Cat. 2, methods based on replacements are proposed, either replacements within one case or across multiple cases. These methods ensure syntactic correctness because only syntactically correct cases are used as replacement candidates, what also makes the augmented graph syntactically correct. The concrete methods used in the experiments swap task nodes along the control-flow, data nodes along the data-flow, and parts of the semantic annotations. Concrete methods for Cat. 3 are more complex than those of the other two categories due to the need for semantic correctness, which requires an understanding of the underlying domain knowledge and constraints. The approach proposes to use AI planning [53, 60] for synthesizing new semantic graphs and transforming existing ones. Both variants build on a planning domain description [104, 111] that models all relations and constraints existing in the case data. Based on this description, an initial state and a goal state can be defined, and a planner [61] is used to create a semantic graph with this information.

3.4 Artifact 2: Model Architecture and Training

Artifact 2 aims to answer RQ2 and consists of *training procedures* and *model architectures* to use the augmented and encoded cases resulting from artifact 1 to learn a semantic graph similarity measure. The variants presented as part of this artifact are extensions of two GNN models from the literature, i. e., the GEM and the GMN introduced by Li *et al.* [97] (see the introduction in Sect. 2.2.3). The architecture of both models is extended as part of artifact 2 (see Sect. 3.4.1). The extended versions of the model are referred to as sGEM and sGMN. In addition, these extended models are used in individual training procedures to specifically address RQ2 (see Sect. 3.4.2).

3.4.1 Extended Architecture of the Graph Neural Networks

The architectural changes extending GEM to sGEM and GMN to sGMN involve specific methods for processing the semantic annotations and the integration of mapping constraints into the propagation layer.

Processing Semantic Annotations

One important architectural change of sGEM and sGMN compared to GEM and GMN is the utilization of specific embedders. While Li *et al.* [97] assume all nodes and edges of the processed graphs to be represented by simple feature vectors and to be processed by straightforward MLPs, the complexity of the node and edge information of semantic graphs (see Sect. 3.3.1) requires more elaborate implementations of the embedder. This especially refers to the sequence and tree encodings of the semantic annotations that need to be handled by specific neural network layers.

The thesis proposes embedder implementations based on a GNN for the tree encoding [67], an RNN [67], and an attention-based transformer layer (both for the sequence encoding) [122].

Semantic annotations that are encoded with the tree encoding method are treated as a graph and processed by a GNN that can be considered a small-scale version of the GNN architecture (see 2.2.3 for an overview) that also processes the entire semantic graph. The GNN consists of several trainable components: a component that embeds composite nodes, a component that embeds atomic nodes, a component that embeds edges, a component that performs message propagation, and a component that aggregates the node information to a single graph representation vector (i. e., the embedded representation of the entire semantic annotation). This variant of the embedder is highly configurable by hyperparameters such as the number of layers and neurons per layer for each trainable component or the number of propagation iterations.

While these configuration possibilities allow for fine-grain control over the neural network, they also add complexity. Processing semantic annotations that are encoded with the sequence encoding method by an unrolled RNN (e. g., [29, 64]) reduces the complexity while simultaneously also reducing the configuration possibilities. An RNN can handle sequences of inputs with different lengths, as they are present in the encodings of semantic annotations. We use the output state of the last step as the result of the unrolled RNN, which is a single embedding vector. One downside of RNNs is the sequential, non-parallel computation through the sequences, which can be a performance bottleneck, especially for large sequences.

This shortcoming is addressed by processing semantic annotations that are encoded with the sequence encoding method by attention-based transformers [154]. Transformers use multi-layered encoders and decoders that transform a sequence of inputs (i. e., atomic attributes of semantic annotations) to a sequence of embedded outputs. A key feature of the approach is its attention mechanism [9, 154] that allows to give more weight to certain elements of the sequence than others. The output sequence can be aggregated to a single representation of the entire semantic annotation by different aggregation mechanisms for neural networks, e. g., [98].

Mapping Constraints

The propagation layer of the GMN uses a cross-graph matching method between the nodes of the two input graphs (see [97] for more information). Each node of one graph is compared with each node of the other graph by their softmax-activated cosine vector similarity. The similarity measure for semantic graphs [15], however, only allows nodes and edges of the same type to be matched, thus *constraining the matching process*. These constraints are integrated into the cross-graph matching component of the sGMN as a form of IML [133], since prior knowledge is used within the ML setup (see previous work [65] for a discussion and an application scenario of IML in POCBR). With the constraints, a cosine similarity greater than zero is only allowed for nodes of the same type. When considering pairs of nodes with different types, a similarity score of 0 is assigned, indicating maximum dissimilarity. Consequently, there is minimal information propagation between nodes of distinct types.

3.4.2 Extended Training Procedures of the Graph Neural Networks

Besides architectural changes, sGEM and sGMN also encompass revised training procedures for processing semantic graphs properly. The proposed training procedures use different variants of TL, and ranking-based training is proposed.

Transfer Learning with Local and Global Similarities

An important aspect of similarity assessment between semantic graphs [15] is the local-global principle [128]. It is used in the way that the global similarity between two semantic graphs is composed of the local similarities between the nodes and edges of these graphs. The architectures of GEM and GMN do not incorporate this information and rely solely on labels of the global similarity. However, since the information about local similarities is available as a result of the labeling procedure, it can be considered by sGEM and sGMN.

Therefore, a TL approach is proposed that pretrains a *local similarity model* that is afterward used as part of the training phase of the *global similarity model* [122]. As Fig. 3.5 depicts, those phases are called the pretraining and the adaptation[3] phases, respectively.

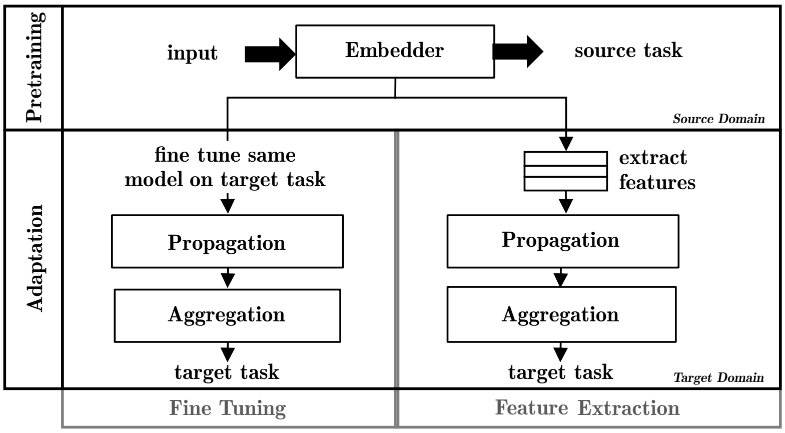


Figure 3.5: Pretraining and Adaptation Phases of Deep Transfer Similarity Learning. Source: [122]

During the *pretraining phase*, the embedder is trained to predict local similarities between nodes and edges. This process involves two steps: First, the embedder creates embedding vectors for nodes and edges based on their semantic descriptions and types. Next, it aggregates these embedding vectors to pairwise similarities. The authors propose to train different embedders for nodes and edges. This distinction is important because their semantic descriptions may differ in structure. The architecture of the embedder depends on how the semantic annotations are encoded (see Sect. 3.4.1). The embedder itself is an SNN, which is the de-facto standard in metric learning tasks (see Chicco [28] for an overview). To compute the local similarity between embedding vectors, an MLP is employed. The MLP is chosen for its expressiveness and computational efficiency. The training is a regression-like problem, where any standard regression loss function such as the *Mean Squared Error (MSE)* can be used.

In the *adaptation phase* of the TL approach, the pretrained embedder is integrated into sGEM and sGMN by replacing the standard, untrained embedder. This adaptation to the task of the target domain is primarily implemented in two ways in the literature [124], i. e., *feature extraction* or *fine-tuning*. Feature extraction is similar to classical feature-based approaches. The trainable parameters of the pretrained embedder are converted to constants (referred to as being "frozen") and the model will not be further trained during the training procedure of sGEM and sGMN. Alternatively, in a fine-tuning setup, the weights and biases of the pretrained embedder are used

[3] Please note that the term "adaptation" refers to its meaning in the context of TL in Sect. 3.4.2 and is not referring to the reuse phase in CBR.

as non-random initializations [33]. These parameters are further trained (fine-tuned) to adapt to the specific task.

Semi-Supervised Transfer Learning

All training procedures presented so far [67, 68, 122] are fully supervised. With the goal of training a pairwise semantic graph similarity measure, this means that the acquisition of labels (e. g., similarity values, case preferences [76], etc.) is necessary as a step before the training, either by a ground-truth similarity measure such as Bergmann and Gil [15] or manually by domain experts. However, this limits the applicability of the approaches in scenarios where, for instance, label acquisition is computationally expensive or infeasible for numerous cases.

To overcome these limitations, a semi-supervised TL method [86, 152] with an *unsupervised and a supervised training phase* is proposed. The first, unsupervised phase pretrains the GNNs with a *triplet learning* procedure [141] and, thus, does not require labeled data. The second, supervised training phase builds upon the pretrained model from the first phase and employs the original supervised training procedure [67] as a fine-tuning step. The training procedure is paired with the process augmentation approach (see Sect. 3.3.2) such that augmented, unlabeled process data is used in the unsupervised training phase and the original, labeled cases of the case base are used in the supervised training phase.

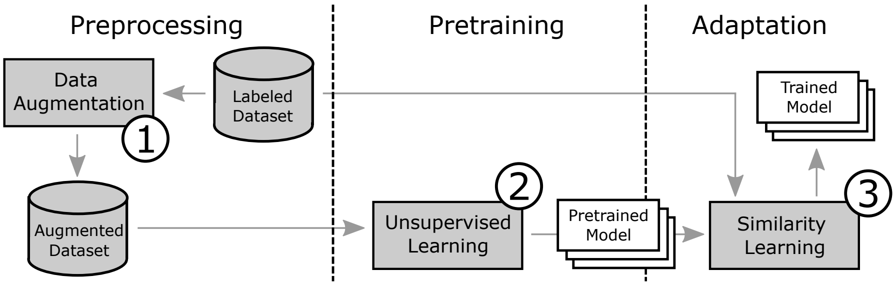


Figure 3.6: Architecture for Semi-Supervised Transfer Learning. Source: [142]

Figure 3.6 gives an overview of all components of the approach. As the supervised training phase is not significantly adjusted in this approach (see [70, 142] for the details), the unsupervised training phase is explained in more detail. The *unsupervised pretraining phase* uses an SNN with a triplet loss [141] for learning graph embeddings. The concrete model architecture can be reused from sGEM or sGMN. The idea of the training procedure is to generate similar embeddings for similar graphs and dissimilar embeddings for dissimilar graphs (in terms of vector space similarity) without the use of label similarities to steer the embedding process. The triplets, i. e., the training examples, consist of an *anchor*, a *negative*, and a *positive* and the goal is to maximize the similarity for the pair of the anchor and the positive and to minimize the similarity for the pair of the anchor and the negative. The proposed way to put together these triplets is by selecting a case from the case base (anchor), augmenting this case with one of the methods described in Sect. 3.3.2 (positive), and selecting another case randomly from the case base (negative). Batches of triplets built this way are then used to train either the sGEM or the sGMN (see [70, 142] for the details).

Ranking-Based Retrieval

Case retrieval in CBR and POCBR is commonly concerned with finding the most similar cases w. r. t. a query. Using sGEM and sGMN as a similarity measure in this regard is possible and matches their initial purpose [97]. While related work (see Sect. 2.3 and 2.4.1 for examples) also pursues this idea, there are many applications where solely the ranking of the retrieval results is important and not the specific similarities that determine this ranking. Predicting *pairwise preferences* such as "*case X is more relevant to the query than case Y*" appears more intuitive in such situations. The integration of LTR [100] principles (also known as preference learning [47]) into sGEM and sGMN follows this goal [68].

A pairwise preference is computed between two pairs of graphs, i. e., the query graph and one case graph per pair. The first change compared to GEM and GMN is the implementation of the final layer where the embedded graphs are not transformed into a similarity value but a preference value in [0,1], indicating a probability which of the cases is more relevant to the query. Training sGEM and sGMN in this setup also differs from the original goal of predicting similarities. Instead of computing a regression loss such as the MSE between predicted and label similarity, the predicted preferences are evaluated by computing a binary cross-entropy loss [3, p. 118]. This loss is suitable for classification tasks, which matches the setup of predicting pairwise preferences. Note that this pairwise ranking approach can be trained directly using label preferences between cases, which can be determined through direct user feedback. Unlike similarity-based models, this approach allows for capturing real-world preferences more naturally. Human experts find it easier to express a preference between two alternatives rather than assigning a similarity value [76].

To transform the predicted pairwise preferences to an actual ranking which can, in turn, be used as the retrieval result, it is necessary to perform a procedure known as *rank aggregation* [4, 75]. Two different methods of rank aggregation are proposed [68], where one is a modified version of Cohen, Schapire, and Singer [32] and the other one is a simpler and faster alternative (see [68] for the details).

3.5 Artifact 3: Hyperparameter Optimization and Configuration

The methods developed in artifact 1 (see Sect. 3.3) and artifact 2 (see Sect. 3.4) reveal numerous configuration and parameterization possibilities when using sGEM and sGMN in a POCBR application context. Some of these parameters depend on each other, such as the GNN-based embedder (see Sect. 3.4.1) requires tree-encoded semantic annotations (see Sect. 3.3.1), but most of the parameterization can be done mix-and-match. This reveals the challenge of *setting these hyperparameters appropriately* to allow for an optimal configuration of sGEM and sGMN in the corresponding application context.

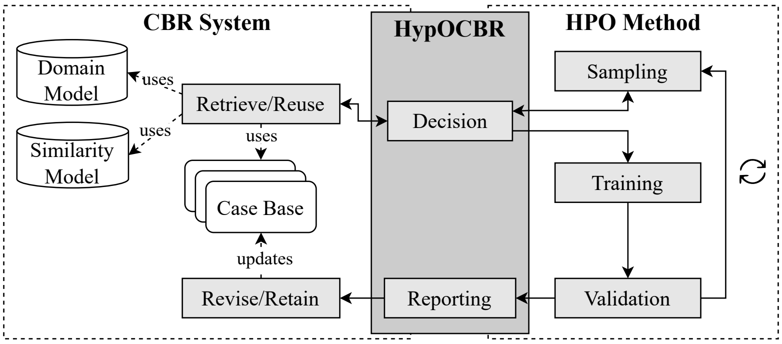


Figure 3.7: Architecture of the CBR system, the HPO method, and HypOCBR. Source: [66]

The third artifact has been designed to address this challenge with an approach called

HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) [66]. HypOCBR can extend state-of-the-art HPO methods [31, 44, 101] with a CBR system that enables the integration of expert knowledge into the optimization process. The approach is based on the observation that ML engineers usually have a lot of implicit expert knowledge, e. g., from development tests or experiments in literature, but lack an option to include this experience in the configuration of common HPO methods that are designed to be fully automatic. Thus, HypOCBR bridges the gap between manual, experience-based and automatic, black box HPO. It also serves as an example of how CBR can improve typical DL or ML tasks such as HPO, which is considered to be a research topic with a lot of potential [88].

Figure 3.7 shows an architectural overview of the approach, consisting of the CBR system, the HPO method, and HypOCBR as the connector. The HPO method performs an optimization procedure by sampling hyperparameter vectors from the search space, conducting training, and validating the model's performance (see [44] for more information). The approach allows for the use of any arbitrary HPO method, as their functionality does not need to be tailored specifically to HypOCBR. In this context, the CBR system supports the optimization procedure by maintaining a case base containing completed trials of hyperparameter vectors and their corresponding loss values. The CBR system also includes domain and similarity models, thus, incorporating domain expert knowledge. HypOCBR bridges the gap between the CBR system and the HPO method using a lightweight implementation of the 4R cycle (as proposed by Aamodt and Plaza [1]): During optimization, when a new hyperparameter vector is sampled, HypOCBR decides whether to proceed with this vector or abort it early and sample another one. Each sampled vector serves as a query to the CBR system, which searches the case base for similar hyperparameter vectors and their corresponding loss values, i. e., retrieving trials. HypOCBR then reuses the most similar trials and makes a decision based on their loss values. Over time, each optimization gradually builds up a case base, starting with an empty one and retaining each completed trial. Note that this approach does not explicitly revise cases but, instead, stores all reported validation results from the HPO component in the case base.

At the core of HypOCBR is an *algorithm that decides* between proceeding with a sampled hyperparameter vector or aborting the trial (see a description with more details in [66]). First, the algorithm tries to fill a case base of hyperparameter vectors and their validation results to avoid the knowledge gap, known as the cold start problem [136], after the initial start of the system. The following steps are performed sequentially and only when reaching the final step of the check, a hyperparameter vector might be aborted: 1) A retrieval with the built case base and a query in the form of a hyperparameter vector is performed, resulting in the k-most similar hyperparameter vectors to the query vector. 2) The similarities of the retrieved cases are aggregated and the search proceeds with the query vector if the aggregated value does not exceed a certain threshold. 3) The validation results, i. e., loss values, of the retrieved cases are aggregated and compared to the distribution of the other cases in the case base by computing a percentile value. If this percentile value denotes the retrieved cases among the worst of the case base, *ABORT* is returned, as the query vector is likely to also perform poorly. This algorithm balances the well-known principles of *exploration* and *exploitation* (e. g., [89]). It can be parameterized by meta-parameters towards exploration, i. e., hyperparameter vectors that are different from all previously encountered vectors or exploitation, i. e., hyperparameters that are similar to known good hyperparameter vectors.

3.6 Software Implementation of the Artifacts

Each of the three artifacts also consists of an implementation. Since the author of this thesis values the principles of open-source software and the accompanied possibilities of public use, the implementations are *openly available* free of charge. The POCBR part of the implementations is implemented in Java as an extension[4] of the POCBR framework *ProCAKE* [16] (see Sect. 2.1.4). The compatibility with ProCAKE ensures flexible usage of the approach for all kinds of POCBR applications. Additionally, the DL part is concerned with all code related to the used DL models and is implemented in Python with *TensorFlow* [2] as the main library[5]. Both parts of the implementation work in combination and fulfill certain tasks to achieve a flexible integration of embedding-based similarity measures into POCBR applications. Figure 3.8 depicts the implementation in a form that is similar to the ML lifecycle [8]. The parts implemented by ProCAKE are shown with a white background and the parts implemented by TensorFlow with a gray background. As ProCAKE holds all the data required to train the DL models, data preparation, i. e., augmentation and encoding, is done at the side of ProCAKE (see Artifact 1, Sect. 3.3). The data is then exported and transferred to the DL part, where the training procedure is initiated. It involves all corresponding aspects that are outlined in the artifacts 2 and 3 (see Sect. 3.4 and Sect. 3.5). Thus, the model architecture is defined, the training procedure is determined, and all components are optimized. The result is the trained model that is served and available for inference. The served model is then used by the POCBR part during similarity assessment and case retrieval. The raw numeric predictions of the model are turned into retrieval results by ProCAKE. This process can be repeated periodically or according to certain triggers, such as changes in the case base.

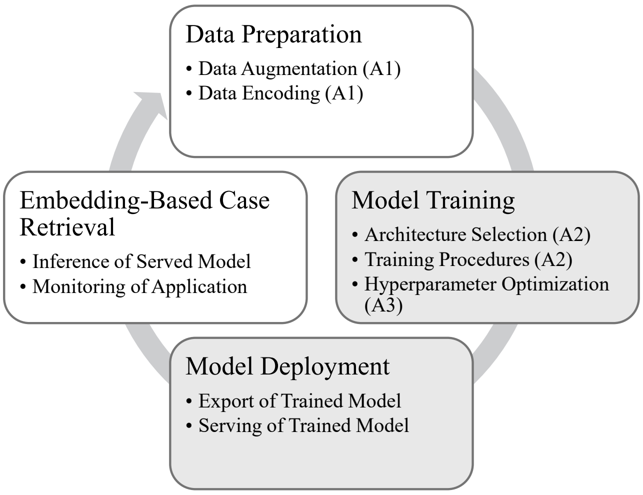

Data Preparation
• Data Augmentation (A1)
• Data Encoding (A1)
Model Training
• Architecture Selection (A2)
• Training Procedures (A2)
• Hyperparameter Optimization (A3)
Model Deployment
• Export of Trained Model
• Serving of Trained Model
Embedding-Based Case Retrieval
• Inference of Served Model
• Monitoring of Application

Figure 3.8: Implementation of the Artifacts.

3.7 Experimental Evaluations

To validate the developed methods and algorithms of each artifact, evaluations have been conducted. This section introduces the general setup of these evaluations, including goals and hypotheses, metrics, and datasets (see Sect. 3.7.1), and gives an overview of the specific evaluation for each artifact (see Sect. 3.7.2).

[4] https://gitlab.rlp.net/procake-embedding/procake-embedding-core
https://gitlab.rlp.net/procake-embedding/procake-embedding-protobuf
https://gitlab.rlp.net/procake-embedding/procake-embedding-experiments

[5] https://gitlab.rlp.net/procake-embedding/procake-embedding-python

3.7.1 General Evaluation Setup

The general idea behind the evaluations of all three artifacts is to compare similarity measures using sGEM and sGMN with other existing similarity measures, such as the A* measure (see Sect. 2.1.3 and [15, 173] for an introduction). It is generally claimed that similarity measures using sGEM and sGMN outperform the similarity measures compared against. As one part of the comparison, it focuses on the quality of the computed similarities that is measured in terms of prediction errors, ranking errors, and retrieval quality. The experiments capture the *prediction errors* with metrics such as the *Mean Absolute Error (MAE).* These metrics express the average difference between two similarity predictions, either two variants of an embedding-based measure or the difference from a ground-truth measure. Correlated to the prediction errors are the metrics measuring the *ranking errors.* One example of such a metric, the correctness [27], describes the conformity of the ranking positions of the case pairs in the predicted retrieval results according to the ground-truth retrieval results. While the raw prediction errors and the ranking measures do not express the performance of a measure in a retrieval situation, the *quality of the retrieval results* is also examined. One measure of this category is described by Müller and Bergmann [116]. It quantifies the degree to which highly similar cases according to the ground-truth retrieval results are present in the predicted retrieval results. If, for example, the predicted results do not accurately contain a highly similar case in the top-k ranks, the quality will decrease. The second part of the comparison is the performance that is also examined in the evaluations. The performance is measured as the computation time of a certain task, e. g., the time different measures take to compute the similarities for one retrieval run. This is an important metric as it allows setting the measures into context. Specifically, the trade-off between expressiveness, e. g., low prediction error, and the time needed to predict is of interest.

The evaluations are conducted with multiple datasets to investigate how the models behave, given different amounts of data or data from different domains. Most of the publications, thereby, investigated three domains, i. e., a cooking domain [71], a data mining domain [174], and a manufacturing domain [103]. The cooking domain consists of manually modeled cooking recipes with ingredients and cooking steps (see Müller [115] for more details). The workflows of the data mining domain are built from sample processes that are delivered with the data analytics platform RapidMiner (see Zeyen, Malburg, and Bergmann [174] for more details). Finally, the manufacturing workflows stem from a smart manufacturing *Internet of Things (IoT)* environment and represent sample production processes (see Malburg, Hoffmann, and Bergmann [103] for more details). The three domains pose different challenges for the learning process of the models. For instance, the data mining workflows typically have many nodes and edges and are composed of complex semantic annotations, while the recipes have fewer nodes and edges with rather simple semantic annotations.

3.7.2 Specific Evaluation of Each Artifact

The concrete evaluations of the three artifacts follow the common goal and setup described before but are more specifically targeting the methods of the individual artifacts. Please see the corresponding publications for additional information.

Evaluation of Artifact 1

The evaluation of artifact 1 examines the influence of different data preparation and augmentation methods on the model quality and performance. In particular, the evaluation compares different encoding schemes, i. e., tree and sequence encoding (see Sect. 3.3.1), and different augmentation methods with different levels of correctness (see Sect. 3.3.2). While no specific hypotheses were formulated for the comparison of the encoding schemes, the hypotheses for the comparison of the augmentation methods claimed that model quality and performance increases with more data and data of higher quality. This means that, first, an increase is expected when using datasets with additional augmented data over the respective original datasets. Second, an increase is also expected when using datasets augmented with methods with a higher level of correctness compared to augmented datasets with a lower level of correctness.

The results are discussed separately regarding the encoding schemes and the augmentation methods. The results of using different encoding schemes are mixed and their influence depends on the domain and whether sGEM or sGMN is used. However, both schemes enable the usage of POCBR data in neural networks and, thus, contribute to the general goal of this thesis. The comparison of the augmentation methods only partly confirmed the hypotheses. First, the results indicate that higher levels of correctness do not consistently lead to better results. The reason for this could be the capability of the neural networks to filter the components of the training examples such that a certain share of incorrect parts is negligible. Second, the results indicate that there is a limit for the amount of training data that, when reached, gradually worsens the quality of the trained model. The reason for this is probably a misalignment of the number of trainable neural network parameters and the number of training examples.

Evaluation of Artifact 2

The evaluation of artifact 2 (see Sect. 3.4) compares different neural network architectures and training procedures. This evaluation is twofold: on the one hand, the experiments compare the DL methods among themselves, analyzing how different architectures and training approaches perform. On the other hand, these DL models are compared to existing non-DL methods such as a feature-based similarity measure [17] and a measure based on A* search [15] for the tasks of similarity assessment and case retrieval. By comparing both types of methods, the evaluation provides insights into the relative advantages and limitations of deep learning for these specific tasks.

For the first aspect of the evaluation, the main hypothesis states that DL methods outperform existing non-DL techniques in terms of accuracy and speed. The results consistently indicate that DL approaches offer notable enhancements in both quality and performance. Specifically, DL models tend to be faster when delivering comparable quality, or significantly more accurate when operating under a similar time budget. The second part of the evaluation is conducted across multiple publications (see Tab. 3.1), where each subsequent publication builds on the previous one by comparing the latest methods with the new ones. The results highlight the successive improvements achieved through new methods. For example, the introduction of TL (see Sect. 3.4.2) improves both the quality of results and the speed of processing compared to models that do not leverage pre-trained networks. Additionally, the evaluation indicates that no single method performs best across all scenarios. Different architectures and training procedures yield better results depending on the specific use case, domain, or objective. This suggests that selecting the appropriate approach for each context is essential for maximizing efficiency and effectiveness.

Evaluation of Artifact 3

The evaluation of artifact 3 (see Sect. 3.5) examines the influence of HypOCBR on model quality and overall performance. This is done by conducting a series of HPO runs, comparing results from two different approaches: one set of runs utilizes HypOCBR, while the other does not. The key hypothesis for this evaluation is that the inclusion of HypOCBR leads to better model quality, with an expectation that its impact is more significant in complex setups, such as those involving longer HPO runs and a larger number of hyperparameters to optimize.

The experimental results largely confirm these hypotheses. Across almost all runs, models tuned with HypOCBR achieve higher quality than those without it, demonstrating that HypOCBR consistently helps to find more optimal hyperparameter configurations. Furthermore, in more complex optimization setups, where the search space is larger or the optimization run is more time-consuming, HypOCBR shows an improvement in performance. In summary, the hypotheses are mostly confirmed, with HypOCBR indicating to be a valuable tool for enhancing both model quality and performance, especially in scenarios requiring longer, more complex optimization runs.

Chapter 4

Conclusion and Future Work

This thesis presents a hybrid approach of DL and POCBR methods for similarity assessment and retrieval of procedural cases. The approach investigates three research questions and yields three artifacts (see Sect. 3.2). The artifacts address data preparation (Artifact 1), DL model architecture and training (Artifact 2), and hyperparameter optimization and -configuration. (Artifact 3). The contributions to these artifacts are published in five accompanied papers (see Ch. 5 – 9). The remainder of this chapter discusses the proposed approach w. r. t. the research questions (see Sect. 4.1), describes its limitations (see Sect. 4.2), and gives directions for future research (see Sect. 4.3).

4.1 Discussion

The thesis aims to answer the three research questions RQ1 to RQ3 with the contributions of the five associated publications.

RQ1 is addressed by a *generic encoding scheme* that transforms the contents of the cases in the case base into a numeric vector representation. This transformation also considers information apart from the case base, such as value ranges of attributes. The encoding scheme is designed to be generic and can be used for arbitrary domains and, thus, a wide variety of POCBR application contexts. The evaluations (see Sect. 3.7) are also performed with data encoded this way and demonstrate a wide applicability in different workflow domains.

The contributions to RQ1 enable training DL models with procedural cases in the first place, which is a requirement to tackle RQ2. RQ2 is concerned with investigating how this data can be used to train DL models for similarity assessment and case retrieval in POCBR applications. This thesis proposes *several architectures and accompanying training procedures*. They are all based on the GEM and the GMN [97] and extend their functionality (then denoted as sGEM and sGMN) according to specific characteristics and requirements of POCBR use cases. These architectures and training procedures can mostly be considered alternatives that can be swapped depending on the concrete domain and use case. However, there is a general trade-off between the two base models that should be considered. The sGMN has a higher expressiveness and can approximate similarities more effectively, but, in turn, it has an increased inference time. This is mainly due to the computational complexity of the cross-graph matching component [97]. The sGEM, on the other hand, is computationally more efficient but lacks expressiveness. These two factors should be weighed up individually for each application. Combinations of both models in one application can also be useful and are discussed in Hoffmann and Bergmann [67] as a MAC/FAC retrieval [46, 82] approach. Dealing with the trade-off between sGEM and sGMN, selecting different variants of these models, and choosing the right hyperparameters for the setup is also addressed as part of RQ2, with the HPO approach *HypOCBR*. HypOCBR consolidates the contributions to model architecture and training procedures by providing a way to optimize them. The approach aims at using experience knowledge about known-good hyperparameter vectors and their properties to guide the optimization procedure. For instance, sGEM and sGMN could be configured this way by increasing or decreasing their expressiveness, e. g., by increasing or decreasing the number of neurons in the layers.

The contributions to RQ3 provide a solution for data scarcity in POCBR (and sometimes also CBR) applications. *Process augmentation* can be used to create new training examples from existing cases in the case base. Thereby, multiple levels of correctness are considered for the augmented cases that can be used according to the requirements of different use cases. However, augmentation alone does not solve the problem of having too little labeled training data, as the augmented data is generally unlabeled. This is addressed by the proposed *semi-supervised training procedure*, where unlabeled data is used in an unsupervised pretraining phase and labeled training data, e. g., the original case base, is used to fine tune the pretrained model in a supervised training procedure.

All these contributions are part of the effort to create a hybrid, neuro-symbolic AI approach for POCBR. As initially outlined in the introduction, the hybrid approach aims to combine the strengths of both methods to avoid the respective weaknesses. The main targeted weakness of CBR and POCBR approaches is the high knowledge-acquisition effort. It is addressed by automatically learned DL models for similarity assessment and retrieval. Especially in the unsupervised variants (see Sect. 3.4.2), these models reduce the knowledge-acquisition effort and contribute to the initial goal.

4.2 Limitations

When looking at the conducted evaluations for each artifact and the corresponding publications (see Sect. 3.7), it becomes apparent that reductions in knowledge-acquisition effort are not directly measured or evaluated as part of this thesis. On the one hand, it is difficult to measure this objectively and, on the other hand, it is evident from the proposed automatic learning framework that it can provide a trained model with no additional knowledge acquisition. The situation is different for the other main goal, i. e., the quality and performance benefits, as most of the conducted experiments aim towards measuring quality and performance (see Sect. 3.7). The results show general potential of the embedding-based models regarding the quality of the predictions, as well as the time needed to predict. When looking beyond performance and knowledge-acquisition benefits, there are still several advantages, disadvantages, and limitations of hybrid approaches using DL models compared to other existing similarity measures to consider. The following discussion uses the A*-based similarity measure by Bergmann and Gil [15] as a placeholder for all existing POCBR similarity measures:

1. The inference runtime of the embedding-based approach remains consistent across different runs, with only marginal variation based on the size of the input graphs. In contrast, the runtime of the A*-based graph matching procedure can exhibit significant variability from run to run due to its integrated search mechanism. On the downside, this also means that the runtime budget of the embedding-based approach is only configurable by a few designated mechanisms based on the requirements of different use cases, e. g., general purpose strategies such as integer quantization [164]. The graph matching approach allows configuration with a larger impact by restricting the number of solution candidates considered during the search process [15].

2. While the embedding-based approach can benefit greatly from hardware accelerators for AI workloads [25] and their expected popularity in the future, the graph matching approach is mainly computed by the CPU, and, thus, harder to scale. There are also acceleration possibilities for A* search, for instance on GPUs (e. g., [69, 177]), but they require custom implementations and need to be adapted to the specific domain.

3. With the embedding-based approach of the sGEM, it is possible to generate and cache embedding vectors. These vectors can then be utilized in their vector form for similarity calculations. By caching all embedding vectors from a process model repository during an offline phase, time can be saved during the online phase of an application, e. g., during retrieval. Caching at this level is not feasible for the graph-matching-based approach.

4. The embedding-based approach lacks explainability compared to the A*-based approach. While the A*-based approach can explain its outcomes with the composition of local similarities that lead to the global similarity, the embedding-based approach only predicts a global similarity. Li *et al.* [97] introduce a way to explain the model's predictions with the attention weights of GMN and sGMN, respectively, but it is not as powerful as the native explainability provided by the A* method.

   Whether explainability is of importance for the underlying application is strongly dependent on the application context. For example, a human subjects study conducted by Gates, Leake, and Wilkerson [52] found the presentation of the cases as the most effective method for explaining retrieval results. Although the study featured a rather simple domain, it might indicate that explaining the concrete similarity values is not always necessary, which favors the use of the embedding-based approach.

This highlights that the choice between an embedding-based measure or a manually modeled one is complex and strongly dependent on the use case. Nevertheless, a thorough comparison of all different variants of sGEM and sGMN and different POCBR similarity measures would be helpful and a good starting point for future work.

4.3 Future Work

Consolidating and extending the evaluations of sGEM and sGMN in different scenarios, domains, and circumstances is straightforward for future work. Besides that, the approach proposed in this thesis could be extended and pursued further in the following ways:

Dynamic Data Encoding and Embedding

The encoding scheme proposed as part of this thesis (see Sect. 3.3.1) is very generic by design to allow its usage in arbitrary domains. This can result in an overhead regarding the size of the encodings as well as the processing time. One opportunity to reduce this overhead in future work could be a more dynamic and domain-specific encoding of the data. One idea for an improved encoding scheme could be an automatic method that analyzes the domain at hand and creates an encoding template specifically for this domain. Based on this template, the initial layers of the DL-model, i. e., the embedder, can be configured accordingly. The overhead is likely reduced as neither the encoding scheme nor the embedder needs to be set up for generic domain data. In a subsequent evaluation, it is important to examine if the dynamic encoding scheme maintains the quality of the generic scheme.

Generic Graph Neural Networks

The model architecture of sGEM and sGMN is derived from the architecture of GEM and GMN [97]. Many other GNN approaches follow the same general architecture (see, e. g., [168, 176, 178]) but are currently not usable with the extensions proposed as part of this thesis, as their implementation and possibly also aspects of the conceptualization are specific to GEM and GMN. However, the proposed approach would benefit from a more generic conceptualization and implementation to make the application of other baseline GNN architectures possible and to be able to apply the proposed ideas to new state-of-the-art GNNs. This future research direction should consequently be followed.

Case Reuse

The approach proposed in this thesis aims at similarity assessment and the retrieve phase of POCBR applications [1]. Other tasks and phases are not considered but, nevertheless, important in POCBR and CBR applications. Especially the reuse phase and the task of case adaptation appears to be a promising candidate for using variants of the proposed DL models, as previous work found [19]. This work builds on the concept of GNN-based embedding but describes a different training procedure. In the training phase of the neural network, RL is used to iteratively predict operations, e. g., delete, replace, and insert, that adapt the retrieved case to better suit the query. The work only describes a concept and demonstrates it with a case study, but it lacks an experimental evaluation. Further research in this direction could increase the applicability of the proposed approach through all phases of POCBR applications. Furthermore, as it is a combination of retrieval and case adaptation, adaptation-guided retrieval [146] is also a promising use case for the presented DL-based models and should be investigated in future work.

eXplainable Artificial Intelligence

Using the proposed DL-based approach for retrieving cases lacks transparency compared to using a conventional retrieval approach, e. g., an A*-based approach [15, 120]. For instance, the A*-based measure can explain its global similarity as the result of the amalgamation of the computed local similarities. The DL-based retrieval only yields pairwise similarities between processes without further information on the origin or composition of the final similarity predictions. Improving the explainability of the proposed approach should be pursued in future work. There is an entire research field called XAI [37] that investigates in this direction. XAI is also an active research area among CBR researchers. Some approaches have the goal of using CBR to explain black-box models such as neural networks (e. g., [80]) while others try to make the explanations of similarities in CBR applications more accessible [143]. The approach proposed in this thesis could benefit from the mentioned approaches or other ones from this research field and integrate explanations into the prediction process.

Uncertainty in Similarity Learning

The predictions of the DL models used in this thesis are not always correct, hence their evaluation by the prediction error such as the MAE. As the models only predict a pairwise similarity or ranking preference, respectively, there is no information about the certainty that this prediction comes with. Explicitly expressing this certainty or uncertainty could be a research topic for future work. Based on DL literature on this topic (e. g., [73, 81]), the proposed DL-based retrieval approach can benefit from specific uncertainty estimations for the predicted retrieval results to know how trustworthy they are. The aspect of uncertainty (also known under the term “fuzzy”) has been studied for CBR systems (e. g., [74, 169]). A combination of the findings of DL and CBR literature about uncertainty appears to be a promising direction for future work.

Ensembling for Robust Predictions

Ensembling [35] is a common way to enhance the performance of an ML model by aggregating the predictions of several models into a single one (called an ensemble). Future work should investigate the potential of using ensembling for the proposed DL-based retrieval approach. For instance, several sGEM models that are parameterized for a different behavior (e. g., fast inference, accurate results, consistent runtime etc.) could be trained on different subsets of the case base and then combined into one ensemble. Ensembling is popular across many ML applications, and the approach from this thesis could possibly benefit from increased robustness and prediction quality.

Lifelong Learning and Case Base Maintenance

The presented approach for DL-based retrieval assumes that the case base of the POCBR application is relatively static, as these cases are used as training examples for the DL model. This assumption, however, is invalid for some application scenarios, e. g., user ticketing systems with many requests [6]. Therefore, future work should investigate how frequent changes in the underlying data and, especially, dynamic case bases can be incorporated into the presented approach. These considerations for ML models are commonly referred to as lifelong or continuous ML [26]. Similar ideas are discussed for the maintenance of CBR systems [131, 134] and a combination of ideas from both research fields seems promising.

Large Language Models for Similarity Learning

The most noticeable progress in AI research can currently be observed in the field of generative LLMs, such as the GPT models [20]. Due to the vast amounts of data that these models are trained on, they are also useful outside classical NLP tasks. This indicates an interesting direction for future work, i. e., to conceptualize similarity learning between semantic graphs with LLMs and to compare the results with the approach of this thesis. A possible concept could be a few-shot learning approach of three steps: First, the model is prompted with the context of the task, including domain information and similarity information. Second, the model is iteratively prompted with pairs of semantic graphs and the expected pairwise similarity. The third step addresses the actual task and prompts the model with unseen semantic graphs, asking for similarity predictions. Other promising methods in the field of LLMs could be chain-of-thought prompting [161] to split the prediction problem into smaller subproblems, similar to the local-global principle [128], and retrieval-augmented generation [96] to make a large amount of structured knowledge available to the model. Since POCBR applications are usually rather complex, it might be beneficial to examine simpler types of data in CBR applications beforehand, or to build on existing approaches outside the scope of CBR (e. g., [106]). BPM researchers also examine LLMs [12, 22, 79, 156], which is interesting due to the similarities between POCBR and BPM.

Bibliography

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994). https://doi.org/10.3233/AIC-1994-7104

2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P.A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning. In: Keeton, K., Roscoe, T. (eds.) 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016, pp. 265–283. USENIX Association (2016)

3. Aggarwal, C.C.: Neural Networks and Deep Learning - A Textbook. Springer (2018)

4. Allwein, E.L., Schapire, R.E., Singer, Y.: Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. J. Mach. Learn. Res. **1**, 113–141 (2000)

5. Amin, K., Kapetanakis, S., Althoff, K., Dengel, A., Petridis, M.: Answering with Cases: A CBR Approach to Deep Learning. In: Cox, M.T., Funk, P., Begum, S. (eds.) Case-Based Reasoning Research and Development - 26th International Conference, ICCBR 2018, Stockholm, Sweden, July 9-12, 2018, Proceedings. LNCS, vol. 11156, pp. 15–27. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01081-2_2

6. Amin, K., Kapetanakis, S., Polatidis, N., Althoff, K., Dengel, A.: DeepKAF: A Heterogeneous CBR & Deep Learning Approach for NLP Prototyping. In: Ivanovic, M., Yildirim, T., Trajcevski, G., Badica, C., Bellatreche, L., Kotenko, I.V., Badica, A., Erkmen, B., Savic, M. (eds.) International Conference on INnovations in Intelligent SysTems and Applications, INISTA 2020, Novi Sad, Serbia, August 24-26, 2020, pp. 1–7. IEEE (2020). https://doi.org/10.1109/INISTA49547.2020.9194679

7. Amin, K., Lancaster, G., Kapetanakis, S., Althoff, K., Dengel, A., Petridis, M.: Advanced Similarity Measures Using Word Embeddings and Siamese Networks in CBR. In: Bi, Y., Bhatia, R., Kapoor, S. (eds.) Intelligent Systems and Applications - Proceedings of the 2019 Intelligent Systems Conference, IntelliSys 2019, London, UK, September 5-6, 2019, Volume 2. Advances in Intelligent Systems and Computing, pp. 449–462. Springer (2019). https://doi.org/10.1007/978-3-030-29513-4_32

8. Ashmore, R., Calinescu, R., Paterson, C.: Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. ACM Comput. Surv. **54**(5), 111:1–111:39 (2022). https://doi.org/10.1145/3453444

9. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

10. Barnett, A.J., Schwartz, F.R., Tao, C., Chen, C., Ren, Y., Lo, J.Y., Rudin, C.: A case-based interpretable deep learning model for classification of mass lesions in digital mammography. Nat. Mach. Intell. **3**(12), 1061–1070 (2021). https://doi.org/10.1038/S42256-021-00423-X

11. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V.F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H.F., Ballard, A.J., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K.R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M.M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. CoRR **abs/1806.01261** (2018). arXiv: 1806.01261

12. Beheshti, A., Yang, J., Sheng, Q.Z., Benatallah, B., Casati, F., Dustdar, S., Motahari-Nezhad, H.R., Zhang, X., Xue, S.: ProcessGPT: Transforming Business Process Management with Generative Artificial Intelligence. In: Ardagna, C.A., Benatallah, B., Bian, H., Chang, C.K., Chang, R.N., Fan, J., Fox, G.C., Jin, Z., Liu, X., Ludwig, H., Sheng, M., Yang, J. (eds.) IEEE International Conference on Web Services, ICWS 2023, Chicago, IL, USA, July 2-8, 2023, pp. 731–739. IEEE (2023). https://doi.org/10.1109/ICWS60048.2023.00099

13. Bengio, Y.: From System 1 Deep Learning to System 2 Deep Learning: Posner Lecture, Neurips, (2019). Available at: https://slideslive.com/38922304. Last accessed: 2024-07-05.

14. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Springer (2002)

15. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014). https://doi.org/10.1016/J.IS.2012.07.005

16. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Kapetanakis, S., Borck, H. (eds.) Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Reasoning co-located with the Twenty-seventh International Conference on Case-Based Reasoning (ICCBR 2019), Otzenhausen, Germany, September 8-12, 2019. CEUR Workshop Proceedings, pp. 156–161. CEUR-WS.org (2019)

17. Bergmann, R., Stromer, A.: MAC/FAC Retrieval of Semantic Workflows. In: Boonthum-Denecke, C., Youngblood, G.M. (eds.) Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida, USA, May 22-24, 2013. AAAI Press (2013)

18. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res. **13**, 281–305 (2012). https://doi.org/10.5555/2503308.2188395

19. Brand, F., Lott, K., Malburg, L., Hoffmann, M., Bergmann, R.: Using Deep Reinforcement Learning for the Adaptation of Semantic Workflows. In: Malburg, L., Verma, D. (eds.) Proceedings of the Workshops at the 31st International Conference on Case-Based Reasoning (ICCBR-WS 2023) co-located with the 31st International Conference on Case-Based Reasoning (ICCBR 2023), Aberdeen, Scotland, UK, July 17, 2023. CEUR Workshop Proceedings, pp. 55–70. CEUR-WS.org (2023)

20. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language Models are Few-Shot Learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Pro-

cessing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)

21. Bunke, H.: Recent Developments in Graph Matching. In: 15th International Conference on Pattern Recognition, ICPR'00, Barcelona, Spain, September 3-8, 2000, pp. 2117–2124. IEEE Computer Society (2000). https://doi.org/10.1109/ICPR.2000.906030
22. Busch, K., Rochlitzer, A., Sola, D., Leopold, H.: Just Tell Me: Prompt Engineering in Business Process Management. In: van der Aa, H., Bork, D., Proper, H.A., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling - 24th International Conference, BPMDS 2023, and 28th International Conference, EMMSAD 2023, Zaragoza, Spain, June 12-13, 2023, Proceedings. LNBIP, vol. 479, pp. 3–11. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-34241-7_1
23. Çayoglu, U., Dijkman, R.M., Dumas, M., Fettke, P., García-Bañuelos, L., Hake, P., Klinkmüller, C., Leopold, H., Ludwig, A., Loos, P., Mendling, J., Oberweis, A., Schoknecht, A., Sheetrit, E., Thaler, T., Ullrich, M., Weber, I., Weidlich, M.: Report: The Process Model Matching Contest 2013. In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers. LNBIP, vol. 171, pp. 442–463. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-319-06257-0_35
24. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. CoRR **abs/1512.01274** (2015). arXiv: 1512.01274
25. Chen, Y., Xie, Y., Song, L., Chen, F., Tang, T.: A Survey of Accelerator Architectures for Deep Neural Networks. Engineering **6**(3), 264–274 (2020). https://doi.org/10.1016/j.eng.2020.01.007
26. Chen, Z., Liu, B.: Lifelong Machine Learning, Second Edition. Morgan & Claypool Publishers (2018)
27. Cheng, W., Rademaker, M., de Baets, B., Hüllermeier, E.: Predicting Partial Orders: Ranking with Abstention. In: Cellular automata. Ed. by S. Bandini, S. Manzoni, H. Umeo, and G. Vizzari, pp. 215–230. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-15880-3_20
28. Chicco, D.: Siamese Neural Networks: An Overview. In: Artificial Neural Networks - Third Edition. Ed. by H.M. Cartwright, pp. 73–94. Springer (2021). https://doi.org/10.1007/978-1-0716-0826-5_3
29. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 1724–1734. ACL (2014). https://doi.org/10.3115/V1/D14-1179
30. Chollet, F. *et al.*: Keras, https://keras.io (2015).
31. Claesen, M., De Moor, B.L.R.: Hyperparameter Search in Machine Learning. CoRR **abs/1502.02127** (2015). arXiv: 1502.02127
32. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to Order Things. J. Artif. Intell. Res. **10**, 243–270 (1999). https://doi.org/10.1613/JAIR.587

33. Dai, A.M., Le, Q.V.: Semi-supervised Sequence Learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pp. 3079–3087 (2015)

34. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics (2019). https://doi.org/10.18653/V1/N19-1423

35. Dietterich, T.G.: Ensemble Methods in Machine Learning. In: Multiple classifier systems. Ed. by J. Kittler, pp. 1–15. Springer (2000). https://doi.org/10.1007/3-540-45014-9_1

36. Dijkman, R.M., Gfeller, B., Küster, J.M., Völzer, H.: Identifying refactoring opportunities in process model repositories. Inf. Softw. Technol. **53**(9), 937–948 (2011). https://doi.org/10.1016/J.INFSOF.2011.04.001

37. Dosilovic, F.K., Brcic, M., Hlupic, N.: Explainable artificial intelligence: A survey. In: Skala, K., Koricic, M., Grbac, T.G., Cicin-Sain, M., Sruk, V., Ribaric, S., Gros, S., Vrdoljak, B., Mauher, M., Tijan, E., Pale, P., Janjic, M. (eds.) 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018, Opatija, Croatia, May 21-25, 2018, pp. 210–215. IEEE (2018). https://doi.org/10.23919/MIPRO.2018.8400040

38. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)

39. van Dun, C.G.J., Moder, L., Kratsch, W., Röglinger, M.: ProcessGAN: Supporting the creation of business process improvement ideas through generative machine learning. Decis. Support Syst. **165**, 113880 (2023). https://doi.org/10.1016/J.DSS.2022.113880

40. van Dyk, D.A., Meng, X.-L.: The Art of Data Augmentation. Journal of Computational and Graphical Statistics **10**(1), 1–50 (2001)

41. Erickson, B.J., Korfiatis, P., Akkus, Z., Kline, T., Philbrick, K.: Toolkits and Libraries for Deep Learning. J. Digit. Imaging **30**(4), 400–405 (2017). https://doi.org/10.1007/S10278-017-9965-6

42. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, pp. 1436–1445. PMLR (2018)

43. Ferludin, O., Eigenwillig, A., Blais, M., Zelle, D., Pfeifer, J., Sanchez-Gonzalez, A., Li, W.L.S., Abu-El-Haija, S., Battaglia, P.W., Bulut, N., Halcrow, J., de Almeida, F.M.G., Lattanzi, S., Linhares, A., Mayer, B.A., Mirrokni, V.S., Palowitch, J., Paradkar, M., She, J., Tsitsulin, A., Villela, K., Wang, L., Wong, D., Perozzi, B.: TF-GNN: Graph Neural Networks in TensorFlow. CoRR **abs/2207.03522** (2022). https://doi.org/10.48550/ARXIV.2207.03522. arXiv: 2207.03522

44. Feurer, M., Hutter, F.: Hyperparameter Optimization. In: Automated Machine Learning - Methods, Systems, Challenges. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren, pp. 3–33. Springer (2019). https://doi.org/10.1007/978-3-030-05318-5_1

45. Fey, M., Lenssen, J.E.: Fast Graph Representation Learning with PyTorch Geometric. CoRR **abs/1903.02428** (2019). arXiv: 1903.02428

46. Forbus, K.D., Gentner, D., Law, K.: MAC/FAC: A Model of Similarity-Based Retrieval. Cogn. Sci. **19**(2), 141–205 (1995). https://doi.org/10.1207/S15516709COG1902_1

47. Fürnkranz, J., Hüllermeier, E.: Preference Learning: An Introduction. In: Preference Learning. Ed. by J. Fürnkranz and E. Hüllermeier, pp. 1–17. Springer (2010). https://doi.org/10.1007/978-3-642-14125-6_1

48. Gabel, T., Godehardt, E.: Top-Down Induction of Similarity Measures Using Similarity Clouds. In: Hüllermeier, E., Minor, M. (eds.) Case-Based Reasoning Research and Development - 23rd International Conference, ICCBR 2015, Frankfurt am Main, Germany, September 28-30, 2015, Proceedings. LNCS, vol. 9343, pp. 149–164. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-24586-7_11

49. Galanti, R., de Leoni, M.: Predictive Analytics for Object-Centric Processes: Do Graph Neural Networks Really Help? In: Weerdt, J.D., Pufahl, L. (eds.) Business Process Management Workshops - BPM 2023 International Workshops, Utrecht, The Netherlands, September 11-15, 2023, Revised Selected Papers. LNBIP, vol. 492, pp. 521–533. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-50974-2_39

50. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1), 113–129 (2010). https://doi.org/10.1007/S10044-008-0141-Y

51. Garnelo, M., Shanahan, M.: Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. Current Opinion in Behavioral Sciences **29**, 17–23 (2019). https://doi.org/10.1016/j.cobeha.2018.12.010

52. Gates, L., Leake, D., Wilkerson, K.: Cases Are King: A User Study of Case Presentation to Explain CBR Decisions. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development - 31st International Conference, ICCBR 2023, Aberdeen, UK, July 17-20, 2023, Proceedings. LNCS, vol. 14141, pp. 153–168. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-40177-0_10

53. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016)

54. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural Message Passing for Quantum Chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, pp. 1263–1272. PMLR (2017)

55. Hammond, K.J.: CHEF: A Model of Case-Based Planning. In: Kehler, T. (ed.) Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science, pp. 267–271. Morgan Kaufmann (1986)

56. Hanga, K.: A Deep Learning Approach to Business Process Mining. PhD thesis, Birmingham City University, Birmingham (2023).

57. Hanney, K., Keane, M.T.: Learning Adaptation Rules from a Case-Base. In: Smith, I.F.C., Faltings, B. (eds.) Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96, Lausanne, Switzerland, November 14-16, 1996, Proceedings. LNCS, vol. 1168, pp. 179–192. Springer, Heidelberg (1996). https://doi.org/10.1007/BFB0020610

58. Hanney, K., Keane, M.T.: The adaptation knowledge bottleneck: How to ease it by learning from cases. In: Case-Based Reasoning Research and Development. Ed. by D.B. Leake and E. Plaza, pp. 359–370. Springer, Berlin and Heidelberg (1997). https://doi.org/10.1007/3-540-63233-6_506

59. Harl, M., Weinzierl, S., Stierle, M., Matzner, M.: Explainable predictive business process monitoring using gated graph neural networks. J. Decis. Syst. **29**(Supplement), 312–327 (2020). https://doi.org/10.1080/12460125.2020.1780780

60. Haslum, P., Lipovetzky, N., Magazzeni, D., Muise, C.: An Introduction to the Planning Domain Definition Language. Morgan & Claypool Publishers (2019)

61. Helmert, M.: The Fast Downward Planning System. CoRR **abs/1109.6051** (2011). arXiv: 1109.6051

62. Hevner, A.R.: A three cycle view of design science research. Scandinavian Journal of Information Systems **19**(2), 4 (2007)

63. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Q. **28**(1), 75–105 (2004)

64. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/NECO.1997.9.8.1735

65. Hoffmann, M., Bergmann, R.: Informed Machine Learning for Improved Similarity Assessment in Process-Oriented Case-Based Reasoning. CoRR **abs/2106.15931** (2021). arXiv: 2106.15931

66. Hoffmann, M., Bergmann, R.: Improving Automated Hyperparameter Optimization with Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 273–288. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_18

67. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

68. Hoffmann, M., Bergmann, R.: Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133039

69. Hoffmann, M., Malburg, L., Bach, N., Bergmann, R.: GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 240–255. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_16

70. Hoffmann, M., Malburg, L., Bergmann, R.: Augmentation of Semantic Processes for Deep Learning Applications. Applied Artificial Intelligence. Taylor and Francis. Submitted for Publication. (2024)

71. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

72. Holt, A., Bichindaritz, I., Schmidt, R., Perner, P.: Medical applications in case-based reasoning. Knowl. Eng. Rev. **20**(3), 289–292 (2005). https://doi.org/10.1017/S0269888906000622

73. Huang, K., Jin, Y., Candès, E.J., Leskovec, J.: Uncertainty Quantification over Graph with Conformalized Graph Neural Networks. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023)

74. Hüllermeier, E.: Case-Based Approximate Reasoning. Springer (2007)

75. Hüllermeier, E., Fürnkranz, J.: Comparison of Ranking Procedures in Pairwise Preference Learning. In: Bouchon-Meunier, B., Coletti, G., Yager, R. (eds.) Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04), Perugia, Italy (2004)

76. Hüllermeier, E., Schlegel, P.: Preference-Based CBR: First Steps toward a Methodological Framework. In: Ram, A., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 19th International Conference on Case-Based Reasoning, ICCBR 2011, London, UK, September 12-15, 2011. Proceedings. LNCS, vol. 6880, pp. 77–91. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23291-6_8

77. Jiang, W., Luo, J.: Graph neural network for traffic forecasting: A survey. Expert Systems with Applications **207**, 117921 (2022). https://doi.org/10.1016/j.eswa.2022.117921

78. Kacimi, F., Tari, A., Kheddouci, H.: Business process graph matching based on vectorial signatures. Int. J. Bus. Process. Integr. Manag. **9**(2), 76–89 (2019). https://doi.org/10.1504/IJBPIM.2019.099869

79. Kampik, T., Warmuth, C., Rebmann, A., Agam, R., Egger, L.N.P., Gerber, A., Hoffart, J., Kolk, J., Herzig, P., Decker, G., van der Aa, H., Polyvyanyy, A., Rinderle-Ma, S., Weber, I., Weidlich, M.: Large Process Models: Business Process Management in the Age of Generative AI. CoRR **abs/2309.00900** (2023). https://doi.org/10.48550/ARXIV.2309.00900. arXiv: 2309.00900

80. Keane, M.T., Kenny, E.M.: How Case-Based Reasoning Explains Neural Networks: A Theoretical Analysis of XAI Using Post-Hoc Explanation-by-Example from a Survey of ANN-CBR Twin-Systems. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 155–171. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_11

81. Kendall, A., Gal, Y.: What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5574–5584 (2017)

82. Kendall-Morwick, J., Leake, D.: A Study of Two-Phase Retrieval for Process-Oriented Case-Based Reasoning. In: Successful Case-based Reasoning Applications-2, pp. 7–27. Springer (2014)

83. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

84. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 188–203. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_13

85. Kolodner, J.L.: Case-Based Reasoning: Proceedings of a Workshop on Case-Based Reasoning. Morgan Kaufmann Publishers, Holiday Inn, Clearwater Beach, Florida (1988)

86. Kudenko, D.: Special Issue on Transfer Learning. Künstliche Intelligenz **28**(1), 5–6 (2014). https://doi.org/10.1007/s13218-013-0289-5

87. Leake, D.: Bridging AI Paradigms with Cases and Networks. Computer Sciences & Mathematics Forum **8**(1) (2023). https://doi.org/10.3390/cmsf2023008071

88. Leake, D., Crandall, D.J.: On Bringing Case-Based Reasoning Methodology to Deep Learning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 343–348. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_22

89. Leake, D., Schack, B.: Exploration vs. Exploitation in Case-Base Maintenance: Leveraging Competence-Based Deletion with Ghost Cases. In: Cox, M.T., Funk, P., Begum, S. (eds.) Case-Based Reasoning Research and Development - 26th International Conference, ICCBR 2018, Stockholm, Sweden, July 9-12, 2018, Proceedings. LNCS, vol. 11156, pp. 202–218. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01081-2_14

90. Leake, D., Wilkerson, Z., Ye, X., Crandall, D.J.: Enhancing Case-Based Reasoning with Neural Networks. In: Compendium of Neurosymbolic Artificial Intelligence. Ed. by P. Hitzler, M.K. Sarker, and A. Eberhart, pp. 387–409. IOS Press (2023). https://doi.org/10.3233/FAIA230150

91. Leake, D., Ye, X., Crandall, D.J.: Supporting Case-Based Reasoning with Neural Networks: An Illustration for Case Adaptation. In: Martin, A., Hinkelmann, K., Fill, H., Gerber, A., Lenat, D., Stolle, R., van Harmelen, F. (eds.) Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021. CEUR Workshop Proceedings. CEUR-WS.org (2021)

92. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539

93. Lenz, M., Bergmann, R.: Case-Based Adaptation of Argument Graphs with WordNet and Large Language Models. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development - 31st International Conference, ICCBR 2023, Aberdeen, UK, July 17-20, 2023, Proceedings. LNCS, vol. 14141, pp. 263–278. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-40177-0_17

94. Lenz, M., Ollinger, S., Sahitaj, P., Bergmann, R.: Semantic Textual Similarity Measures for Case-Based Retrieval of Argument Graphs. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 219–234. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_15

95. Leonardi, G., Montani, S., Striani, M.: Process Trace Classification for Stroke Management Quality Assessment. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 49–63. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_4

96. Lewis, P.S.H., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)

97. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, pp. 3835–3845. PMLR (2019)

98. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.S.: Gated Graph Sequence Neural Networks. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016)

99. Liao, C., Liu, A., Chao, Y.: A Machine Learning Approach to Case Adaptation. In: First IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018, Laguna Hills, CA, USA, September 26-28, 2018, pp. 106–109. IEEE Computer Society (2018). https://doi.org/10.1109/AIKE.2018.00023

100. Liu, T.-Y.: Learning to Rank for Information Retrieval. Springer, Berlin and Heidelberg (2011)

101. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Netw. Model. Anal. Health Informatics Bioinform. **5**(1), 18 (2016). https://doi.org/10.1007/S13721-016-0125-6

102. Malburg, L., Brand, F., Bergmann, R.: Adaptive Management of Cyber-Physical Workflows by Means of Case-Based Reasoning and Automated Planning. In: Sales, T.P., Proper, H.A., Guizzardi, G., Montali, M., Maggi, F.M., Fonseca, C.M. (eds.) Enterprise Design, Operations, and Computing. EDOC 2022 Workshops - IDAMS, SoEA4EE, TEAR, EDOC Forum, Demonstrations Track and Doctoral Consortium, Bozen-Bolzano, Italy, October 4-7, 2022, Revised Selected Papers. LNBIP, vol. 466, pp. 79–95. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-26886-1_5

103. Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. **61**(1), 83–111 (2023). https://doi.org/10.1007/S10844-022-00766-W

104. Malburg, L., Klein, P., Bergmann, R.: Converting semantic web services into formal planning domain descriptions to enable manufacturing process planning and scheduling in industry 4.0. Eng. Appl. Artif. Intell. **126**, 106727 (2023). https://doi.org/10.1016/J.ENGAPPAI.2023.106727

105. Marie, F., Corbat, L., Chaussy, Y., Delavelle, T., Henriet, J., Lapayre, J.: Segmentation of deformed kidneys and nephroblastoma using Case-Based Reasoning and Convolutional Neural Network. Expert Syst. Appl. **127**, 282–294 (2019). https://doi.org/10.1016/J.ESWA.2019.03.010

106. Marjieh, R., Sucholutsky, I., Sumers, T.R., Jacoby, N., Griffiths, T.: Predicting Human Similarity Judgments Using Large Language Models. In: Culbertson, J., Rabagliati, H., Ramenzoni, V.C., Perfors, A. (eds.) Proceedings of the 44th Annual Meeting of the Cognitive Science Society, CogSci 2022, Toronto, ON, Canada, July 27-30, 2022. cognitivesciencesociety.org (2022)

107. Martin, K., Wiratunga, N., Sani, S., Massie, S., Clos, J.: A Convolutional Siamese Network for Developing Similarity Knowledge in the SelfBACK Dataset. In: Sánchez-Ruiz, A.A., Kofod-Petersen, A. (eds.) Proceedings of ICCBR 2017 Workshops (CAW, CBRDL, PO-CBR), Doctoral Consortium, and Competitions co-located with the 25th International Conference on Case-Based Reasoning (ICCBR 2017), Trondheim, Norway, June 26-28, 2017. CEUR Workshop Proceedings, pp. 85–94. CEUR-WS.org (2017)

108. Mathisen, B.M., Aamodt, A., Bach, K., Langseth, H.: Learning similarity measures from data. Prog. Artif. Intell. **9**(2), 129–143 (2020). https://doi.org/10.1007/S13748-019-00201-2

109. Mathisen, B.M., Aamodt, A., Langseth, H.: Data Driven Case Base Construction for Prediction of Success of Marine Operations. In: Sánchez-Ruiz, A.A., Kofod-Petersen, A. (eds.) Proceedings of ICCBR 2017 Workshops (CAW, CBRDL, PO-CBR), Doctoral Consortium, and Competitions co-located with the 25th International Conference on Case-Based Reasoning (ICCBR 2017), Trondheim, Norway, June 26-28, 2017. CEUR Workshop Proceedings, pp. 104–113. CEUR-WS.org (2017)

110. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. **51**(11), 8107–8118 (2021). https://doi.org/10.1007/S10489-021-02251-3

111. McDermott, D., Ghallab, M., Howe, A.E., Knoblock, C.A., Ram, A., Veloso, M.M., Weld, D.S., Wilkins, D.E.: PDDL - The Planning Domain Definition Language: Technical Report CVC TR-98-003/DCS TR-1165, (1998).

112. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014). https://doi.org/10.1016/J.IS.2013.06.004

113. Morgan, N., Bourlard, H.: Generalization and Parameter Estimation in Feedforward Netws: Some Experiments. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989], pp. 630–637. Morgan Kaufmann (1989)

114. Morrison, E.D., Menzies, A., Koliadis, G., Ghose, A.K.: Business Process Integration: Method and Analysis. In: Kirchberg, M., Link, S. (eds.) Conceptual Modelling 2009, Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009), Wellington, New Zealand, January 20-23 2009. CRPIT, pp. 29–37. Australian Computer Society (2009)

115. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer, Wiesbaden (2018)

116. Müller, G., Bergmann, R.: A Cluster-Based Approach to Improve Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, pp. 639–644. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-639

117. Neu, D.A., Lahann, J., Fettke, P.: A systematic literature review on state-of-the-art deep learning methods for process prediction. Artif. Intell. Rev. **55**(2), 801–827 (2022). https://doi.org/10.1007/S10462-021-09960-8

118. Nguyen, B., Mutum, D.S.: A review of customer relationship management: successes, advances, pitfalls and futures. Bus. Process. Manag. J. **18**(3), 400–419 (2012). https://doi.org/10.1108/14637151211232614

119. Niesen, T., Dadashnia, S., Fettke, P., Loos, P.: A Vector Space Approach to Process Model Matching using Insights from Natural Language Processing. In: (2016)

120. Ontañón, S.: An Overview of Distance and Similarity Functions for Structured Data. CoRR **abs/2002.07420** (2020). arXiv: 2002.07420

121. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 8024–8035 (2019)

122. Pauli, J., Hoffmann, M., Bergmann, R.: Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning Using Graph Neural Networks and Transfer Learning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133040

123. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. J. Manag. Inf. Syst. **24**(3), 45–77 (2008). https://doi.org/10.2753/MIS0742-1222240302

124. Peters, M.E., Ruder, S., Smith, N.A.: To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks. In: Augenstein, I., Gella, S., Ruder, S., Kann, K., Can, B., Welbl, J., Conneau, A., Ren, X., Rei, M. (eds.) Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019, pp. 7–14. Association for Computational Linguistics (2019). https://doi.org/10.18653/V1/W19-4302

125. Pfeiffer, P., Lahann, J., Fettke, P.: Multivariate Business Process Representation Learning Utilizing Gramian Angular Fields and Convolutional Neural Networks. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings. LNCS, vol. 12875, pp. 327–344. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-85469-0_21

126. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep Learning for Predictive Business Process Monitoring: Review and Benchmark. IEEE Transactions on Services Computing **16**(01), 739–756 (2023). https://doi.org/10.1109/TSC.2021.3139807

127. Reijers, H., Liman Mansar, S.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega **33**(4), 283–306 (2005). https://doi.org/10.1016/j.omega.2004.04.012

128. Richter, M.M.: Foundations of Similarity and Utility. In: Wilson, D., Sutcliffe, G. (eds.) Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7-9, 2007, Key West, Florida, USA, pp. 30–37. AAAI Press (2007)

129. Richter, M.M., Weber, R.O.: Case-Based Reasoning - A Textbook. Springer (2013)

130. Rosa, M.L., Dumas, M., Uba, R., Dijkman, R.M.: Business Process Model Merging: An Approach to Business Process Consolidation. ACM Trans. Softw. Eng. Methodol. **22**(2), 11:1–11:42 (2013). https://doi.org/10.1145/2430545.2430547

131. Roth-Berghofer, T.: Knowledge maintenance of case-based reasoning systems - the SIAM methodology. PhD thesis, Kaiserslautern University of Technology, Germany (2003).

132. Roth-Berghofer, T., Cassens, J.: Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems. In: Muñoz-Avila, H., Ricci, F. (eds.) Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Proceedings. LNCS, vol. 3620, pp. 451–464. Springer, Heidelberg (2005). https://doi.org/10.1007/11536406_35

133. von Rüden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Pfrommer, J., Pick, A., Ramamurthy, R., Walczak, M., Garcke, J., Bauckhage, C., Schuecker, J.: Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. IEEE Trans. Knowl. Data Eng. **35**(1), 614–633 (2023). https://doi.org/10.1109/TKDE.2021.3079836

134. Salamó, M., Golobardes, E.: Dynamic Case Base Maintenance for a Case-Based Reasoning System. In: Lemaître, C., García, C.A.R., González, J.A. (eds.) Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, Mexico, November 22-26, 2004, Proceedings. LNCS, vol. 3315, pp. 93–103. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30498-2_10

135. Salamó, M., López-Sánchez, M.: Adaptive case-based reasoning using retention and forgetting strategies. Knowledge-Based Systems **24**(2), 230–247 (2011). https://doi.org/10.1016/j.knosys.2010.08.003

136. Sánchez, L.Q., Bridge, D.G., Díaz-Agudo, B., Recio-García, J.A.: A Case-Based Solution to the Cold-Start Problem in Group Recommenders. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, pp. 3042–3046. IJCAI/AAAI (2013)

137. Sani, S., Wiratunga, N., Massie, S.: Learning Deep Features for kNN-Based Human Activity Recognition. In: Sánchez-Ruiz, A.A., Kofod-Petersen, A. (eds.) Proceedings of ICCBR 2017 Workshops (CAW, CBRDL, PO-CBR), Doctoral Consortium, and Competitions co-located with the 25th International Conference on Case-Based Reasoning (ICCBR 2017), Trondheim, Norway, June 26-28, 2017. CEUR Workshop Proceedings, pp. 95–103. CEUR-WS.org (2017)

138. Sarker, M.K., Zhou, L., Eberhart, A., Hitzler, P.: Neuro-symbolic artificial intelligence. AI Commun. **34**(3), 197–209 (2021). https://doi.org/10.3233/AIC-210084

139. Schoenborn, J.M., Weber, R.O., Aha, D.W., Cassens, J., Althoff, K.-D.: Explainable Case-Based Reasoning: A Survey. In: Madumal, P., Tulli, S., Weber, R., Aha, D. (eds.) Proceedings for the Explainable Agency in AI Workshop at the 35th AAAI Conference on Artificial Intelligence, The Internet (2021)

140. Schoknecht, A., Thaler, T., Fettke, P., Oberweis, A., Laue, R.: Similarity of Business Process Models - A State-of-the-Art Analysis. ACM Comput. Surv. **50**(4), 52:1–52:33 (2017). https://doi.org/10.1145/3092694

141. Schroff, F., Kalenichenko, D., Philbin, J.: FaceNet: A unified embedding for face recognition and clustering. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pp. 815–823. IEEE Computer Society (2015). https://doi.org/10.1109/CVPR.2015.7298682

142. Schuler, N., Hoffmann, M., Beise, H., Bergmann, R.: Semi-supervised Similarity Learning in Process-Oriented Case-Based Reasoning. In: Bramer, M., Stahl, F.T. (eds.) Artificial Intelligence XL - 43rd SGAI International Conference on Artificial Intelligence, AI 2023, Cambridge, UK, December 12-14, 2023, Proceedings. LNCS, vol. 14381, pp. 159–173. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-47994-6_12

143. Schultheis, A., Hoffmann, M., Malburg, L., Bergmann, R.: Explanation of Similarities in Process-Oriented Case-Based Reasoning by Visualization. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development - 31st International Conference, ICCBR 2023, Aberdeen, UK, July 17-20, 2023, Proceedings. LNCS, vol. 14141, pp. 53–68. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-40177-0_4

144. Schultheis, A., Zeyen, C., Bergmann, R.: An Overview and Comparison of Case-Based Reasoning Frameworks. In: Case-Based Reasoning Research and Development - 31st International Conference, ICCBR 2023, Aberdeen, Scotland, July 17-20, 2023, Proceedings. LNCS, vol. 14141, pp. 327–343. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-40177-0_21

145. Seiger, R., Huber, S., Heisig, P., Aßmann, U.: Toward a framework for self-adaptive workflows in cyber-physical systems. Softw. Syst. Model. **18**(2), 1117–1134 (2019). https://doi.org/10.1007/S10270-017-0639-0

146. Smyth, B., Keane, M.T.: Adaptation-Guided Retrieval: Questioning the Similarity Assumption in Reasoning. Artif. Intell. **102**(2), 249–293 (1998). https://doi.org/10.1016/S0004-3702(98)00059-9

147. Sonntag, A., Hake, P., Fettke, P., Loos, P.: An Approach for Semantic Business Process Model Matching using Supervised Machine Learning. In: 24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016, Research–in–Progress Paper 47 (2016)

148. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014). https://doi.org/10.5555/2627435.2670313

149. Stahl, A., Bergmann, R.: Applying Recursive CBR for the Customization of Structured Products in an Electronic Shop. In: Blanzieri, E., Portinale, L. (eds.) Advances in Case-Based Reasoning, pp. 297–308. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

150. Stewart, G.: Supply-chain operations reference model (SCOR): the first cross-industry framework for integrated supply-chain management. Logistics Information Management **10**(2), 62–67 (1997). https://doi.org/10.1108/09576059710815716

151. Sungkono, K.R., Sarno, R., Salsabila, M.C., Dewi, C.P.: A Graph-based Method for Merging Business Process Models by Considering Semantic Similarity. International Journal of Intelligent Engineering and Systems **16**(2), 166–175 (2023). https://doi.org/10.22266/ijies2023.0430.14

152. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A Survey on Deep Transfer Learning. In: Kurková, V., Manolopoulos, Y., Hammer, B., Iliadis, L.S., Maglogiannis, I. (eds.) Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III. LNCS, vol. 11141, pp. 270–279. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01424-7_27

153. van der Aalst, W., Carmona, J. (eds.): Process mining handbook. Springer, Cham (2022)

154. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)

155. Venugopal, I., Töllich, J., Fairbank, M., Scherp, A.: A Comparison of Deep-Learning Methods for Analysing and Predicting Business Processes. In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021, pp. 1–8. IEEE (2021). https://doi.org/10.1109/IJCNN52387.2021.9533742

156. Vidgof, M., Bachhofner, S., Mendling, J.: Large Language Models for Business Process Management: Opportunities and Challenges. In: Francescomarino, C.D., Burattin, A., Janiesch, C., Sadiq, S.W. (eds.) Business Process Management Forum - BPM 2023 Forum, Utrecht, The Netherlands, September 11-15, 2023, Proceedings. LNBIP, vol. 490, pp. 107–123. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-41623-1_7

157. Waikhom, L., Patgiri, R.: A survey of graph neural networks in various learning paradigms: methods, applications, and challenges. Artif. Intell. Rev. **56**(7), 6295–6364 (2023). https://doi.org/10.1007/S10462-022-10321-2

158. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. CoRR **abs/1909.01315** (2019). arXiv: 1909.01315

159. Watson, I.D.: Case-based reasoning is a methodology not a technology. Knowl. Based Syst. **12**(5-6), 303–308 (1999). https://doi.org/10.1016/S0950-7051(99)00020-9

160. Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE. Ed. by J.A.B. Jr., J. Krogstie, O. Pastor, B. Pernici, C. Rolland, and A. Sølvberg, pp. 381–395. Springer (2013). https://doi.org/10.1007/978-3-642-36926-1_31

161. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022)

162. Weidlich, M., Dijkman, R.M., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13094-6_37

163. Wilkerson, Z., Leake, D., Crandall, D.J.: On Combining Knowledge-Engineered and Network-Extracted Features for Retrieval. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) Case-Based Reasoning Research and Development - 29th International Conference, ICCBR 2021, Salamanca, Spain, September 13-16, 2021, Proceedings. LNCS, vol. 12877, pp. 248–262. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-86957-1_17

164. Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P.: Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. CoRR **abs/2004.09602** (2020). arXiv: 2004.09602

165. Wu, L.Y., Fisch, A., Chopra, S., Adams, K., Bordes, A., Weston, J.: StarSpace: Embed All The Things! In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 5569–5577. AAAI Press (2018). https://doi.org/10.1609/AAAI.V32I1.11996

166. Wu, L., Cui, P., Pei, J., Zhao, L. (eds.): Graph Neural Networks: Foundations, Frontiers, and Applications. Springer Singapore, Singapore (2022)

167. Wu, S., Sun, F., Zhang, W., Xie, X., Cui, B.: Graph Neural Networks in Recommender Systems: A Survey. ACM Comput. Surv. **55**(5), 97:1–97:37 (2023). https://doi.org/10.1145/3535101

168. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A Comprehensive Survey on Graph Neural Networks. IEEE Trans. Neural Networks Learn. Syst. **32**(1), 4–24 (2021). https://doi.org/10.1109/TNNLS.2020.2978386

169. Yager, R.R.: “A Unified View of Case Based Reasoning and Fuzzy Modeling.” In: Fuzzy Logic Foundations and Industrial Applications. Ed. by D. Ruan. Boston, MA: Springer US, 1996, pp. 5–26. ISBN: 978-1-4613-1441-7. https://doi.org/10.1007/978-1-4613-1441-7_1.

170. Ye, X., Leake, D., Crandall, D.: Case Adaptation with Neural Networks: Capabilities and Limitations. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 143–158. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_10

171. Ye, X., Leake, D., Jalali, V., Crandall, D.J.: Learning Adaptations for Case-Based Classification: A Neural Network Approach. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) Case-Based Reasoning Research and Development - 29th International Conference, ICCBR 2021, Salamanca, Spain, September 13-16, 2021, Proceedings. LNCS, vol. 12877, pp. 279–293. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-86957-1_19

172. Yuan, H., Yu, H., Gui, S., Ji, S.: Explainability in Graph Neural Networks: A Taxonomic Survey. IEEE Trans. Pattern Anal. Mach. Intell. **45**(5), 5782–5799 (2023). https://doi.org/10.1109/TPAMI.2022.3204236

173. Zeyen, C., Bergmann, R.: A*-Based Similarity Assessment of Semantic Graphs. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 17–32. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_2

174. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 388–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_26

175. Zhao, Z., Zheng, P., Xu, S., Wu, X.: Object Detection With Deep Learning: A Review. IEEE Trans. Neural Networks Learn. Syst. **30**(11), 3212–3232 (2019). https://doi.org/10.1109/TNNLS.2018.2876865

176. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020). https://doi.org/10.1016/J.AIOPEN.2021.01.001

177. Zhou, Y., Zeng, J.: Massively Parallel A* Search on a GPU. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, pp. 1248–1255. AAAI Press (2015). https://doi.org/10.1609/AAAI.V29I1.9367

178. Zhou, Y., Zheng, H., Huang, X.: Graph Neural Networks: Taxonomy, Advances and Trends. CoRR **abs/2012.08752** (2020). arXiv: 2012.08752

Part II

Publications

Chapter 5

Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning

Bibliographic Information

Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

Abstract

Similarity-based retrieval of semantic graphs is a core task of *Process-Oriented Case-Based Reasoning (POCBR)* with applications in real-world scenarios, e.g., in smart manufacturing. The involved similarity computation is usually complex and time-consuming, as it requires some kind of inexact graph matching. To tackle these problems, we present an approach to modeling similarity measures based on embedding semantic graphs via *Graph Neural Networks (GNNs)*. Therefore, we first examine how arbitrary semantic graphs, including node and edge types and their knowledge-rich semantic annotations, can be encoded in a numeric format that is usable by GNNs. Given this, the architecture of two generic graph embedding models from the literature is adapted to enable their usage as a similarity measure for similarity-based retrieval. Thereby, one of the two models is more optimized towards fast similarity prediction, while the other model is optimized towards knowledge-intensive, more expressive predictions. The evaluation examines the quality and performance of these models in preselecting retrieval candidates and in approximating the ground-truth similarities of a graph-matching-based similarity measure for two semantic graph domains. The results show the great potential of the approach for use in a retrieval scenario, either as a preselection model or as an approximation of a graph similarity measure.

Keywords

Case-Based Reasoning · Process-Oriented Case-Based Reasoning · Graph Embedding · Siamese Graph Neural Networks · Similarity-Based Retrieval · Neural Networks · Graph Encoding · MAC/FAC Retrieval

Copyright Notice

5.1 Introduction

Case-Based Reasoning (CBR) [1, 51] is used widely across different domains, e.g., for cooking assistance [47], in smart manufacturing [41], and for data mining support [57]. CBR is a methodology for problem solving where problems and their respective solutions (bundled as *cases* in a *case base*) are used to solve upcoming problems (*queries*). This process relies on similarity computations that are harnessed to find the best-matching case with regard to a given query. It is assumed that the solution of a case can be reused for the given query if the problem of the case is similar to the problem in the query. These core principles are reflected in the CBR cycle [1], which describes four phases of CBR applications: a case is retrieved from the case base according to a given query, reused as a candidate solution, revised to check the fit for the current problem, and retained to be used for further problems. A well-working similarity-based retrieval tool is especially important because, as the first phase, it influences the results of the subsequent phases. A subfield of CBR, which is called *Process-Oriented Case-Based Reasoning (POCBR)* [8, 10, 45], focuses on the application of CBR methods and principles in process and workflow management. POCBR particularly aims at managing procedural experiential knowledge, e.g., retrieving workflow models from large repositories [8]. The involved processes are usually represented as semantic graphs [23, 41, 57], which adds complexity to all involved tasks. For instance, in similarity-based retrieval, the main influential factor on performance is the definition of the similarity measures and the underlying case representation. A simple case representation in the form of feature vectors can be assessed in linear time, while most similarity measures between semantic graphs rely on some form of subgraph isomorphism check, which is computationally expensive, usually with polynomial or exponential complexity [8, 13, 49]. This example highlights the need for efficient similarity measures between semantic graphs since similarity computations are ubiquitous in POCBR.

One approach to tackle this problem is to reduce the complexity of the semantic graph case representation and to compute similarities with a simplified representation, e.g., by using similarity measures on constructed feature vectors of graphs [11]. However, this reveals a new challenge of identifying important features of semantic graphs, at best, without the need for manual modeling, since it results in increased knowledge acquisition effort [24]. To mitigate this problem and to enable automatic representation learning for semantic graphs, *Deep Learning (DL)* [35] can be used. A key factor of success for DL is the wide applicability to different types of data, ranging from simple data in the form of one-dimensional feature vectors to more complex data such as graph-structured objects [21, 53]. A popular DL method for processing graph-structured data is referred to as graph embedding [14, 22]. Graph embedding models are neural networks that learn to represent a graph or its elements in a low-dimensional latent space. The resulting vector representation of embedded graphs enables downstream algorithms to work more efficiently compared to being used on the complex graph structure. Therefore, graph embedding methods are suitable to address the challenge of automatically learning a simplified case representation in POCBR. Further, by adding a component to the neural networks that transforms pairs of embedded graphs to a similarity value, these models can also be directly used as similarity measures.

As a first step towards this goal, our previous work [31] presents the use of an embedding technique in POCBR that accelerates the retrieval process while maintaining almost the same level of retrieval quality as other manually modeled approaches (e.g., [11, 48]). The approach uses a general-purpose embedding framework that embeds graph nodes individually, based on graph triplets of two nodes and an edge between them. Thereby, however, only the graph structure is considered in the embedding process, with no integration of the semantic annotations of nodes and edges. Due to this shortcoming, our subsequent approaches [26, 27] pursue the idea of using *Graph Neural Networks (GNNs)* for embedding in POCBR, focusing on the embedding of semantic annotations as well as the graph structure. These publications are extended in this paper to describe graph embedding in POCBR in a broader context. The focus is on integrating embedding methods of semantic graphs as similarity measures in case retrieval. Our main contributions are:

- a comprehensive encoding scheme that enables the integration of semantic graphs with their semantic annotations and structure to be used in GNNs;
- two specialized, adapted GNN architectures for learning similarities between semantic graphs, based on GNNs from the literature [37];
- an evaluation of the GNNs in different retrieval scenarios with regard to performance and quality.

The remainder of the paper is organized as follows: Section 5.2 presents the foundations of our semantic graph format and the graph matching algorithm for similarity assessment between these semantic graphs. In addition, two variants of similarity-based retrieval are introduced and related work is discussed. In Section 5.3, we present the architecture of two GNNs, introduced by Li *et al.* [37], that serve as the basis for our adapted GNNs. Section 5.4 then elaborates on the encoding procedure for our semantic graph format that transforms the available semantic information into numeric vector space encodings. Furthermore, we present how to adapt the GNNs from Section 5.3 to be used for predicting pairwise graph similarities. Section 5.5 describes a system architecture for integrating the GNN-based similarity measures into a POCBR framework and the underlying process of case retrieval. Additionally, Section 5.6 evaluates the adapted GNNs in the context of different similarity-based retrieval scenarios. Eventually, the conclusions of this paper, together with future research directions, are given in Section 5.7.

5.2 Foundations and Related Work

The prerequisite for applying graph embedding techniques in POCBR is handling the underlying data representation in the form of semantic graphs, which is introduced in Section 5.2.1. In addition, we explain the similarity assessment procedure for pairs of these semantic graphs (see Section 5.2.2), which plays a key role in similarity-based retrieval (see Section 5.2.3). Further, related work is presented to highlight previous and current developments in the use of embedding techniques in *Case-Based Reasoning* (*CBR*; see Section 5.2.4).

5.2.1 Semantic Graph Representation

POCBR applications are characterized by a high degree of modeled knowledge. Our semantic graph format allows the integration of semantic knowledge within graph structures. The format is mainly used to model processes and workflows in various domains (e.g., [23, 41, 47, 57]). We represent all cases and queries as semantically annotated directed graphs referred to as *NEST* graphs, introduced by Bergmann and Gil [8]. More specifically, a *NEST* graph is a quadruple $G = (N,E,S,T)$ that is composed of a set of nodes N and a set of edges $E \subseteq N \times N$. Each node and each edge has a specific type from $\mathcal{T}$ that is indicated by the function $T : N \cup E \to \mathcal{T}$. Additionally, the function $S : N \cup E \to \mathcal{S}$ assigns a semantic description from $\mathcal{S}$ (*semantic metadata language*, e.g., an ontology) to nodes and edges. Whereas nodes and edges are used to build the structure of each graph, types and semantic descriptions are additionally used to model semantic information. Hence, each node and each edge can have a semantic description. We denote the number of nodes in a graph as $|N| \in \mathbb{R}$ and the number of edges as $|E| \in \mathbb{R}$.

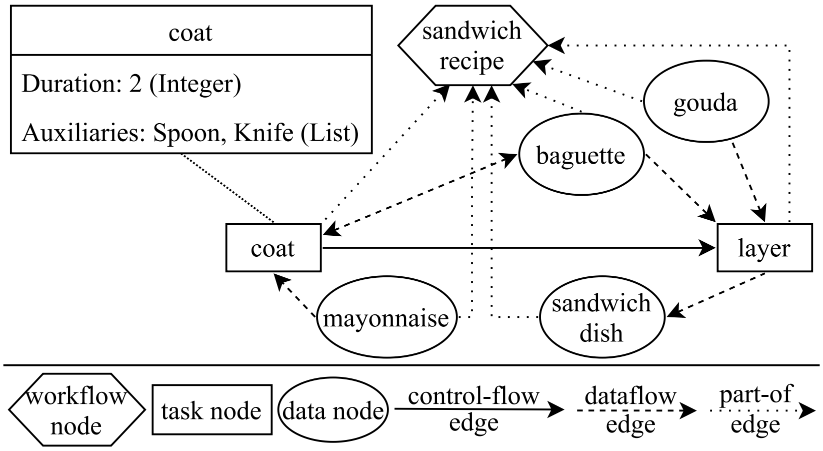


Figure 5.1: Exemplary cooking recipe represented as a NEST graph.

Figure 5.1 shows a simple example of a *NEST* graph that represents a cooking recipe for making a sandwich. The mayonnaise–gouda sandwich is prepared by executing the cooking steps `coat` and `layer` (task nodes) with the ingredients `mayonnaise`, `baguette`, `sandwich dish`, and `gouda` (data nodes). All components are linked by edges that indicate relations, e.g., `mayonnaise` is consumed by `coat`. Semantic descriptions of task nodes and data nodes are used to further specify semantic information belonging to the recipe components. Figure 5.1 shows an example of the semantic description of the task node `coat`. The provided information is used to describe the task more precisely. In this case, a spoon and a knife are needed to execute the task (`Auxiliaries`) and the estimated time that the task takes is two minutes (`Duration`). The definition of NEST graphs [8] does not strictly specify the contents of the semantic descriptions in $\mathcal{S}$. To frame the scope of this work regarding semantic descriptions, we refer to the definitions of the POCBR framework ProCAKE [9] since it is widely used in the POCBR literature (e.g., [26, 27, 31, 36, 57]). Entries of semantic descriptions in ProCAKE have a certain *data type* and a *content* (or value). For instance, the attribute `Duration` within the semantic description of `coat` (see Figure 5.1) has the content `2` and the data type `Integer`. The available semantic descriptions can be divided into *atomic* and *composite* ones, denoted as $\mathcal{S}_{\text{atom}} \subset \mathcal{S}$ and $\mathcal{S}_{\text{comp}} \subset \mathcal{S}$. The atomic descriptions comprise strings, numerics (integer and double), timestamps, and booleans and represent simple base data. Composite descriptions are more complex and define a structure (or relations) over other composite or atomic data. The composite data comprise attribute–value pairs (dictionaries), lists, and sets. For instance, the semantic description in our example is itself a list of attribute–value pairs, where the attribute `Duration` is an integer (atomic) and the attribute `Auxiliaries` is a list (composite). This complexity and flexibility in representation is not easy to handle for DL models (see [6, 27, 32] for more details) and requires specific data encoding methods that will be discussed in our approach.

5.2.2 Similarity Assessment of Semantic Graphs

Determining the similarity between two *NEST* graphs, i.e., a query graph QG and a case graph CG, requires a similarity measure that assesses the structure of nodes and edges as well as the semantic descriptions and types of these components. Bergmann and Gil [8] propose a semantic similarity measure that determines a similarity based on the local–global principle [50]. A global similarity, i.e., the similarity between two graphs, is composed of local similarities, i.e., the pairwise similarities of nodes and edges. The similarity between two nodes with identical types is defined as the similarity of the semantic descriptions of these nodes. The similarity value between two edges with identical types is not only composed of the similarity between the semantic descriptions of these edges, but, in addition, the similarity of the connected nodes is taken into account. For instance, the similarity between the data node `baguette` from Figure 5.1 and an arbitrary node from another graph is determined by first checking if their node types are equal. If this is the case, the similarity between both nodes is defined as the similarity between the semantic descriptions of both nodes. The similarity of the node `baguette` is also part of the similarity assessment of all connected edges, e.g., the dataflow edge to the task node `coat`. The local–global principle [50] is also harnessed to compute similarities between semantic descriptions of nodes and edges. This means that the global similarity between two semantic descriptions is composed of the local similarities according to the structure of atomic and composite data of these semantic descriptions. When considering the task node `coat` from Figure 5.1 as an example, the global similarity between `coat` and another arbitrary task node with the same structure is composed of the local similarities between the values of `Duration` and `Auxiliaries`. The similarity of `Auxiliaries` is, in turn, dependent on the similarities of individual list items. To make use of the local–global principle for similarity assessment, similarity measures for all components of the semantic descriptions are part of the similarity knowledge associated with the domain. This knowledge usually stems from domain experts and specifically takes into account the type of data [7], e.g., Levenshtein distance for strings and *Mean Absolute Error (MAE)* for numerics. A common domain model is also required for all graphs and their elements in order to allow the definition of similarity measures between objects that have a comparable structure. The global similarity of the two graphs $sim(QG, CG)$ is finally calculated by finding an injective partial mapping that maximizes the aggregated local similarities of all pairs of mapped nodes and edges.

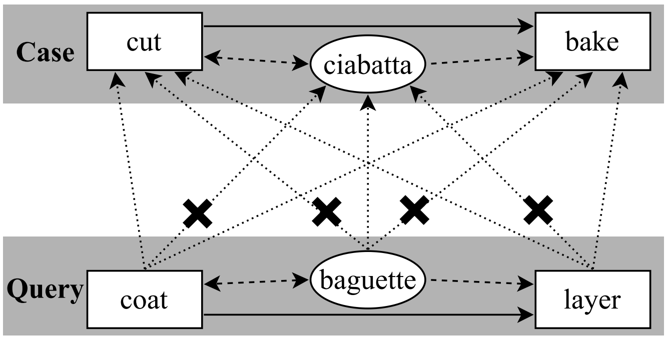


Figure 5.2: Mapping procedure of nodes with illegal mappings.

The possible node mappings between an excerpt of the graph from Figure 5.1 and another arbitrary graph are visualized in Figure 5.2. It highlights the algorithm's property that nodes and edges with different types are not allowed to be mapped (depicted with bold crosses). That is, the mapping process is constrained according to the types of nodes and edges. Additionally, the figure points out the increase in complexity that comes with the number of nodes and edges, as every node and edge can be mapped onto many other nodes and edges. The complexity of finding a mapping that maximizes the global similarity between a query QG and a single case CG is tackled by utilizing an A* search algorithm, also introduced by Bergmann and Gil [8]. However, A* search is usually time-consuming and can lead to long retrieval times [27, 31, 49, 56]. Therefore, the algorithm features an adjustable parameter for setting the maximum number of partially mapped solutions (called *MaxPMS*). Adjusting MaxPMS serves as a trade-off between the optimality of the mappings and the time required for finding them, i.e., reducing MaxPMS results in solutions that are not optimal at a lower computation time and vice versa. In most practical applications of the A* search, the search space is very large, which makes it unfeasible to search for a solution to the mapping problem if MaxPMS is unlimited.

5.2.3 Similarity-Based Retrieval of Semantic Graphs

While the similarity assessment that is shown in Section 5.2.2 can be used to determine a single similarity between two semantic graphs, it is more common to use it in the *retrieve* phase of CBR. A retrieval aims to find the most similar cases from a case base CB for a given query QG. The most similar cases are determined by computing the pairwise similarity between the query and each case from the case base, before ranking the cases in descending order according to the computed similarities. Many CBR applications only consider the k-most similar cases as result of the retrieval, which is similar to the well-known *k-nearest neighbors* (kNN) algorithm. Although the retrieval algorithm itself has a computational complexity of $O(\mid CB \mid)$, the retrieval time is usually dominated by the similarity measure that is used to compare the cases. For instance, measures that perform any kind of inexact subgraph matching (such as the similarity measure introduced in Section 5.2.2) belong to the class of NP-complete algorithms [8, 13, 49] and thus have a great impact on performance. To overcome possible performance issues in retrieval situations, the approach of *MAC/FAC* ("Many are called but few are chosen"), introduced by Forbus, Gentner, and Law [19], can be used. MAC/FAC is a two-staged retrieval approach that aims to decrease the computation time by pre-filtering the case base in the MAC phase to reduce the number of cases that have to be evaluated by a (potentially) computationally complex similarity measure in the subsequent FAC phase. The MAC phase is parameterized by giving a specific similarity measure and a filter size. The approximating similarity measure of the MAC phase usually has low computational complexity and is knowledge-poor, thus introducing the trade-off between quality and performance of the similarity assessment. The filter size is a parameter of the MAC/FAC algorithm that can be used to control how many of the most similar cases according to the MAC similarity measure are transferred to the FAC phase. In general, the results from the MAC stage can be seen as *candidates* that might be part of the most similar cases regarding the query. With the pre-filtered cases as an input, the FAC phase applies the computationally intensive similarity measure on these candidates. The result is a list of the same length as the filter size that is not guaranteed to contain the most similar cases of the case base according to the query graph. Further truncation to the k-most similar cases can be performed analogously to the standard retrieval. Eventually, MAC/FAC is a trade-off between retrieval quality and retrieval time. The (possibly) decreased quality results from the pre-filtering that might not return the same graphs that a standard retrieval would have returned. The decreased retrieval time is due to a smaller amount of graph pairs that have to be evaluated by the computationally complex, knowledge-intensive similarity measure in the FAC phase. This points out the importance of a well-designed and well-integrated MAC phase as it significantly influences the results of using a MAC/FAC implementation [19, 29]. Related MAC/FAC approaches in the context of POCBR use similarity measures that are defined on clustered representations of the case base [48], manually modeled graph features [11], and embeddings of graph triplets [31].

5.2.4 Related Work

Several approaches have been proposed in CBR research that use DL methods as a key component. There is also strong interest in the combination of DL and CBR methods in the community [32]. To the best of our knowledge, only the work of Klein, Malburg, and Bergmann [31] utilizes a DL-based embedding procedure in the subfield of POCBR. They use a general-purpose embedding framework to learn vector representations in an unsupervised manner, based on the structural properties of semantic graphs, such as the relation between task and data nodes. The approach is evaluated as a similarity measure in retrieval scenarios, where it outperforms other automatically learned approaches and achieves comparable performance to other approaches that are manually modeled. Our approaches in this paper and previous work [26, 27] differ from the work of Klein, Malburg, and Bergmann [31] as we not only consider structural graph information for the embedding procedure but also the semantic information of nodes and edges. Outside of POCBR, Mathisen, Bach, and Aamodt [43] and Amin *et al.* [3] use embedding techniques and Siamese neural networks in CBR retrieval. The approaches train neural networks to learn similarity measures that are applied in the domains of aquaculture and customer support management, respectively.

The following approaches are related to our work in the broader sense due to their integration of DL components into CBR or vice versa. Most of the approaches automatically learn a similarity measure, similar to our approach. Corchado and Lees [17] integrate a neural network into CBR retrieval and reuse phases, where the network is dynamically retrained during runtime with regard to the current query. The application is used to predict water temperatures along a sea route. Dieterle and Bergmann [18] use neural networks for several tasks within their CBR application that predicts prices of domain names, e.g., for the feature weighting of case attributes. Mathisen *et al.* [42] investigate methods for learning similarity measures from data. They propose two different strategies with different levels of required manual effort, where the measure with minimal manual modeling outperforms other methods. The application of DL methods in case adaptation is discussed by several other approaches, e.g., [34, 39]. These two exemplary approaches pursue the idea of using neural networks for case adaptation by learning to transfer differences between case problems to the respective case solutions. Leake and Ye [33] advance this idea by taking into account that retrieval knowledge and adaptation knowledge are related in CBR. Therefore, they use a specific algorithm to optimize according to both aspects when training neural networks. Furthermore, several papers, such as Gabel and Godehardt [20] and Keane and Kenny [28], tackle a major drawback of DL applications when compared to CBR applications: the reduced explainability. The former approach addresses the problem by projecting DL predictions onto real cases before using them. The latter approach tries to explain the predictions of DL methods with CBR components in a twin-systems approach.

5.3 Neural Networks for Graph Embedding

For embedding semantic graphs, we rely on GNNs that process graph nodes and edges to represent the graph in a low-dimensional latent space. The specific GNN architecture that our approach is based on is presented by Li *et al.* [37]. Although there are several different GNN architectures presented in the literature (see [6] for an overview of some approaches), that of Li et al. is chosen as a foundation, since they specifically describe and evaluate their approach for the task of similarity-based search, which is closely related to our domain. They describe two GNN variants that are called *Graph Embedding Model (GEM)* and *Graph Matching Network (GMN).* Both neural networks work with message-passing between nodes and their features along the edge structure [21, 53] as core learning mechanisms. In addition, both neural networks have a Siamese architecture [5, 12] that is used to embed two graphs with shared weights and apply a vector similarity measure on the embedding vectors of both graphs. In the following, we introduce the general structure of GEM and GMN (see Section 5.3.1) and explain the specific components in more detail (see Sections 5.3.2–5.3.4).

5.3.1 General Neural Network Structure

The general setup of the neural networks is composed of four main components that are put together in successive order (see Figure 5.3).

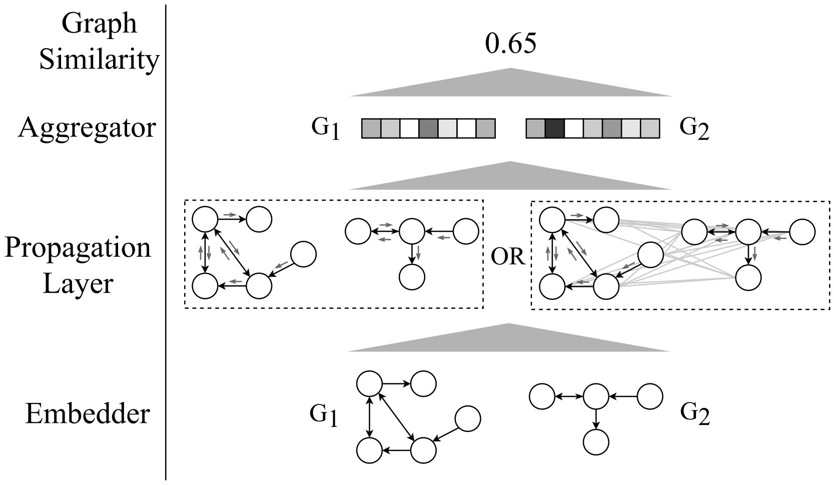


Figure 5.3: Graph Embedding Model (GEM; **left branch**) and Graph Matching Network (GMN; **right branch**) (derived from [37]).

These components are the *embedder*, the *propagation layer*, the *aggregator*, and the *graph similarity*. The embedder transforms the raw graph input data into an initial vector representation, resulting in a single embedding vector for each node and edge, respectively. During propagation, the vector-based node information is iteratively merged according to the edge structure of the graphs. Thereby, the node embedding vectors are updated via element-wise summations according to the information of all incoming edges and the nodes that are connected via these edges. This captures information about the local neighborhood of each node in its embedding vector. After propagating information for a certain number of rounds, the aggregator combines the embeddings of all nodes from each graph to form a single whole-graph embedding. The final component computes a single scalar similarity value with the embedding vectors of both graphs by utilizing a vector similarity measure such as the cosine similarity. The difference between both models comes from a different propagation strategy. The GEM uses an isolated propagation strategy that only propagates information within a single graph. In contrast, the GMN uses an attention-based cross-graph matching component that propagates information across both graphs in an early state of similarity assessment. In the following, all four components are described more formally and in more detail. The inputs of each neural network are two graph structures $G_1 = (N_1,E_1)$ and $G_2 = (N_2,E_2)$, although a general graph $G = (N,E)$ with nodes N and edges $E \subseteq N \times N$ is used as a placeholder for components in the Siamese architecture that operate on a single graph.

5.3.2 Embedder

The embedder computes an initial embedding for all nodes $v, w \in N$ and edges $(v, w) \in E$ to be used in the propagation phase. It is assumed that all nodes and edges have a numeric feature vector that expresses the contained information of a node or an edge. The functions $feat_{\text{node}} : N \to \mathbb{R}^{l_n}$ and $feat_{\text{edge}} : E \to \mathbb{R}^{l_e}$ map nodes and edges to their vector encoding in the l_n-dimensional and l_e-dimensional real space, respectively. For instance, when dealing with NEST graphs, these functions should encode the type and the semantic description of a node or an edge. We discuss the implementation of these functions for our case in Section 5.4.1 and stick to the generic definition at this point. The embedding procedure can be defined as follows:

$$\begin{aligned} h_v^{(0)} = &\ \text{MLP}_{\text{node}}(feat_{\text{node}}(v)), \quad \forall v \in N \\ h_{v,w} = &\ \text{MLP}_{\text{edge}}(feat_{\text{edge}}(v,w)), \quad \forall (v, w) \in E \end{aligned} \tag{5.1}$$

Each node v is mapped to an embedding vector $h_v^{(0)} \in \mathbb{R}^{l_{nEmb}}$ that is part of an l_{nEmb}-dimensional vector space. This embedding vector has the value of transformation step 0 (i.e., the initial embedding), which is depicted by the superscript 0. The mapping is performed by a *Multi-Layer Perceptron (MLP)* that can be configured in terms of the number of layers, number of hidden neurons, and so on. The edges are mapped in the same fashion. The feature vector of each edge is embedded into $h_{v,w} \in \mathbb{R}^{l_{eEmb}}$ by a separate MLP. The resulting edge embedding is located in the l_{eEmb}-dimensional vector space. In contrast to node embeddings, edge embeddings do not have an associated transformation step (i.e., superscript 0), because edge representations remain unchanged after the embedding by the embedder. The embeddings of node features are simultaneously used as the initial representation for nodes in the following propagation component (see Section 5.3.3). They are constantly updated by the neural network, which is indicated by the assigned transformation step, e.g., $h_v^{(1)}$ after the first update and $h_v^{(5)}$ after the fifth update. Furthermore, the values of l_{nEmb} and l_{eEmb} (referred to as *embedding size*) can be used to parameterize the MLPs. The larger the embedding size, the more information can be stored in the particular embedding vector but the more time it takes to compute these vectors.

5.3.3 Propagation Layer

With the initial embeddings of all nodes and edges, the propagation layer can be applied. This layer differs significantly for GEM and GMN. Therefore, the propagation concept of the GEM will be explained first, and the propagation concept of the GMN afterwards. The propagation layer of the GEM uses edges between nodes to iteratively update the representation of these nodes. In a single one of the K propagation steps, the representation of each node is updated.

$$m_{w,v} = \text{NN}_{\text{message}}([h_v^{(k)}, h_w^{(k)}, h_{v,w}]), \quad \forall (v,w) \in E \tag{5.2}$$

$m_{w,v}$ is the representation of a single message between nodes. It is composed of the concatenated inputs of the node representation of v, w, and the representation of the edge $e_{v,w}$, transformed by the neural network layer $\text{NN}_{\text{message}}$. The type of this layer can be freely chosen but an MLP is recommended by Li *et al.* [37]. As follows, the message representation $m_{w,v}$ is used to update all node representations from the current state $h_v^{(k)}$ to the next state $h_v^{(k+1)}$:

$$h_v^{(k+1)} = \text{NN}_{\text{node}} \left(\left[h_v^{(k)}, \sum_{(w,v) \in E} m_{w,v} \right] \right), \quad \forall v \in N \tag{5.3}$$

The neural network layer NN_{node} from Equation (5.3) takes as inputs a concatenation of the current node state of node v, i.e., $h_v^{(k)}$, and an aggregation of the message representations of all incoming edges of node v (see Equation (5.2)). This aggregation can be any commutative element-wise vector operator, e.g., sum, mean, maximum, or minimum. The layer NN_{node} can either be an MLP or a recurrent layer core such as a *Gated Recurrent Unit (GRU)* [16] and a *Long Short-Term Memory (LSTM)* [25]. The GMN approaches propagation from a different point of view to the GEM. Whereas the GEM only considers messages (in the form of edges) within a single graph, the GMN matches nodes from both graphs at an early stage of the similarity assessment.

$$att(h_v^{(k)}, h_w^{(k)}) = \frac{e^{vsim(h_v^{(k)}, h_w^{(k)})}}{\sum\limits_{w' \in N : N \ni w} e^{vsim(h_v^{(k)}, h_{w'}^{(k)})}} \tag{5.4}$$

Equation (5.4) shows the computation of the function *att* that computes attention weights using a softmax attention (see [4, 55] for a comprehensive explanation of attention in neural networks). In this case, the softmax function can be interpreted as the indication of relevance that the node representation $h_w^{(k)}$ has for matching with the node representation $h_v^{(k)}$. The function *vsim* can be any vector similarity measure, e.g., cosine similarity, that yields a scalar value. To indicate the attention weights of a node from one graph to all nodes of another graph, the descriptor *attmat* is used. For instance, $attmat(v \in G_1)$ represents a matrix where each row contains the pairwise attention weights (as computed by the function *att*) of the node v from graph G_1 to other nodes from graph G_2.

$$h_v^{(k+1)} = \text{NN}_{\text{node}} \left(\left[h_v^{(k)}, \sum_{(w,v) \in E_1} (m_{w,v}), \, h_v^{(k)} \ominus \sum_{w' \in N_2} attmat(v) \cdot h_{w'}^{(k)} \right] \right) \tag{5.5}$$

These attention weights are then used in the process of constructing node representations for the next transformation step, i.e., $h_v^{(k+1)}$, by extending the update process of the GEM (see Equation (5.5)). The extended function NN_{node} of the GMN takes a third parameter that adds a *cross-graph matching component*. The cross-graph matching component is made up of the attention weights introduced by Equation (5.4). The matrix of attention weights of node v, i.e., $attmat(v)$ is multiplied with the representation vector of all nodes from G_2, i.e., $attmat(v) \cdot h_{w'}^{(k)}$. The resulting vectors are finally aggregated and subtracted element-wise from the node representation of node v. Taking the difference in this case intuitively expresses the distance between node v and the closest neighbor in the other graph [37].

5.3.4 Aggregator and Graph Similarity

The next step after the propagation layer is the aggregator. The aggregator aggregates all node representations from graph G that result from the K-th iteration of propagation to form a single whole-graph representation h_G.

$$h_G = \text{MLP}_G \left(\sum_{v \in N} \left(\sigma\big(\text{MLP}_{\text{gate}}(h_v^{(K)})\big) \odot \text{MLP}_{\text{state}}(h_v^{(K)}) \right) \right) \tag{5.6}$$

The aggregation approach in Equation (5.6) originates from Li *et al.* [38]. It describes a weighted sum that uses gating vectors as a method to differentiate relevant from irrelevant node representations. At the single node level, the function MLP_{gate} transforms the node representation $h_v^{(K)}$ to a gating vector. This gating vector is then activated with the softmax function σ (similar to Equation (5.4)). $\text{MLP}_{\text{state}}$ transforms the node representation into a new vector that can then be used in an element-wise multiplication (indicated by the symbol $\odot$), resulting in the final single-node representation. The representations of all individual nodes are afterwards combined by an element-wise sum (indicated by the sum symbol Σ) to form a single vector. This single vector finally acts as an input for MLP_G, leading to the whole-graph representation h_G. h_G is an element of $\mathbb{R}^{l_g}$, whereas the vector length l_g can be parameterized and contributes to a trade-off between representation quality (i.e., increased l_g) and computation time (i.e., decreased l_g).

To obtain a single scalar from the whole-graph representation of the two graphs, represented as the final similarity $\mathit{fsim}(h_{G_1}, h_{G_2})$, it is necessary to apply a vector similarity measure on the graph vectors (see Equation (5.7)).

$$\mathit{fsim}(h_{G_1}, h_{G_2}) = \mathit{vsim}(h_{G_1}, h_{G_2}) \tag{5.7}$$

Li *et al.* [37] do not specify the type of vector similarity measure. Common candidates are cosine similarity, dot product, and Euclidean distance. To have a similarity value that is bounded between [0,1], it is recommended to use the cosine similarity in combination with a ReLU activation of MLP_G.

5.4 Neural-Network-Based Semantic Graph Similarity Measure

To apply the GEM and the GMN (introduced in Section 5.3) as a similarity measure in a retrieval scenario, the neural networks have to be suitable for handling the involved data, i.e., pairs of semantic graphs as input data and a similarity value as output data. To establish a proper integration, we first define an encoding scheme for our semantic graphs that involves encoding the semantic descriptions and types (see Section 5.4.1). The GEM and the GMN are then adapted to process the encoded semantic graphs and be used as a similarity measure in similarity-based retrieval (see Section 5.4.2).

5.4.1 Encoding Semantic Graphs

An arbitrary NEST graph [8] to encode G has the following four components: the nodes N, the edges E, the semantic descriptions $\mathcal{S}$, and the types $\mathcal{T}$. When looking at the introduced neural networks in Section 5.3, it is apparent that they are capable of handling the structure of nodes and edges but are not designed to process the rich semantic information and the types of nodes and edges. Li *et al.* [37] assume nodes and edges to be represented by a feature vector, which is expressed by the functions $\mathit{feat}_{\text{node}} : N \to \mathbb{R}^{l_n}$ and $\mathit{feat}_{\text{edge}} : E \to \mathbb{R}^{l_e}$. Since they do not specify how this feature vector is generated, our approach describes specific encoding methods for the types $\mathcal{T}$ (see Section 5.4.1) and the semantic descriptions $\mathcal{S}$ (see Section 5.4.1) in the following. Together with the graph structure that is natively integrated into both GNNs, this makes it possible to process our semantic graph format.

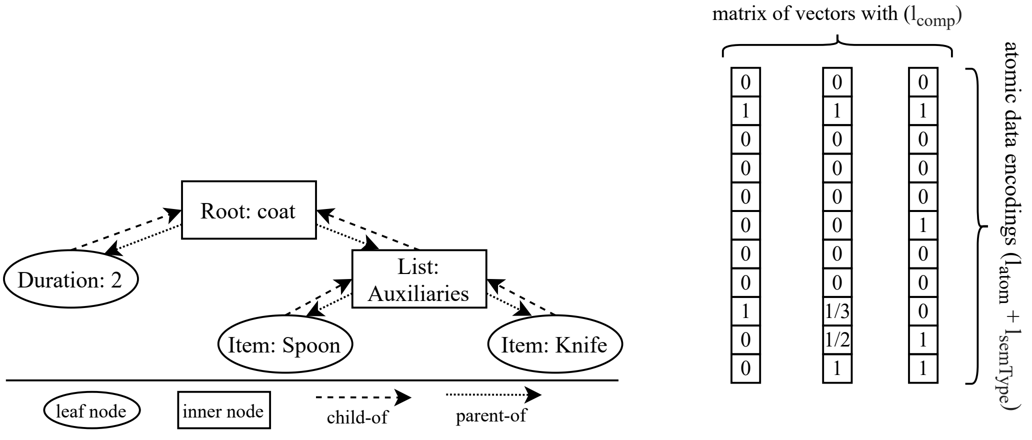
Encoding Node and Edge Types

The types of nodes and edges (see Section 5.2.1) are encoded as they are an integral part of semantic graphs. This is also crucial because the types are used to distinguish nodes and edges in terms of similarity. Only nodes and edges with identical types are mapped during similarity computation (see Section 5.2.2). Each node and each edge has exactly one type, whereas the amount of possible different types, for both nodes and edges, is small. Hence, the types are encoded in the form of *one-hot encodings*, which is represented by the function $enc_{\text{type}} : \mathcal{T} \rightarrow \mathbb{R}^{l_{\text{type}}}$. According to the NEST definition (see Section 5.2.1 and [8]), there are, in total, nine different node and edge types, which leads to $l_{\text{type}} = 9$. The exemplary graph contains six of these nine different type encodings that are specified by all types in the legend of Figure 5.1. Nodes or edges with the same type always have the same type of encoding.

Encoding Semantic Descriptions

When encoding semantic descriptions, it is important to describe the underlying structure of these data. As introduced in Section 5.2.1 and illustrated by the task node `coat` from Figure 5.1, semantic descriptions are composed of atomic and composite data with a data type and the respective data content. The possible arrangement of composite types holding atomic types, other nested composite types holding more nested types, and so on can be visualized as a *hierarchical tree structure.*

Figure 5.4a shows our semantic description used as a running example in tree form. The depiction distinguishes between leaf nodes (ovals) and inner nodes (rectangles), as well as child-of (dashed lines) and parent-of (dotted lines) relations. The relations between these nodes are defined according to the hierarchical structure of the semantic description, e.g., the item `Spoon` is a child of the list of `Auxiliaries`. More formally, the tree representation of the semantic description can be redefined as a graph $G_{\text{sem}} = (N_{\text{comp}} \subseteq \mathcal{S}_{\text{comp}}, N_{\text{atom}} \subseteq \mathcal{S}_{\text{atom}}, E_{\text{sem}} \subseteq (N_{\text{comp}} \cup N_{\text{atom}}) \times (N_{\text{comp}} \cup N_{\text{atom}}))$ with two sets of nodes and edges between these nodes. The encoding procedure covers the encoding data type and content of entries within the semantic description. This enables the neural network to learn to distinguish different semantic information with regard to both aspects. For instance, in the semantic description of `coat`, the integer `2` is used as a `Duration` of type integer. The same number could also be used in a different context (e.g., as the required skill of this task on a 0 to 5 scale), which makes encoding the data type in combination with the content crucial.


Root: coat
Duration: 2
List: Auxiliaries
Item: Spoon
Item: Knife
leaf node
inner node
child-of
parent-of
matrix of vectors with (l$_{\text{comp}}$)
atomic data encodings (l$_{\text{atom}}$ + l$_{\text{semType}}$)

0	0	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	1
0	0	0
0	0	0
1	1/3	0
0	1/2	1
0	1	1

(**a**) (**b**)

Figure 5.4: Encoding of composite types: (**a**) Tree encoding (**b**) Sequence encoding

We encode the data type as a one-hot vector with the function $enc_{\text{semType}} : \mathcal{S} \rightarrow \mathbb{R}^{l_{\text{semType}}}$. Given a total of five atomic types and three composite types, all vectors have a common vector length of $l_{\text{semType}} = 8$ elements. Please note that we are referring to the number of data types of the POCBR framework ProCAKE [8] with these numbers. In general, the approach is generic and it allows the use of any other framework, as well. This encoding procedure can be extended in certain ways: First, it is possible to encode other atomic or composite data types in the same way. If the atomic types are extended by a data type for timestamps, for instance, it is possible to add a new one-hot encoding for this extension. Second, only encoding the base types, i.e., string, float, list, etc., can be insufficient in certain scenarios where the domain model is very complex (see [41, 57] for examples of such domains). The proposed approach can be extended in these cases to use individual one-hot encodings for specific data types in the domain model. For instance, the string entries in the list of `Auxiliaries` might be defined in a taxonomy, which motivates the use of an individual one-hot vector instead of the generic type vector that is used for all strings.

The second part of encoding semantic descriptions deals with their content. Due to the variety of different types of data, it is difficult to encode these data into a common vector space. This requirement also hinders the use of established existing methods out of the box, e.g., one-hot encoding for string vocabularies [44] or a single scalar value for numerics. We propose to encode the content of one atomic data entry as a single vector with a common vector length l_{atom} by the function $enc_{\text{atom}} : \mathcal{S}_{\text{atom}} \rightarrow \mathbb{R}^{l_{\text{atom}}}$. The elements of this vector are set by a specific encoding method for each data type. Due to space restrictions in this paper, the specific encoding methods cannot be presented in detail. In total, our implementation contains the following specific encoding methods for atomic types: strings that are defined by an enumeration, strings that are defined by a taxonomy, integers, doubles, booleans, and timestamps. Please note that the length of the vector l_{atom} has to be equal for all encoding procedures to maintain a tensor structure and to allow processing by a neural network.

Encoding composite data mainly focuses on encoding the structure of the atomic data. To be more flexible with defining the architecture of our GNNs, we present two means of encoding composite data that vary in their complexity and require different neural networks for processing: *tree encoding* to be processed by a GNN and *sequence encoding* to be processed by a *Recurrent Neural Network* (*RNN*; e.g., a LSTM [25] or a GRU [16]). The tree encoding method preserves the hierarchical structure of the semantic descriptions as modeled in $G_{\text{sem}} = (N_{\text{comp}}, N_{\text{atom}}, E_{\text{sem}})$. This means that we encode the information provided by both types of nodes, as well as the information provided by the edges. Regarding the nodes from N_{atom}, we can reuse the functions enc_{semType} and enc_{atom} for encoding the data type and the content of the respective node. The content of the nodes of N_{comp} is usually characterized by the atomic vectors that they are composed of. For instance, the list of `Auxiliaries` is described by the items in this list. Nevertheless, it can be useful to consider content encodings for composite types in case there is specific information to encode, e.g., the number of list items or their order. We still omit encoding the content and, thus, elements of N_{comp} are only encoded with the function enc_{semType}. The encoding of relations between nodes (E_{sem}) represents the type of relation, i.e., parent-of or child-of. Consequently, it is encoded as a one-hot encoding of length two, given by the function $enc_{\text{edge}} : E_{\text{sem}} \rightarrow \mathbb{R}^2$.

The tree encoding method aims at preserving as much semantic and structural information from the semantic descriptions and the underlying domain model as possible. The drawbacks of its usage are the complex encoding procedure and the need for processing the semantic descriptions with complex GNNs. To mitigate these drawbacks, we provide an alternative encoding method where G_{sem} is encoded as a *sequence of atomic types* that can be processed by RNNs. The function $enc_{\text{seq}} : N_{\text{comp}} \rightarrow \mathbb{R}^{l_{\text{comp}} \times l_{\text{atom}} + l_{\text{semType}}}$ is used for this purpose. Each vector in this sequence of atomic data is a concatenation of the data type and the content. Figure 5.4b shows the encoded sequence of our running example. It contains three atomic encoding vectors that represent the three atomic types of the semantic description given by N_{atom}. Since the sequence encoding does not explicitly model the structure of the semantic description, it is represented by the order of the atomic encoding vectors in the sequence. Therefore, it is necessary to define an ordering scheme for the sequence of each composite type. The only requirement for this ordering scheme is that the atomic vectors are always ordered deterministically. This is important to produce the same encodings for the same semantic descriptions when randomness is present, e.g., for atomic types of an unordered set type. In total, our implementation contains the following specific encoding methods for composite types: attribute–value pairs, lists, and sets.

5.4.2 Adapted Neural Network Structure

We adjust the architecture of GEM and GMN by Li *et al.* [37] (see Section 5.3) for our purpose of similarity-based retrieval in POCBR. The adjusted models are further denoted as *Semantic Graph Embedding Model (sGEM)* and *Semantic Graph Matching Network (sGMN)*. Resulting from the specific encoding scheme of semantic graphs (see Section 5.4.1), we present an adjusted embedder (see Section 5.4.2). In addition, the propagation layer (see Section 5.4.2) and the final graph similarity of the GMN (see Section 5.4.2) are also adapted to create a more guided learning process that closely resembles the similarity assessment between semantic graphs. Eventually, we present the loss function that is used for training the neural networks to predict similarities (see Section 5.4.2).

Adapted Embedder

Introduced in Section 5.3.2, the embedder creates initial embeddings for nodes and edges. Li *et al.* [37] assume nodes and edges to be represented by a simple feature vector, which is expressed by the functions $\mathit{feat}_{\text{node}} : N \to \mathbb{R}^{l_n}$ and $\mathit{feat}_{\text{edge}} : E \to \mathbb{R}^{l_e}$. The challenge for processing our semantic graph format with sGEM and sGMN is how to bridge the gap between our more complex encoded information, i.e., one-hot encoded types and tree- or sequence-encoded semantic descriptions, to the simple one-dimensional feature vector. To achieve this, the adapted embedder embeds the information of semantic descriptions and types separated from each other and concatenates both vectors to form a single feature vector for a node or an edge.

$$\begin{aligned} \mathit{feat}_{\text{node}}(v) = \quad & h_v^{(0)} = \left[h_v^{\text{type}}, h_v^{\text{sem}}\right], \quad \forall v \in N \\ \mathit{feat}_{\text{edge}}(v,w) = \quad & h_{v,w} = \left[h_{v,w}^{\text{type}}, h_{v,w}^{\text{sem}}\right], \quad \forall (v,w) \in E \end{aligned} \tag{5.8}$$

Equation (5.8) shows how the embeddings of types, i.e., h_v^{type} and $h_{v,w}^{\text{type}}$, and semantic descriptions, i.e., h_v^{sem} and $h_{v,w}^{\text{sem}}$, are combined for each node and edge. This shows how the features of nodes and edges are put together from the available information. In the following, the embedding of types and semantic descriptions is described. The types are embedded as shown in Equation (5.9):

$$\begin{aligned} h_v^{\text{type}} = \quad & \text{MLP}_{\text{nType}}(\mathit{enc}_{\text{type}}(T(v))), \quad \forall v \in N \\ h_{v,w}^{\text{type}} = \quad & \text{MLP}_{\text{eType}}(\mathit{enc}_{\text{type}}(T(v,w))), \quad \forall (v,w) \in E \end{aligned} \tag{5.9}$$

The equation shows the process of embedding node types (h_v^{type}) and edge types ($h_{v,w}^{\text{type}}$). Nodes and edges utilize separate MLPs in this process, i.e., $\text{MLP}_{\text{nType}}$ for node types and $\text{MLP}_{\text{eType}}$ for edge types, which are applied to the one-hot encoded type vectors. The second part of the combined feature vectors deals with embedding the semantic description of a node or an edge. Since we propose two different encoding methods for semantic descriptions, i.e., a sequence encoding and a tree encoding (see Section 5.4.1), these two encodings are also handled differently.

$$\begin{aligned} h_v^{\text{sem}} =&\ \text{RNN}_{\text{nSem}}(enc_{\text{seq}}(S(v))), \quad \forall v \in N \\ h_{v,w}^{\text{sem}} =&\ \text{RNN}_{\text{eSem}}(enc_{\text{seq}}(S(v,w))), \quad \forall (v,w) \in E \end{aligned} \tag{5.10}$$

Equation (5.10) shows the embedding of semantic descriptions of nodes (h_v^{sem}) and edges ($h_{v,w}^{\text{sem}}$) that are encoded as a sequence. The sequence structure of the semantic description's encodings is processed by an unrolled RNN (e.g., [16, 25]). An RNN can handle sequences of inputs with different lengths, as they are present in the encodings of semantic descriptions. We use the output state of the last step as the result of the unrolled RNN, which is a single vector. Please note that the sequence contains concatenations of content and data type of semantic description entries (see Section 5.4.1) such that the respective embedding vector captures this information.

When using semantic descriptions that are encoded with the tree encoding method, the embedding step is not implemented with RNNs. Instead, the graph $G_{\text{sem}} = (N_{\text{comp}}, N_{\text{atom}}, E_{\text{sem}})$ (introduced in Section 5.4.1) is processed by utilizing a simple GNN that is similar to the GNN architecture that also processes the entire semantic graph. Therefore, the reader can consult the explanations in Section 5.3 for more details on certain steps. The GNN consists of several trainable components: a component that embeds composite nodes from N_{comp}, i.e., NN_{comp}, a component that embeds atomic nodes from N_{atom}, i.e., NN_{atom}, a component that embeds edges from E_{sem}, i.e., NN_{edge}, a component that performs message propagation, i.e., NN_{prop}, and a component that aggregates the node information to a single graph representation vector, i.e., NN_{agg}. The embedding process is defined as follows:

$$\begin{aligned} htree_v^{(0)} =&\ \text{NN}_{\text{comp}}(enc_{\text{semType}}(S(v))), \quad \forall v \in N_{\text{comp}} \\ htree_v^{(0)} =&\ \text{NN}_{\text{atom}}([enc_{\text{semType}}(S(v)), enc_{\text{atom}}(S(v))]), \quad \forall v \in N_{\text{atom}} \\ htree_{v,w} =&\ \text{NN}_{\text{edge}}(enc_{\text{edge}}(v,w)), \quad \forall (v,w) \in E_{\text{sem}} \end{aligned} \tag{5.11}$$

In the first step (see Equation (5.11)), all nodes and edges are embedded, which results in the initial embedding vectors $htree_v^{(0)}$ for each node $v \in N_{\text{atom}} \cup N_{\text{comp}}$ and $htree_{v,w}$ for each edge $(v,w) \in E_{\text{sem}}$. Thereby, we use the data type and content for embedding nodes from N_{atom} and only the data type for embedding nodes from N_{comp}. Embedding vectors of edges from E_{sem} capture the encoded information of the edge representing a child-of or a parent-of relation. The propagation component passes messages between the nodes and updates their state vector accordingly.

$$\begin{aligned} mtree_{w,v} =& [htree_v^{(k)}, htree_w^{(k)}, htree_{v,w}], \quad \forall (v,w) \in E_{\text{sem}} \\ htree_v^{(k+1)} =& \text{NN}_{\text{prop}}\left(\left[htree_v^{(k)}, \sum_{(w,v)\in E_{\text{sem}}} mtree_{w,v}\right]\right), \quad \forall v \in N_{\text{comp}} \cup N_{\text{atom}} \end{aligned} \tag{5.12}$$

Equation (5.12) shows the propagation process, where an updated state vector $htree_v^{(k+1)}$ is computed for each node $v \in N_{\text{comp}} \cup N_{\text{atom}}$ in each propagation step $k \le K_{\text{tree}}$. This specifies the message-passing step of the GNN, where node information is shared across the edges within the graph. We propose to set the maximum number of propagation steps to 5 based on recommendations from the literature [21, 37], i.e., $K_{\text{tree}} = 5$.

$$htree_G = \text{NN}_{\text{agg}} \left(\sum_{v \in N_{\text{comp}} \cup N_{\text{atom}}} h_v^{(K_{\text{tree}})} \right) \tag{5.13}$$

The final step of the GNN is the aggregation of all node vectors to a single embedding vector that represents the embedded information of the semantic description, i.e., $htree_G$. Equation (5.13) shows the aggregation where the state vectors of all nodes after the final propagation step K_{tree} are summed up with a commutative element-wise vector function (e.g., sum, mean, etc.) to a single vector. This vector is eventually passed through the neural network NN_{agg}. Please note that the embedding procedure of semantic descriptions in the tree encoding representation is identical for the semantic descriptions of nodes and edges. Thus, the explanations of Equations (5.11) to (5.13) hold for nodes as well as edges and $htree_G$ is a representative for h_v^{sem} and $h_{v,w}^{\text{sem}}$. Together with the embedded nodes, this completes $h_v^{(0)}$ and $h_{v,w}$ from Equation (5.8).

Constrained Propagation in the Semantic Graph Matching Network

The propagation layer of the GMN uses a cross-graph matching method between the nodes of the two graphs G_1 and G_2 (see Section 5.3.3). Each node of G_1 is compared with each node of G_2 through their softmax-activated cosine vector similarity (ranging between 0 and 1). In this way, information is propagated between the nodes of the two graphs with regard to their pairwise attention. According to the definition of the used graph matching algorithm (see Section 5.2.2 and [8]), only nodes and edges with the same type are allowed to be matched. That is, the matching process is constrained according to the types of nodes and edges. These *constraints* can be integrated as an alternative means of message-passing into the cross-graph propagation component of sGMN by only allowing a cosine similarity greater than zero for nodes of the same type. All pairs of nodes with different types are assigned a similarity of 0, representing maximum dissimilarity. Thus, their attention is close to 0, which leads to almost no information propagation between nodes of different types. This extension of GMN's original cross-graph propagation component can be seen as a form of *informed machine learning* [52], where prior knowledge is used within a machine learning setup such as our sGMN. See our previous work [26] for a discussion and an application scenario of informed machine learning in POCBR.

Trainable Graph Similarity of Semantic Graph Matching Network

In the definition of GEM and GMN, a vector similarity measure being applied on the vector representations of two graphs G_1 and G_2, i.e., $vsim(h_{G_1}, h_{G_2})$, computes the final similarity value. Such a vector similarity measure is purely syntactic and not trainable regarding patterns in these vectors. Therefore, an MLP-based approach is used to compute the pairwise similarity of two graphs in the sGMN. This emphasizes the nature of the sGMN as a more knowledge-intensive measure (compared to the sGEM) [37] and trades a higher computation time for a better-quality similarity assessment.

$$fsim_{\text{sGMN}}(h_{G_1}, h_{G_2}) = \sigma(\text{MLP}_{\text{fsim}}([h_{G_1}, h_{G_2}])) \tag{5.14}$$

The approach (see Equation (5.14)) applies an MLP on the concatenated vector representations of the two graphs. The result of this transformation is lastly activated by a logistic function (σ) to keep the final similarity in the range of [0,1]. It is important to note that this MLP-based pairwise graph similarity is not symmetrical, which means that, for most graph pairs,

$\mathit{fsim}_{\text{sGMN}}(h_{G_1}, h_{G_2}) \neq \mathit{fsim}_{\text{sGMN}}(h_{G_2}, h_{G_1})$. This is also a property of the graph matching algorithm (see Section 5.2.2) and other similarity measures in POCBR [7] that is only reflected by this MLP-based graph similarity and not by the cosine vector similarity measure used in the sGEM.

Training and Optimization

Whereas Li *et al.* [37] assume their training examples to be pairs of graphs that are labeled as similar or dissimilar for training, our training examples are graph pairs with ground-truth similarity values. Hence, each of our training graph pairs is labeled with the ground-truth similarity value, in the range of [0,1]. This yields a regression-like problem where the neural network aims to predict similarities that are close to the ground-truth similarities. For this regression problem, the use of a *Mean Squared Error (MSE)* loss is suitable. When given a batch B of graph pairs $(G_1, G_2) \in B$, the ground-truth similarity of each graph pair $\mathit{sim}(G_1, G_2) \in [0,1]$, and the predicted similarity from one of our models for each graph pair $\mathit{fsim}(h_{G_1}, h_{G_2})$, the MSE loss L_{mse} is computed as follows:

$$L_{\text{mse}} = \frac{\sum\limits_{(G_1,G_2)\in B} (\mathit{sim}(G_1,G_2) - \mathit{fsim}(h_{G_1}, h_{G_2}))^2}{|B|} \tag{5.15}$$

The MSE sums up all squared differences in predicted similarity $\mathit{fsim}(h_{G_1}, h_{G_2})$ and labeled similarity $\mathit{sim}(G_1,G_2)$, and then divides this value by the number of all batched training examples to obtain the average deviation. As originally proposed by Li *et al.* [37], sGEM and sGMN also use the Adam optimizer [30] to optimize the networks' weights according to the MSE loss value.

5.5 Application of Semantic Graph Embedding Model and Semantic Graph Matching Network in Similarity-Based Retrieval

The sGEM and the sGMN are designed to be used as trainable similarity measures—that is, learning how to predict the similarities of pairs of NEST graphs. The possible application scenarios of sGEM and sGMN in POCBR are widespread since similarity measures are used in several different tasks, e.g., for retrieving, adapting, or retaining cases [1, 51]. We restrict ourselves to the application of the embedding models for retrieving the k-most similar cases according to a query (case retrieval) in this work. Thereby, the integration of the neural networks into a standard retrieval scenario and a MAC/FAC retrieval scenario is examined (see Section 5.2.3 for an introduction to retrieval and [27, 31] for previous work on MAC/FAC in POCBR). Figure 5.5 shows the retrieval process, which makes use of two frameworks, i.e., a POCBR framework (e.g., ProCAKE [9]), and a DL framework (e.g., TensorFlow [2]). These two frameworks interact while training the neural network in an offline phase (see Section 5.5.1) and predicting pairwise graph similarities in a retrieval with the neural network. A MAC/FAC retrieval, additionally, performs a subsequent similarity computation with a knowledge-intensive similarity measure (see Section 5.5.2).

5.5.1 Offline Training

The offline training phase has the goal of making the neural network learn how to predict similarities between graphs. The phase is initiated by the POCBR framework that encodes the training data and exports them to be used for training. The training data are composed of a case base of semantic graphs, encoded according to the procedure in Section 5.4.1, and the ground-truth pairwise similarities of these cases. The encoded graphs act as the input data to the neural network and the ground-truth similarities are used as training targets. These target values can be taken from various sources, e.g., computed by the graph similarity measure (see Section 5.2.2) as in previous work [26, 27], or determined by human expert experience. The generated training data are further exported and made available for the training session in the DL framework. The DL framework imports the training data, trains a neural network model (sGEM or sGMN), and exports the trained model. The exported model can be used to resume training at a later point in time, which is helpful to reduce training time in scenarios where dynamic case bases are used (e.g., [54, 57]). There are several factors to consider for the training session, e.g., training time and network configuration. Depending on the use case, these settings influence the training success and, eventually, the retrieval quality. Consequently, hyperparameter optimization should be performed for the training process with each case domain. When preparing the trained model to be used in the POCBR framework, it is possible to cache embeddings for all cases of the case base. However, this is only possible with the sGEM since the sGMN has to compute the cross-graph matching component for every pair of the query and the cases (see Section 5.3.3).

5.5.2 Retrieval

The data that are utilized during retrieval comprise the case base of NEST graphs, the query NEST graph, and the trained neural network. All cases as well as the query have to be encoded in order to be used with the neural network. Given the encoded graphs, the DL framework is used to predict the similarity of each pair of the query and the cases from the case base. The cached graph representation of each case can be utilized to calculate these similarities if applicable. In this way, only the query graph has to be embedded by the neural network. Hence, the result of the prediction is a list of pairwise similarities that can further be used to determine the nearest neighbors of the query in the case base. A standard retrieval is finished after the k-nearest neighbors are determined. However, sGEM and sGMN are also applicable in a MAC/FAC retrieval (see Section 5.2.3). The main difference between a standard retrieval and a MAC/FAC retrieval is the use of two consecutive retrieval phases (MAC and FAC phase), where different

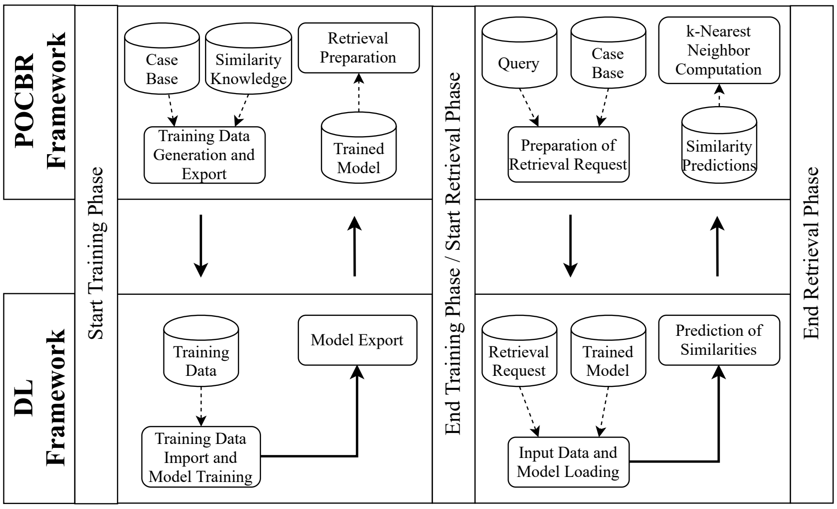

POCBR Framework
DL Framework
Start Training Phase
End Training Phase / Start Retrieval Phase
End Retrieval Phase
Case Base
Similarity Knowledge
Retrieval Preparation
Training Data Generation and Export
Trained Model
Query
Case Base
k-Nearest Neighbor Computation
Preparation of Retrieval Request
Similarity Predictions
Training Data
Model Export
Training Data Import and Model Training
Retrieval Request
Trained Model
Prediction of Similarities
Input Data and Model Loading

Figure 5.5: Retrieval by using a neural-network-based similarity measure.

similarity measures are used. Neural-network-based similarity measures such as the sGEM and the sGMN can be used in both phases, as other approaches demonstrate [27, 31]. However, applying a neural network in the MAC phase is usually more reasonable, since the predicted similarities are only an approximation of the true similarities. This can lead to problems with the candidate set, which, on the one hand, might contain cases whose true similarity is lower than the *fs*-threshold or, on the other hand, might not contain cases whose true similarity is higher than the *fs*-threshold. The cases from the possibly wrong set of candidates are then used to compute the final similarity values according to the query. The FAC similarity measure is usually one that is guaranteed to compute the true similarities, e.g., our A* graph matching algorithm (see Section 5.2.2), since the final similarities directly specify the k-most similar cases, which are returned to the user.

5.6 Experimental Evaluation

We evaluate the sGEM and the sGMN by applying them as similarity measures in standard retrieval and MAC/FAC retrieval scenarios. Thereby, we compare different variants of the embedding models and different retrieval scenarios. The variants are marked with subscripts in our terminology, with a total of six different variants:

sGEM sGEM used with sequence encoding;

sGEM$_{\text{tree}}$ sGEM used with tree encoding;

sGMN sGMN used with sequence encoding;

sGMN$_{\text{tree}}$ sGMN used with tree encoding;

sGMN$_{\text{const}}$ sGMN used with matching constraints;

sGMN$_{\text{tree,const}}$ sGMN used with tree encoding and matching constraints.

As ground-truth similarities for training the neural networks and comparing their predictions, we use similarities computed by the A*-similarity measure by Bergmann and Gil [8] (**A*M**; see Section 5.2.2). The ground-truth similarities are computed with a configuration of the measure that uses a very high parameter value for MaxPMS, to allow a good solution to be found in the mapping process. Additionally, we also use this measure with a lower value of MaxPMS in the evaluation to measure the deviations from the ground-truth similarities. This allows us to see how much the computed similarities of A*M deviate from the ground-truth similarities if MaxPMS is reduced. Another evaluated measure is the feature-based measure by Bergmann and Stromer [11] (**FBM**), which uses a manually modeled feature representation of NEST graphs, specifically designed to be used in combination with a lightweight similarity measure in MAC/-FAC retrieval situations. It is included in order to achieve a comparison between automatically learned measures and manually modeled ones, but comparison with this measure is not the main focus. Furthermore, the embedding-based measure of Klein, Malburg, and Bergmann [31] (**EBM**) is evaluated. It is also designed for MAC/FAC retrieval and utilizes learned graph embeddings and a vector-based similarity (see Section 5.2.4 for more details). EBM will be the main measure to compare sGEM and sGMN against since it is also automatically learned and embedding-based. We investigate the following hypotheses in two experiments:

Hypothesis 1 (H1) *Using the sGEM variants as MAC similarity measures of a MAC/FAC retrieval leads to improved retrieval results compared to using EBM as a MAC similarity measure;*

Hypothesis 2 (H2) *The sGMN variants can approximate the ground-truth graph similarities better than A*M, considering a reduced MaxPMS value such that the retrieval time of both retrievers is comparable.*

Each hypothesis refers to a dedicated experiment. The first experiment examines the measures in a MAC/FAC setup, where the focus is placed on the suitability of sGEM as a MAC similarity measure. We focus on evaluating sGEM against the other automatically learned similarity measure, EBM, in H1 as it is, by design, more optimized for performance, which fits the requirements of a MAC similarity measure (see Section 5.5.2). The second experiment examines the degree to which sGMN is capable of approximating the ground-truth A* similarities. As introduced before, A*M can be configured (by adjusting the MaxPMS parameter) to compute solutions faster with lower quality in return. H2 aims at comparing sGMN to a configuration of A*M with a reduced value of MaxPMS where the deviations from the ground-truth similarities are comparable.

5.6.1 Experimental Setup

We perform our experiments with an implementation of the presented approach in the POCBR framework *ProCAKE* [9] and the DL framework *TensorFlow* [2]. The source code and all supplementary data can be retrieved with the instructions given in the *Data Availability Statement* on page 81. The neural networks are trained with the help of TensorFlow and the similarity assessment is performed in ProCAKE. Our experiments examine two case bases from different domains that are both represented as semantic NEST graphs. The cooking processes (*CB-I*) contain 800 sandwich recipes with ingredients and cooking steps [47], split into 660 training cases, 60 validation cases, and 80 test cases. The processes of the data mining domain (*CB-II*) are built from sample processes that are delivered with RapidMiner (see [57] for more details), split into 509 training cases, 40 validation cases, and 60 test cases. A single training instance is a pair of graphs with the associated similarity value to learn, leading to $660^2 = 435{,}600$ and $509^2 = 259{,}081$ training instances, respectively. Thereby, the training case base is used as training input for the neural networks, while the validation case base is used to monitor the training process and to optimize hyperparameter values. Hyperparameter tuning is performed individually per domain with the two base models of sGEM and sGMN. The model variants then use the same hyperparameter settings as the associated base model, e.g., sGEM$_{\text{tree}}$ uses the same hyperparameter configuration as sGEM in the same domain. The training of all models is stopped as soon as the validation loss does not further decrease for two epochs (early stopping).

The metrics that are used to evaluate our approach cover performance and quality. The performance is measured by taking the retrieval time, which is the entire retrieval time including pre-processing of the data if necessary. Since all variants of sGEM and EBM allow for caching of embedding vectors for the case base in an offline phase, these measures only embed the query during the experiments and do not include the caching time in the results. The quality of the results to evaluate RL_{eval} is measured by comparing them to the ground-truth retrieval results RL_{true} in terms of MAE, correctness (see [15, 27] for more details), and k-NN quality (see [27, 31, 48] for more details). The MAE (ranging between 0 and 1) expresses the average similarity error between all pairs of query graph and case graph in RL_{true} and the same pairs in RL_{eval}. The correctness (ranging between -1 and 1) describes the conformity of the ranking positions of the graph pairs in RL_{eval} according to RL_{true}. Given two arbitrary graph pairs $GP_1 = (QG, CG_1)$ and $GP_2 = (QG, CG_2)$, the correctness is decreased if GP_1 is ranked before GP_2 in RL_{eval} although GP_2 is ranked before GP_1 in RL_{true} or vice versa. The k-NN quality (ranging between 0 and 1; see Equation (5.16)) quantifies the degree to which highly similar cases according to

RL_{true} are present in RL_{eval}.

$$quality(QG, RL_{\text{true}}, RL_{\text{eval}}) = 1 - \frac{1}{|\, RL_{\text{true}} \,|} \cdot \sum_{CG \in RL_{\text{true}} \setminus RL_{\text{eval}}} sim(QG, CG) \tag{5.16}$$

Therefore, the cases from RL_{true} are compared with RL_{eval}. Each case from RL_{true} that is missing in RL_{eval} decreases the quality, with highly relevant cases affecting the quality more strongly than less relevant cases.

The machine that is used for training and testing computations is a PC with an Intel i7 6700 CPU (4 cores, 8 threads), an NVIDIA GTX 3070 GPU, and 48 GB of RAM, running Windows 10. The measures (EBM, all sGEM variants, and all sGMN variants) that require an offline training phase are trained on the GPU with the two training case bases, resulting in two models per measure, i.e., one for each domain. The complete list of training and model parameters can be found in the published source code. To summarize, the models are parameterized such that the GMN models have larger embedding sizes (nodes, edges, graph) than the GEM models and the models for CB-II have larger embedding sizes than the models for CB-I. The average training time is approx. 24 h for the sGMN variants and 8 h for the sGEM variants on average. The inference of all measures is performed only by using the CPU in order to allow a fair comparison. A retrieval is always conducted with a query from the testing case base and with the cases from the training case base. To produce meaningful performance and quality values, the results of the retrieval runs of all query cases from a single domain are averaged.

5.6.2 Experimental Results

The first experiment aims to answer hypothesis H1 and evaluates the variants of sGEM and sGMN as MAC similarity measures in a scenario of MAC/FAC retrieval. Different combinations of *fs*, i.e., the number of candidates cases of the MAC phase, and k, i.e., the number of retrieval results from the FAC phase, are examined. These values are chosen to be similar to the experiments in previous work [31]. The FAC similarity measure is an A* measure that is used in the same configuration in every retrieval. Table 5.1 shows the evaluation results, which include the metrics k-NN quality and retrieval time (in milliseconds). Besides the variants of the neural network models, we also evaluate FBM and EBM since these two measures are specifically designed for MAC/FAC applications. The highlighted values represent the maximum quality and minimum time, respectively, for each combination of *fs* and k.

Table 5.1: Evaluation results of the MAC/FAC experiment.

			sGEM		**sGEM$_{\text{tree}}$**		**sGMN**		**sGMN$_{\text{tree}}$**		**sGMN$_{\text{const}}$**		**sGMN$_{\text{tree,const}}$**		**FBM**		**EBM**	
	fs	**k**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
CB-I	5	5	0.508	16	0.511	**14**	0.489	1051	0.490	2067	0.489	1074	0.489	2169	**0.557**	510	0.499	17
	50	5	0.613	100	0.600	**95**	0.522	1137	0.523	2156	0.511	1165	0.505	2263	**0.836**	611	0.562	100
	10	10	0.520	24	0.536	**22**	0.502	1061	0.503	2077	0.500	1085	0.505	2180	**0.585**	522	0.516	25
	80	10	0.623	155	0.609	**151**	0.581	1195	0.576	2225	0.548	1224	0.575	2330	**0.862**	671	0.613	158
	25	25	0.545	50	0.567	**48**	0.534	1088	0.536	2106	0.518	1113	0.534	2208	**0.652**	554	0.549	53
	100	25	0.606	192	0.593	**187**	0.646	1240	0.678	2268	0.607	1266	0.676	2373	**0.833**	714	0.642	195
CB-II	5	5	0.392	66	0.394	84	0.381	2811	0.449	3539	0.399	2729	0.463	3666	**0.646**	457	0.329	**57**
	50	5	0.557	866	0.563	895	0.629	3150	0.757	3833	0.671	3016	0.767	3928	**0.922**	619	0.385	**295**
	10	10	0.432	113	0.448	123	0.467	2825	0.516	3572	0.457	2785	0.508	3690	**0.667**	477	0.362	**84**
	80	10	0.625	1146	0.637	1158	0.745	3406	0.837	4064	0.749	3372	0.853	4197	**0.939**	744	0.444	**430**
	25	25	0.506	281	0.511	628	0.550	3004	0.623	3683	0.547	2856	0.628	3793	**0.694**	533	0.416	**198**
	100	25	0.683	1371	0.697	1303	0.799	3597	0.872	4234	0.797	3537	0.881	4320	**0.907**	866	0.500	**518**

The results for CB-I show that the two variants of sGEM perform similarly, with similar quality values and retrieval times. The same applies to the quality values of the sGMN variants.

The retrieval times of the sGMN variants reveal a negative influence of the variants that use tree encoding, which is likely caused by the more complex embedding process. The sGMN variants and the sGEM variants perform similarly in terms of quality, despite the more expressive embedding process of sGMN. EBM shows quality values that are on par with the values of the sGEM variants. FBM shows the highest quality values across all combinations of *fs* and k. The retrieval times are higher than those of the sGEM variants and EBM but lower than those of the sGMN variants. The results for CB-II present a similar picture. FBM still outperforms all other measures with regard to quality. However, some sGMN variants, such as $\text{sGMN}_{\text{tree,const}}$, are performing in a similar range. It is also important to note that the sGMN variants with tree encoding are still the slowest measures but outperform all other automatically learned measures for CB-II. The sGMN variants also outperform the sGEM variants for almost all combinations of *fs* and k, which shows that the variants of sGMN favor retrieval situations with more complex semantic descriptions of task and data nodes, as present in CB-II. The performance of the sGEM and sGMN measures for retrieving graphs from a rather simple domain, such as in CB-I, is respectable but does not consistently surpass the currently available, automatically learned approach of EBM. When only looking at the automatically learned measures in the results of CB-II, i.e., the variants of sGEM, the variants of sGMN, and EBM, sGEM variants are the most suitable for a MAC/FAC scenario since they show a good combination of very low retrieval times and high quality values. The FBM, with its manually modeled similarity measure, still performs best for both case bases, taking into account the combination of quality and time. However, under conditions of strict time requirements, sGEM or EBM can become better suited than FBM. Due to their speed, the measures allow for the performance of a MAC phase with high values of *fs*, which, in turn, produce better overall quality results for the MAC/FAC retrieval. For instance, sGEM can filter with $fs = 80$ in less time and with higher ultimate quality than FBM can filter with $fs = 10$ to obtain the results for $k = 10$. Overall, H1 is partly confirmed due to the similar results of the comparison between sGEM and EBM. None of the different measures clearly outperforms the other. However, sGEM can improve the overall retrieval results under certain conditions (*fs*, k, variant, etc.), which have to be tested for the given scenario.

The second experiment aims to answer hypothesis H2 and examines the degree to which the variants of sGEM and sGMN as well as EBM and FBM can approximate the ground-truth graph similarities (see Table 5.2). We also include a configuration of A*M with an adjusted parameter value of MaxPMS (see Section 5.2.2) in the experiment. The value for MaxPMS is chosen in such a way that the retrieval time of A*M is similar to that of the sGMN variants. Aligning the retrieval times of A*M and sGMN enables a fair comparison of the resulting MAE and correctness. All reported times are measured in milliseconds.

For CB-I, $\text{sGMN}_{\text{const}}$ has the lowest MAE value and A*M has the highest correctness value. The sGEM variants report relatively high MAE values and relatively low values of correctness. The MAE and the correctness values of all sGMN variants are in a similar range, while all outperform the MAE of A*M but are outperformed by A*M in terms of correctness. FBM achieves a high level of correctness but lags in terms of MAE. EBM shows high values of MAE and low values of correctness. When comparing the results of CB-I and CB-II, it becomes apparent that $\text{sGMN}_{\text{tree,const}}$ has the lowest MAE and A*M again has the highest value of correctness. The results of the other measures are similar to the results for CB-I. However, it is noticeable that the gap between the sGMN variants and A*M in terms of correctness is smaller. This leads to the assumption that the suitability of sGMN increases with more complex cases. The fact that the variants of sGMN outperform A*M in terms of MAE is even more remarkable when considering that the neural network learns to assess the similarity of graphs without knowing the original algorithmic context, e.g., similarities of semantic descriptions or node and edge mappings. Additionally, this experiment shows that FBM and EBM are not

Table 5.2: Evaluation results of the A* approximation experiment.

Domain		CB-I			CB-II	
Retriever	**MAE**	**Correctness**	**Time**	**MAE**	**Correctness**	**Time**
sGEM	0.158	0.017	1.3	0.337	0.331	1.1
$\text{sGEM}_{\text{tree}}$	0.219	0.053	**0.9**	0.323	0.310	**0.9**
sGMN	0.033	0.287	1025.5	0.034	0.583	2697.8
$\text{sGMN}_{\text{tree}}$	0.039	0.322	2115.1	0.026	0.724	3508.1
$\text{sGMN}_{\text{const}}$	**0.029**	0.330	1051.9	0.033	0.583	2643.7
$\text{sGMN}_{\text{tree,const}}$	0.037	0.327	2118.3	**0.024**	0.732	3403.8
FBM	0.193	0.598	482.7	0.199	0.584	335.4
EBM	0.380	0.224	1.2	0.397	0.006	1.1
A*M	0.062	**0.669**	1265.5	0.041	**0.824**	3801.8

suitable for generating similarities that are close to the ground-truth similarities. The reason for this could be the inadequate processing of semantic annotations and the workflow structure. Thus, we partly accept H2 since the sGMN variants consistently outperform the MAE values of A*M but do not report higher values of correctness.

5.6.3 Discussion

The two experiments show the suitability of sGEM and sGMN and their different variants in similarity-based retrieval. We would like to discuss some points in particular. First, sGEM and sGMN present inconsistency in the effects that different variants have on different domains. Compared to the base models, these effects can lead to completely different results in different domains. For instance, in the second experiment, $\text{sGEM}_{\text{tree}}$ shows a lower MAE than the base model for CB-II but a significantly higher MAE for CB-I. This leads to the need for individual testing of different methods and subsequent hyperparameter tuning. Since the hyperparameters of the models in our experiments are only tuned for the base models, this can certainly lead to further performance improvements. The performance of the sGEM and sGMN variants can further be improved by performing inference on the GPU instead of the CPU (as done in these experiments). Since computations on the GPU are not possible for all other measures, this is a unique characteristic of the neural-network-based models. Furthermore, it is shown that the additional integration of the semantic information of sGEM and sGMN and the usage of GNNs can outperform the simple, structural measure, EBM. This confirms the assumption that partly motivated this paper. The effect might be amplified by domains that are even more complex than CB-II, such as argumentation [36] or flexible manufacturing [41]. Additionally, it is shown that sGEM and sGMN, although integrating rich semantic information, are not able to consistently outperform FBM in MAC/FAC retrieval tasks. The usage of manually modeled features and the integration of expert knowledge seems to be superior to automatic learning. Nevertheless, automatically learned measures such as sGEM, sGMN, and EBM still have the advantage of greatly reduced effort when adapting to changes in the similarity definition or the underlying domain models. The overall effort of knowledge acquisition is greatly reduced when using an automatically learned measure. Automatic learning is also helpful when dealing with user-labeled data, since users are usually not capable of labeling cases with concrete similarity values but rather with binary indications, e.g., similar or not similar. The neural networks can learn similarity functions based on these labels with a modified loss function [37, 42]. In contrast, this is not straightforward for A*M and FBM as it involves manual configuration effort.

5.7 Conclusions and Future Work

This paper examines the potential of using two Siamese Graph Neural Networks (GNNs) as similarity measures for retrieving semantic graphs in POCBR. The two presented neural networks, i.e., the sGEM and the sGMN, can be used in a similarity-based retrieval by predicting the similarities of graph pairs. Therefore, a novel encoding scheme is presented that covers the graph structure, the types of nodes and edges, and their semantic annotations. Given this encoding scheme, sGEM and sGMN are constructed by adapting two neural networks from the literature to fully process pairs of semantic graphs to predict a pairwise similarity value. Setting up the retrieval process with these neural networks in a POCBR framework and a DL framework is discussed as well. The experimental evaluation investigates how differently configured variants of sGEM and sGMN perform in similarity-based retrieval. The evaluation covers two domains with different properties, nine different similarity measures, and two different retrieval approaches. The results show the suitability of sGEM and sGMN in the evaluated scenarios. Thereby, sGEM is suitable for a MAC/FAC setup, due to its fast similarity computation and reasonable retrieval quality. Furthermore, sGMN shows great potential in approximating the ground-truth graph similarities.

A focus of future research should be on optimizing the presented approach of a GNN-based retrieval. This optimization ranges from aspects of parameterization to adjustments of the data encoding scheme and the usage of different neural network structures. The neural network structures could be optimized to better process other graph domains, e.g., argument graphs [36], or even other types of complex similarity measures [46]. A structural change could be, for instance, using a differentiable ranking loss function that optimizes according to the ground-truth ordering of the retrieval results (e.g., [40]). Furthermore, the approaches from this paper should be applied to other POCBR tasks, such as case adaptation (e.g., [34]). Since adaptation is usually a knowledge-intensive process, our approaches can help by providing expressive learning capabilities combined with possibilities for domain knowledge integration. Additionally, the neural networks that are used in this work are black boxes and, thus, are not capable of explaining the results they produce. In current research (and also in the CBR community, e.g., [28]), this lack of explainability is tackled in the context of *Explainable Artificial Intelligence (XAI)*. Future research should address this issue by investigating which methods are suitable for increasing the explainability of the presented neural networks.

Author Contributions

Conceptualization, M.H. and R.B.; methodology, M.H.; software, M.H.; validation, M.H.; formal analysis, R.B.; resources, M.H., R.B.; data curation, M.H.; writing—original draft preparation, M.H.; writing—review and editing, M.H. and R.B.; visualization, M.H.; supervision, R.B. All authors have read and agreed to the published version of the manuscript.

Funding

The publication was funded by the Open Access Fund of the University of Trier and the German Research Foundation (DFG) within the Open Access Publishing funding programme.

Data Availability Statement

The source code, the training data, the training parameters, the trained models, and all other supplementary resources are available online at https://gitlab.rlp.net/procake-embedding/procake-embedding-experiments. To set up the workspace and repeat the experiments, follow the instructions in the corresponding *ReadMe file*. Please use the code version with the tag name "MDPI_Algorithms_Graph_Embedding_Applications". The data was last accessed on 13 December 2021.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994). https://doi.org/10.3233/AIC-1994-7104
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P.A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning. In: Keeton, K., Roscoe, T. (eds.) 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016, pp. 265–283. USENIX Association (2016)
3. Amin, K., Lancaster, G., Kapetanakis, S., Althoff, K., Dengel, A., Petridis, M.: Advanced Similarity Measures Using Word Embeddings and Siamese Networks in CBR. In: Bi, Y., Bhatia, R., Kapoor, S. (eds.) Intelligent Systems and Applications - Proceedings of the 2019 Intelligent Systems Conference, IntelliSys 2019, London, UK, September 5-6, 2019, Volume 2. Advances in Intelligent Systems and Computing, pp. 449–462. Springer (2019). https://doi.org/10.1007/978-3-030-29513-4_32
4. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
5. Baldi, P., Chauvin, Y.: Neural Networks for Fingerprint Recognition. Neural Comput. **5**(3), 402–418 (1993). https://doi.org/10.1162/NECO.1993.5.3.402
6. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V.F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H.F., Ballard, A.J., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K.R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M.M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. CoRR **abs/1806.01261** (2018). arXiv: 1806.01261
7. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Springer (2002)
8. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014). https://doi.org/10.1016/J.IS.2012.07.005
9. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Kapetanakis, S., Borck, H. (eds.) Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Reasoning co-located with the Twenty-seventh International Conference on Case-Based Reasoning (ICCBR 2019), Otzenhausen, Germany, September 8-12, 2019. CEUR Workshop Proceedings, pp. 156–161. CEUR-WS.org (2019)

10. Bergmann, R., Müller, G.: Similarity-Based Retrieval and Automatic Adaptation of Semantic Workflows. In: Synergies Between Knowledge Engineering and Software Engineering. Ed. by G.J. Nalepa and J. Baumeister, pp. 31–54. Springer (2018). https://doi.org/10.1007/978-3-319-64161-4_2

11. Bergmann, R., Stromer, A.: MAC/FAC Retrieval of Semantic Workflows. In: Boonthum-Denecke, C., Youngblood, G.M. (eds.) Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida, USA, May 22-24, 2013. AAAI Press (2013)

12. Bromley, J., Bentz, J.W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E., Shah, R.: Signature Verification Using A "Siamese" Time Delay Neural Network. Int. J. Pattern Recognit. Artif. Intell. **7**(4), 669–688 (1993). https://doi.org/10.1142/S0218001493000339

13. Bunke, H.: Recent Developments in Graph Matching. In: 15th International Conference on Pattern Recognition, ICPR'00, Barcelona, Spain, September 3-8, 2000, pp. 2117–2124. IEEE Computer Society (2000). https://doi.org/10.1109/ICPR.2000.906030

14. Cai, H., Zheng, V.W., Chang, K.C.: A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. CoRR **abs/1709.07604** (2017). arXiv: 1709.07604

15. Cheng, W., Rademaker, M., de Baets, B., Hüllermeier, E.: Predicting Partial Orders: Ranking with Abstention. In: Cellular automata. Ed. by S. Bandini, S. Manzoni, H. Umeo, and G. Vizzari, pp. 215–230. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-15880-3_20

16. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 1724–1734. ACL (2014). https://doi.org/10.3115/V1/D14-1179

17. Corchado, J.M., Lees, B.: Adaptation of Cases for Case Based Forecasting with Neural Network Support. In: Soft Computing in Case Based Reasoning. Ed. by S.K. Pal, T.S. Dillon, and D.S. Yeung, pp. 293–319. Springer (2001). https://doi.org/10.1007/978-1-4471-0687-6_13

18. Dieterle, S., Bergmann, R.: A Hybrid CBR-ANN Approach to the Appraisal of Internet Domain Names. In: Lamontagne, L., Plaza, E. (eds.) Case-Based Reasoning Research and Development - 22nd International Conference, ICCBR 2014, Cork, Ireland, September 29, 2014 - October 1, 2014. Proceedings. LNCS, vol. 8765, pp. 95–109. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-11209-1_8

19. Forbus, K.D., Gentner, D., Law, K.: MAC/FAC: A Model of Similarity-Based Retrieval. Cogn. Sci. **19**(2), 141–205 (1995). https://doi.org/10.1207/S15516709COG1902_1

20. Gabel, T., Godehardt, E.: Top-Down Induction of Similarity Measures Using Similarity Clouds. In: Hüllermeier, E., Minor, M. (eds.) Case-Based Reasoning Research and Development - 23rd International Conference, ICCBR 2015, Frankfurt am Main, Germany, September 28-30, 2015, Proceedings. LNCS, vol. 9343, pp. 149–164. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-24586-7_11

21. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural Message Passing for Quantum Chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, pp. 1263–1272. PMLR (2017)

22. Goyal, P., Ferrara, E.: Graph Embedding Techniques, Applications, and Performance: A Survey. CoRR **abs/1705.02801** (2017). arXiv: 1705.02801

23. Grumbach, L., Bergmann, R.: Using Constraint Satisfaction Problem Solving to Enable Workflow Flexibility by Deviation (Best Technical Paper). In: Bramer, M., Petridis, M. (eds.) Artificial Intelligence XXXIV - 37th SGAI International Conference on Artificial Intelligence, AI 2017, Cambridge, UK, December 12-14, 2017, Proceedings. LNCS, vol. 10630, pp. 3–17. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-71078-5_1

24. Hanney, K., Keane, M.T.: The adaptation knowledge bottleneck: How to ease it by learning from cases. In: Case-Based Reasoning Research and Development. Ed. by D.B. Leake and E. Plaza, pp. 359–370. Springer, Berlin and Heidelberg (1997). https://doi.org/10.1007/3-540-63233-6_506

25. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/NECO.1997.9.8.1735

26. Hoffmann, M., Bergmann, R.: Informed Machine Learning for Improved Similarity Assessment in Process-Oriented Case-Based Reasoning. CoRR **abs/2106.15931** (2021). arXiv: 2106.15931

27. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

28. Keane, M.T., Kenny, E.M.: How Case-Based Reasoning Explains Neural Networks: A Theoretical Analysis of XAI Using Post-Hoc Explanation-by-Example from a Survey of ANN-CBR Twin-Systems. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 155–171. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_11

29. Kendall-Morwick, J., Leake, D.: A Study of Two-Phase Retrieval for Process-Oriented Case-Based Reasoning. In: Successful Case-based Reasoning Applications-2, pp. 7–27. Springer (2014)

30. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

31. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 188–203. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_13

32. Leake, D., Crandall, D.J.: On Bringing Case-Based Reasoning Methodology to Deep Learning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 343–348. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_22

33. Leake, D., Ye, X.: Harmonizing Case Retrieval and Adaptation with Alternating Optimization. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) Case-Based Reasoning Research and Development - 29th International Conference, ICCBR 2021, Salamanca, Spain, September 13-16, 2021, Proceedings. LNCS, vol. 12877, pp. 125–139. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-86957-1_9

34. Leake, D., Ye, X., Crandall, D.J.: Supporting Case-Based Reasoning with Neural Networks: An Illustration for Case Adaptation. In: Martin, A., Hinkelmann, K., Fill, H., Gerber, A., Lenat, D., Stolle, R., van Harmelen, F. (eds.) Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021. CEUR Workshop Proceedings. CEUR-WS.org (2021)

35. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539

36. Lenz, M., Ollinger, S., Sahitaj, P., Bergmann, R.: Semantic Textual Similarity Measures for Case-Based Retrieval of Argument Graphs. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 219–234. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_15

37. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, pp. 3835–3845. PMLR (2019)

38. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.S.: Gated Graph Sequence Neural Networks. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016)

39. Liao, C., Liu, A., Chao, Y.: A Machine Learning Approach to Case Adaptation. In: First IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018, Laguna Hills, CA, USA, September 26-28, 2018, pp. 106–109. IEEE Computer Society (2018). https://doi.org/10.1109/AIKE.2018.00023

40. Liu, T.-Y.: Learning to Rank for Information Retrieval. Springer, Berlin and Heidelberg (2011)

41. Malburg, L., Seiger, R., Bergmann, R., Weber, B.: Using Physical Factory Simulation Models for Business Process Management Research. In: del-Río-Ortega, A., Leopold, H., Santoro, F.M. (eds.) Business Process Management Workshops - BPM 2020 International Workshops, Seville, Spain, September 13-18, 2020, Revised Selected Papers. LNBIP, vol. 397, pp. 95–107. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-66498-5_8

42. Mathisen, B.M., Aamodt, A., Bach, K., Langseth, H.: Learning similarity measures from data. Prog. Artif. Intell. **9**(2), 129–143 (2020). https://doi.org/10.1007/S13748-019-00201-2

43. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. **51**(11), 8107–8118 (2021). https://doi.org/10.1007/S10489-021-02251-3

44. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. In: Bengio, Y., LeCun, Y. (eds.) 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013)

45. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014). https://doi.org/10.1016/J.IS.2013.06.004

46. Mougouie, B., Bergmann, R.: Similarity Assessment for Generalized Cases by Optimization Methods. In: Advances in Case-Based Reasoning. Ed. by S. Craw and A. Preece, pp. 249–263. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-46119-1_19

47. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer, Wiesbaden (2018)

48. Müller, G., Bergmann, R.: A Cluster-Based Approach to Improve Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, pp. 639–644. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-639

49. Ontañón, S.: An Overview of Distance and Similarity Functions for Structured Data. CoRR **abs/2002.07420** (2020). arXiv: 2002.07420

50. Richter, M.M.: Foundations of Similarity and Utility. In: Wilson, D., Sutcliffe, G. (eds.) Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7-9, 2007, Key West, Florida, USA, pp. 30–37. AAAI Press (2007)

51. Richter, M.M., Weber, R.O.: Case-Based Reasoning - A Textbook. Springer (2013)

52. von Rüden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Pfrommer, J., Pick, A., Ramamurthy, R., Walczak, M., Garcke, J., Bauckhage, C., Schuecker, J.: Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. IEEE Trans. Knowl. Data Eng. **35**(1), 614–633 (2023). https://doi.org/10.1109/TKDE.2021.3079836

53. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The Graph Neural Network Model. IEEE Trans. Neural Networks **20**(1), 61–80 (2009). https://doi.org/10.1109/TNN.2008.2005605

54. Stram, R., Reuss, P., Althoff, K.: Dynamic Case Bases and the Asymmetrical Weighted One-Mode Projection. In: Cox, M.T., Funk, P., Begum, S. (eds.) Case-Based Reasoning Research and Development - 26th International Conference, ICCBR 2018, Stockholm, Sweden, July 9-12, 2018, Proceedings. LNCS, vol. 11156, pp. 385–398. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01081-2_26

55. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)

56. Zeyen, C., Bergmann, R.: A*-Based Similarity Assessment of Semantic Graphs. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 17–32. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_2

57. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 388–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_26

Chapter 6

Augmentation of Semantic Processes for Deep Learning Applications

Bibliographic Information

Hoffmann, M., Malburg, L., Bergmann, R.: Augmentation of Semantic Processes for Deep Learning Applications. Applied Artificial Intelligence. Taylor and Francis. Submitted for Publication. (2024)

Abstract

The popularity of *Deep Learning (DL)* methods used in business process management research and practice is constantly increasing. One important factor that hinders the adoption of DL in certain areas is the availability of sufficiently large training datasets, particularly affecting domains where process models are mainly defined manually with a high knowledge-acquisition effort. In this paper, we examine *process model augmentation* in combination with *semi-supervised transfer learning* to enlarge existing datasets and train DL models effectively. The use case of similarity learning between manufacturing process models is discussed. Based on a literature study of existing augmentation techniques, a concept is presented with different categories of augmentation from knowledge-light approaches to knowledge-intensive ones, e. g., based on automated planning. Specifically, the impacts of augmentation approaches on the syntactic and semantic correctness of the augmented process models are considered. The concept also proposes a semi-supervised transfer learning approach to integrate augmented and non-augmented process model datasets in a two-phased training procedure. The experimental evaluation investigates augmented process model datasets regarding their quality for model training in the context of similarity learning between manufacturing process models. The results indicate a large potential with a reduction of the prediction error of up to 53%.

Keywords

Data Augmentation · Business Process Management · Graph Neural Networks · Similarity Learning · AI Planning

Copyright Notice

6.1 Introduction

Data-driven *Artificial Intelligence (AI)* methods such as *Deep Learning (DL)* have recently gained significant importance in practice and research. One of these research areas for the use of DL techniques is *Business Process Management (BPM)* [e. g., 11, 52, 55, 56]. However, to fully utilize the DL techniques and their generalization capabilities, a rich set of training data with meaningful training examples is needed, which leads to considerable effort for data acquisition [23, 62]. As this effort is not always manageable and sometimes training data is inaccessible, DL methods can be combined with other AI techniques in an approach commonly known as informed machine learning or hybrid AI [59]. There, the disadvantage of not enough or imbalanced data is compensated by symbolic knowledge. A more common technique to mitigate these data issues in practice and research is *data augmentation* [10, 50, 77, 78]. Data augmentation aims to increase the amount of available training data to make the DL models trained on it more accurate and robust. Popular augmentation methods originate in the domains of image processing and *Natural Language Processing (NLP)* [e. g., 6], where elementary transformational augmentations, e. g., the rotation of an image or the replacement of words with synonyms, are widely used. Augmentations can also be based on synthesis, which is usually more complex, e. g., a *Generative Adversarial Network (GAN)* trained on the original training data creates new examples. In BPM, however, there are only a few augmentation techniques available, which are currently very domain specific and are only concerned with event logs [e. g., 29, 30, 34, 68]. The combination of a high demand for DL-based applications in BPM and a largely unexplored field of augmentations for process models, motivates this work.

The goal of this paper is to discuss approaches of process model augmentation as well as an effective training method of DL models with augmented and non-augmented process models. The contributions of the paper are the following: 1) we present a literature study on process augmentation techniques that are currently available to be used in DL contexts, 2) we discuss three categories of augmentation from knowledge-light approaches, i. e., deleting and replacing nodes and edges, to knowledge-intensive ones based on automated planning [41, 47], 3) we present a training procedure based on semi-supervised transfer learning [32, 66, 70] to train DL models effectively with augmented and non-augmented data, 4) we conduct an extensive evaluation in which we determine the suitability of different augmentation methods for improving the training procedure of DL models based on our previous work [26, 62]. Throughout the paper, we examine the task of learning similarities between process models from a manufacturing domain [38, 39, 64] as the main use case.

The paper is structured as follows: In Sect. 6.2, we describe the foundations that consist of the process model representation and the application domain applied in this work, the semantic similarity assessment between process models, and the basics of using graph neural networks for graph embedding. In addition, we present and discuss related augmentation approaches. Based on these foundations and the identified research gaps, a concept is presented to augment process models and use them to train DL models (see Sect. 6.3). To assess the utility of the concept for the addressed problems, an experimental evaluation is conducted by using several training data configurations for *Graph Neural Networks (GNNs)* based on the discussed augmentation techniques (see Sect. 6.4). In this context, we use several well-known metrics to evaluate the quality of the trained models for process model similarity learning. Finally, a conclusion is given, and future work is discussed in Sect. 6.6.

6.2 Foundations and Related Work

In this section, we introduce the foundations of the semantic process model representation and the manufacturing domain with its underlying domain model that serves as an application scenario (see Sect. 6.2.1 and Sect. 6.2.2). Since the focus of the work is using DL models for similarity learning, Sect. 6.2.3 introduces how similarities are computed between workflows and Sect. 6.2.4 follows with GNNs that are trained for this task. Finally, the proposed approach is embedded into a broader literature context with a focus on generic graph augmentation methods (see Sect. 6.2.5) and more specific approaches from BPM literature (see Sect. 6.2.5).

6.2.1 Semantic Workflow Representation and Domain

All process models in this work are represented as semantically annotated directed graphs, referred to as *NEST* graphs [4]. NEST graphs are used as the process model representation since their strength is on modeling semantic information with an underlying domain model, and they are well-integrated in our previous work [37]. To ensure compatibility with other more common process model representation formats such as *Business Process Model and Notation (BPMN)*, we employ a converter from BPMN to NEST and back. The converter is available as part of the ProCAKE framework[1] [5]. All process models used in the experiments are represented in NEST and BPMN, created with this converter. See Sect. 6.4 and the statement on data availability at the end of the article for more information.

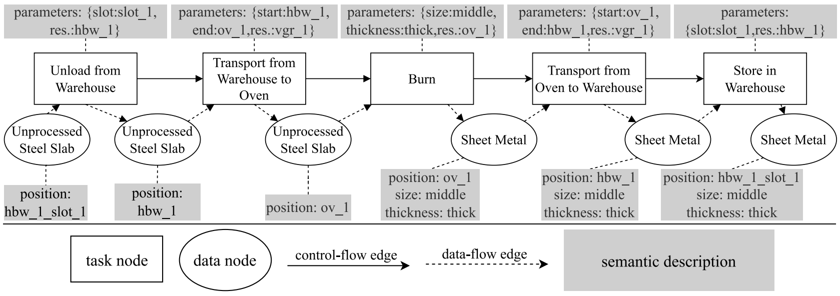

parameters: {slot:slot_1, res.:hbw_1}
parameters: {start:hbw_1, end:ov_1,res.:vgr_1}
parameters: {size:middle, thickness:thick,res.:ov_1}
parameters: {start:ov_1, end:hbw_1,res.:vgr_1}
parameters: {slot:slot_1,res.:hbw_1}
Unload from Warehouse
Transport from Warehouse to Oven
Burn
Transport from Oven to Warehouse
Store in Warehouse
Unprocessed Steel Slab
Unprocessed Steel Slab
Unprocessed Steel Slab
Sheet Metal
Sheet Metal
Sheet Metal
position: hbw_1_slot_1
position: hbw_1
position: ov_1
position: ov_1 size: middle thickness: thick
position: hbw_1 size: middle thickness: thick
position: hbw_1_slot_1 size: middle thickness: thick
task node
data node
control-flow edge
data-flow edge
semantic description

Figure 6.1: A NEST Graph Representing a Sheet Metal Manufacturing Process [Based on: 36].

Each *NEST* graph is a quadruple $G = (N,E,S,T)$, defined by a set of nodes N, a set of edges $E \subseteq N \times N$, a function $T : N \cup E \rightarrow \mathcal{T}$, assigning a type to each node and edge[2], and a function $S : N \cup E \rightarrow \mathcal{S}$, assigning a semantic description from a *semantic metadata language* (e. g., an ontology) to nodes and edges. While nodes and edges build the structure of each graph, types and semantic descriptions are employed to further capture semantic information. As a result, both nodes and edges always have a type and usually also have a semantic description.

The definition of NEST graphs is rather generic to allow their usage in diverse domains, e. g., to represent cooking recipes [49], to solve constraint satisfaction problems [15], to model scientific workflows [76], to represent manufacturing processes [40], or to define argumentation graphs [33]. As the process models addressed in this work are characterized by certain rules on top of the generic NEST graph definition, we introduce the concept of a *sequential NEST process model*:

[1] available at https://gitlab.rlp.net/procake/procake-framework

[2] The defined node types comprise task, data, control-flow, and workflow nodes. The defined edge types comprise control-flow, data-flow, part-of, and constraint edges.

A sequential NEST process model is a NEST graph with a sequential control-flow, a sequential data-flow, and without missing semantic annotations.

By a *sequential control-flow*, we mean that every task node in the process is connected to exactly one preceding task node and one following task node by a control-flow edge. There are two exceptions to this rule, i. e., the start task with only one outgoing control-flow edge and the end task with only one ingoing control-flow edge. The second important extension of NEST graphs towards sequential NEST process models is their *sequential data-flow*. This property is similar to the sequential control-flow but refers to data nodes: Each data node has to be produced by exactly one task and consumed by exactly one task. There is also the exception of two data nodes, i. e., at the beginning of the sequence and at the end, which are consumed or produced only, respectively. A sequential NEST process model with a sequential data-flow prohibits that, for instance, a task consumes or produces multiple data nodes. The third property of sequential NEST process models is about the semantic annotations of nodes and edges that must not be missing. This restriction contributes to the completeness of the process' semantic knowledge and aims to prevent processes with nodes or edges that lack any form of semantic annotation and can only be characterized by their type.

The manufacturing application scenario that we utilize for this work is an example of a domain of sequential NEST process models. The scenario is based on process models that are executed in a smart factory[3] [38, 39, 64]. These processes are considered as *cyber-physical workflows* [43, 63] because they are executed by *actuators* and are driven by *sensors* capturing the production environment. Figure 6.1 illustrates an exemplary manufacturing process as a sequential *NEST* process model, representing the production steps for producing sheet metals. The illustrated process is a typical example of the processes used in this work. *Task nodes* represent concrete activities during production that are executed by actuators, i. e., machines, in the smart factory by a corresponding service. The state of the product produced is captured by the *data nodes*. Each data node represents the current production state of the sheet metal. The semantic descriptions of task and data nodes are used to describe the properties of these nodes. For example, the semantic description of *Transport from Warehouse to Oven* contains relevant parameters for configuring the machine that executes it, e. g., the start and end position of the transport. Similarly, the semantic description of the first *Sheet Metal* data node is composed of the concrete position of the product, i. e., *ov_1*, and its characteristics, e. g., the *size* and *thickness*.

This example shows that the semantic annotations are a key aspect of the information a process holds. In particular, it shows the tight coupling of the *syntactic information* that is given by the representation as a sequential NEST process model, e. g., node and edge types and graph structure, and the *semantic information* that is expressed by the annotations of nodes and edges, e. g., the position of a workpiece. However, a process such as the one shown in Fig. 6.1 that is consistent with all constraints of sequential NEST process models and has well-formed semantic annotations according to the respective semantic metadata language is not guaranteed to be executable in the smart factory, i. e., the manufacturing environment. The reason for this is the inability of the semantic metadata language to describe the behavior of the process tasks in the environment during execution. For example, the task *Burn* in Fig. 6.1 is parameterized by its semantic annotation to produce a middle-sized thick sheet metal. After being burned, the sheet metal is transported to the first *High-Bay Warehouse (HBW)* to be stored there. While the semantic annotations of all the tasks involved are within the definitions of the semantic metadata language, there is still room for errors. It might be possible that the first *Vacuum Gripper Robot (VGR)* is only capable of moving small-sized workpieces, or that the shop floor

[3] More information on the used smart factory can be found at: https://iot.uni-trier.de.

layout could prevent the first VGR from reaching the designated storage buckets in the first HBW. Both problems could be solved by using the second VGR that transports the workpiece to the second HBW.

6.2.2 Domain Model

An additional knowledge representation, called *domain model*, is utilized to check if a sequential NEST process model is executable in its designated manufacturing environment. The domain model also allows interoperability of a process with other systems of the application context, e. g., *Enterprise Resource Planning (ERP)* systems, *Manufacturing Execution Systems (MESs)*, or *Workflow Management Systems (WfMSs)* [64]. The complexity and completeness of the information provided by the domain model can vary greatly and depend on the respective domain. While rather simple domains might have simpler domain models, more complex domains might also have more complex domain models. The model used in this work is given as a comprehensive planning domain description based on the *Planning Domain Definition Language (PDDL)* [47]. The planning domain model consists of the available actions of the smart factory enhanced with semantic information about the preconditions that must be satisfied for execution and the effects that hold after a successful execution. We use a semantic service-based architecture for the smart factory [see 38, 64] and, based on this architecture, a domain expert creates the planning actions contained in the domain description.

Listing 6.1: Planning Action for the Burn Service.

```
1    (:action ov_1_burn
2    :parameters (?resource - oven ?size - sheet_size ?thickness -
         sheet_thickness ?processID - processID)
3    :precondition
4    (and
5    (at ?processID ov_1_pos) (hasPosition ?resource ov_1_pos)
6    (not (isInactive ?resource))
7    (isReady ?resource)
8    (isSteelSlab ?processID) (not (isSheetMetal ?processID)))
9    :effect
10   (and (isSheetMetal ?processID)
11   (isSheetMetalWithSize ?size ?processID)
12   (isSheetMetalWithThickness ?thickness ?processID)
13   (not (isSteelSlab ?processID))
14   (increase (total-cost) (Service_OV_Burn_With_Resource_OV_1
         _With_Size_Parameter_With_Thickness _Parameter_Cost ?size
         ?thickness))))
15   )
```

Listing 6.1 illustrated a planning action created by the domain expert for the *Burn* service. A planning action in PDDL consists of an action name (Line 1), parameters for configuring the action (Line 2), preconditions that need to be satisfied for execution (Lines 3–8), and effects that represent the state transition after successful execution (Lines 9–14). Each task executed by a service in the smart factory has several parameters to be configured (see Sect. 6.2.1). For the illustrated *Burn* task, the executing resource, the size of the burned sheet metal, and the thickness must be specified. In this context, it is important to note that the process ID is a reference to the corresponding process from the smart factory. The preconditions specify the world state in which the action can be executed. For the *Burn* service, the workpiece must be at the location of the oven (Line 5). In addition, the oven must be ready and not inactive (Lines 6–7) as well as the workpiece must be a steel slab, i. e., it must not be burned already (Line 8). After executing the action, the state of the product changes: the workpiece is no steel slab anymore but a sheet metal with a certain size and thickness (Lines 9–13). To capture the costs for executing an action with certain parameters, the total cost function is increased by a predefined value for the service executed (Line 14).

The information in the domain model allows us to check whether a process model can be executed successfully by considering the defined preconditions and effects. It also allows the generation of process models by using *automated planning* [12, 17], which is applied for process model augmentation in this paper.

6.2.3 Semantic Workflow Similarity

The combination of syntax and semantics of sequential NEST process models shows the complexity that is involved with all applications working with this data. One task that arises in these applications is the comparison of process models, e. g., for finding a replacement in case the execution of a different process model fails [36, 37]. To compare different process models, similarities are a common measurement. We also use similarity computation as a demonstrator in the experiments of the paper, as it is widely used in different fields of AI. One example is *Case-Based Reasoning (CBR)*: In the problem-solving methodology of CBR, similarities are widely used to determine which process models are alike and can help to solve emerging problems. Thereby, a repository of process models is queried with a different process model to find the best-matching process models of the repository. This is useful since the found process models might be better suited to solve the problem at hand, e. g., a sudden failure of a machine, than the currently executed process model.

There are various methods for calculating similarities between process models. These methods are sometimes consolidated under the term business process matching [69]. Schoknecht *et al.* [60] categorize different methods based on seven characteristics, e. g., objective and implementation, and, thus, provide a comprehensive overview of the available literature. We use the semantic similarity measure proposed by Bergmann and Gil [4] as it is specifically tailored for NEST graphs and focuses on semantic similarity measures on a node-level, which fits our use case.

To compute the similarity between two *NEST* graphs, the similarity measure evaluates the structure of nodes and edges, as well as the semantic descriptions and types of these components. The final similarity at the graph level is determined by the similarities between the nodes and edges of both graphs [57]. Therefore, a global similarity, denoting the similarity between two graphs, consists of local similarities, i. e., the pairwise similarities of nodes and edges. The similarity between two identical types of nodes is defined as the similarity of the semantic descriptions of these nodes. The similarity value between two edges of the same type includes not only the similarity between the respective semantic descriptions but also the similarity of the connected nodes. The similarities of the semantic descriptions are computed based on the underlying domain model. For instance, in the given exemplary process model illustrated in Fig. 6.1, the similarity between two task nodes would include further similarities between their parameters. This procedure requires the definition of similarity measures for all components of the semantic descriptions as part of the domain's similarity knowledge. These metrics are usually designed with the input of experts in the field and consider the particular data types in use. For example, they could utilize word embeddings [48] for textual data or the *Mean Absolute Error (MAE)* for numerical values [3].

To determine the global similarity between a pair of graphs from the local similarities of the nodes and edges, an injective partial mapping is computed. The objective of this mapping is to maximize the aggregated local similarities between all mapped pairs of corresponding nodes and edges. Nonetheless, as the number of nodes and edges gets larger, the complexity of finding this mapping increases substantially, given that each node and edge can potentially be mapped to multiple counterparts. To address this complexity, Bergmann and Gil [4] propose using an A* search algorithm. A* search is faster than exhaustive search but typically still requires a considerable amount of time and, thus, can result in a slow similarity computation [23, 25, 26, 31, 53, 75].

6.2.4 Graph Embedding

Previous work by Hoffmann and Bergmann [23] tackles the problem of slow similarity computations by using an automatically learned similarity measure based on a Siamese GNN, denoted as the *Graph Embedding Model (GEM)*. The basic idea is to learn to predict the pairwise similarities of workflows with the GEM for faster and sometimes even more accurate similarity computations [23]. Therefore, the GEM transforms the graph structure and the semantic annotations and types of all nodes and edges into a whole-graph latent vector representation. These vectors are then combined to calculate a similarity value. An overview of the used GNN architecture is illustrated in Fig. 6.2.

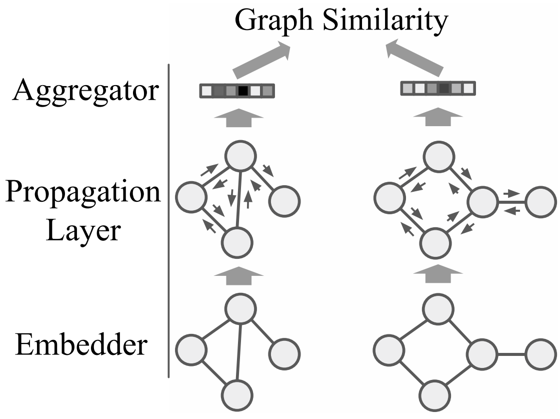


Figure 6.2: The Graph Embedding Model [Source: 26].

The GEM follows a general architecture composed of four parts: First, the *embedder* starts by transforming the node and edge features into initial embeddings within a vector space. These features encompass semantic annotations and types, and they undergo a specialized encoding and processing procedure tailored for semantic graphs [for more details, see 23]. Next, the *propagation layer* collects information for each node from its local surroundings by transmitting messages, as described by Gilmer *et al.* [13]. In detail, the vector representation of a node is continuously updated by incorporating the vectors of neighboring nodes linked through incoming edges. This iterative process is repeated multiple times. Following that, the *aggregator* combines the node embeddings at this stage to generate a vector representation of the entire graph. Eventually, the *similarity* between two graphs is calculated by vector similarity measures, such as cosine similarity, based on the respective graph representations within the vector space.

The trainable parameters of the GEM are modeled as part of the embedder, the propagation layer, and the aggregator. They are trained and adjusted according to a comparison of pairs of predicted similarities and ground-truth similarities by a *Mean Squared Error (MSE)* loss function. After being trained, the neural network can be used as a replacement for the conventional similarity measure based on graph matching (see Sect. 6.2.3).

Overall, the following advantages and disadvantages of both methods are observed:

- The inference runtime of the embedding-based approach is consistent across different runs and only marginally varies based on the size of the input graphs. The runtime of the A*-based graph matching procedure can vary greatly from run to run, due to the integrated search. On the downside, this also means that there are only a few designated mechanisms to configure the runtime budget of the embedding-based approach based on the requirements of different use cases, e. g., general purpose strategies such as integer quantization [72]. The graph matching approach can be precisely configured by restricting the number of solution candidates to consider during search [for more details, see 4].
- The embedding-based approach can benefit greatly from modern hardware accelerators for AI workloads [7]. The graph matching approach is mainly carried out by the CPU and acceleration on, for instance, GPUs requires custom implementations [e. g., 25, 79].
- It is possible to create and cache embedding vectors with the embedding-based approach and use them in their vector form for similarity computations. When caching all embedding vectors of a process model repository in an offline phase, time is saved during the online phase of an application, e. g., for process model similarity assessment. Caching at this level is not possible for the graph-matching-based approach.

6.2.5 Data Augmentation

Another advantage of the embedding-based approach is a result of its self-learning capabilities, and directly motivates this paper: Larger amounts of training data are expected to improve the accuracy of the predictions. Therefore, this paper investigates data augmentation methods [10, 50, 77, 78] to enlarge the available amount of training data and, ultimately, to increase the robustness and quality of the DL models trained on it.

Mumuni and Mumuni [50] distinguish between methods based on *transformation*, where an original training example is transformed into an augmented one, and *synthesis*, where new training examples are generated according to the characteristics of the original training set. Both of these categories are covered for the domain of sequential NEST process models in the proposed approach. The remainder of this section discusses specific augmentation methods for graph data (see Sect. 6.2.5) and for process data within the research field of BPM (see Sect. 6.2.5). While specific augmentation methods for processes are more relevant to this contribution, generic graph augmentation methods are universally applicable and useful as a baseline.

Graph Augmentation

There are several recent surveys on generic graph augmentation that summarize and categorize existing methods:

Ding *et al.* [9] present different methods for graph augmentation based on a taxonomy of two main use cases: 1) reliable graph learning for enhancing model quality and training data utility, and 2) low-resource graph learning for enlarging the labeled training data. For all augmentation methods, node-level, edge-level, and graph-level tasks are considered. The integration of augmentation in an existing training pipeline is also discussed, with one option being a separate augmentation phase before starting the training, and another option being augmentation as part of model training (with trainable augmenters).

Zhao *et al.* [77] structure the presented graph augmentation methods according to three perspectives, i. e., operated data (structure, features, labels), downstream task (node-level, edge-level, graph-level), and whether the augmentations are rule-based or learnable. The methods within these categories are further separated into groups, e. g., methods based on feature removal, addition, and manipulation.

The discussion of augmentation methods by Adjeisah *et al.* [1] is less focused on covering as many different methods as possible but rather on experiments with a selection of popular methods. Therefore, the authors examine eight state-of-the-art methods that augment on either the topology-level or the feature-level and do a mathematical analysis. The methods are also applied to three different widely used GNN architectures to test their effectiveness.

Zhou *et al.* [78] present four taxonomies of graph augmentation methods with a focus for the graph that is augmented, the addressed target task, whether the augmenter is learnable, and the augmentation mechanism. In addition, suitable evaluation metrics for augmentation methods and possible applications are thoroughly discussed.

Marrium and Mahmood [44] categorize augmentation methods based on the addressed task, the augmentation technique, and the learning objective. They also provide benchmarks of several methods and a collection of their implementations. This work has a strict focus on the downstream task that augmentation is used in, e. g., node classification, graph classification, and link prediction.

Yu *et al.* [74] specifically analyze learnable augmentation methods, framed by the term "Graph Augmentation Learning". The strategies are structured as node-, edge-, subgraph-, and graph-level and application scenarios w. r. t. data-specific, model-specific, and hybrid are described. The authors also provide an experimental guideline for choosing a suitable augmentation strategy regarding the application context.

A key takeaway of these surveys is the magnitude of different generic graph augmentation methods available in the literature. Two important factors for distinguishing these methods in our context are the downstream task and the data that is operated on. The downstream task is similarity learning, which is a graph-level task and, thus, makes methods suitable that take a graph as input and a graph as output. The data that is operated on can be any part of the graph, e. g., features and topology, and these aspects should be considered.

Augmentation Methods From Business Process Management Literature

The presented surveys do not discuss specific augmentation methods for processes or semantic graphs. These augmentation methods come from the research field of BPM and are closely related to this work, but only a few approaches can be found. The approaches can, for example, be used for predictive process monitoring with DL methods [for an overview, see 56]. Although the augmentation methods in this work operate on process models, augmentation of event logs and process instances is also discussed, as literature about augmentation in BPM is scarce.

Leoni, Aalst, and Dees [34] enrich event logs by adding data to them. Although, their approach is not named augmentation, and it is not set in a deep learning context, the methods could probably be used for this purpose.

Venkateswaran *et al.* [68] examine data augmentation methods for facilitating robust deep learning applications in predictive process monitoring. However, concrete augmentation methods are not part of the proposed approach, as simple generic methods are used only in the experiment to test the robustness to changes.

Käppel, Schönig, and Jablonski [30] present an application of small sample learning [65] in BPM applications for dealing with insufficient amounts of event log training data. While this is not directly an augmentation method, the described concept and integration are still very close.

Käppel and Jablonski [29] present augmentation methods for event logs for the task of predictive process monitoring [56]. They present a total of nine augmentation methods, mostly generic methods adapted to event log data. All methods are explained in detail with the intentions behind them. The used augmentation pipeline also discusses the integration of the data augmentation methods into the existing learning procedure.

The presented approaches show that augmentation methods specifically for process models are largely unexplored. The work of Käppel and Jablonski [29] is the one closest to our approach, but still has a different application context, i. e., process event logs. This makes it challenging to reuse due to the differences between event logs and process models as input data to the augmentation methods. Please note that there is also a branch of work in BPM research that is concerned with the generation of (synthetic) event logs. Grüger *et al.* [14] discuss and summarize this branch in more detail and find that most approaches are insufficient to consider semantic data apart from the control-flow.

6.3 Synthesis and Augmentation of Process Models

This section discusses three categories of augmentation approaches and their ability to create useful augmented process models. To derive the augmentation approaches, we first discuss key criteria (see Sect. 6.3.1). These key criteria, especially the importance of semantic information for the use case of this paper, reveal another aspect to consider for the definition of augmentation methods: the so-called correctness (see Section 6.3.2). The notion of correctness is based on the observation that arbitrary augmentation of an input process model might lead to an output process model that is either syntactically or semantically incorrect, and, thus, violating rules of the process model structure and the underlying domain model, respectively. Given the definition and characteristics of correctness, three categories of augmentation that differ in the level of correctness they can guarantee are presented in Sect. 6.3.3, 6.3.4, and 6.3.5. Possible augmentation methods as well as properties of the resulting augmented process models are discussed for each category. As the augmented process models are to be used for training DL models, Sect. 6.3.6 establishes this connection and proposes an approach based on semi-supervised transfer learning. The idea here is to pair an unsupervised training procedure that uses only augmented data with a supervised training procedure that uses only original, non-augmented data, with no need for potentially costly label acquisition for the augmented data.

6.3.1 Key Criteria for the Proposed Augmentation Methods

The discussion of the related work (see Sect. 6.2.5) and the introduction of the process model representation and the domain (see Sect. 6.2.1 and 6.2.2) reveal a discrepancy between existing approaches and the needs of the process models considered in this work. Therefore, several key criteria for the usage of augmentation in the current application context are presented. Please note that these criteria should not be interpreted as requirements but more as guidance for the augmentation of sequential NEST process models.

Focus on semantic information

Semantic annotations are one of the most important parts of sequential NEST process models, which should also be reflected in augmentation methods that operate on the semantic annotations. Thereby, the semantic information is given by the process model, or more precisely its nodes and edges, and the underlying domain model. The semantic annotations can be very complex and heavily customized for the domain, making it difficult for generic augmentation methods to properly handle them. Semantic annotations are also an important part of the similarity computation, i. e., the main learning goal of our use case. Therefore, it should be noted that changes in the semantic annotations by augmentation might have a large impact on the use of the augmented process model as a training example.

Focus on control-flow and data-flow

The process model representation of the manufacturing processes to augment is sequential in its nature and by its definition (see Sect. 6.2.1), with the control-flow as one part and the data-flow as the other part [36, 37]. This is an important property to consider for augmentation methods, and it is also one property that is not considered by generic graph augmentation methods (see Sect. 6.2.5). Although not dealing with process models, Käppel and Jablonski [29] are also confronted with the sequential nature of process event logs but, generally, only consider control-flow and face an overall lower complexity of the used data with semantics playing a small role. It is important to note that the manufacturing processes in this paper (see Fig. 6.1) are an example of processes that rely mainly on their data-flow. The control-flow is, in these cases, indirectly determined by the data nodes connected via data-flow edges to the corresponding task nodes. There are other types of processes in which the control-flow perspective defined by the task nodes is the main factor, i. e., business processes [49].

Small changes may have large effects

Augmentation methods are generally designed to transform some kind of input data into a similar kind of output data, e. g., an image is rotated but still expresses the same content. This property is also desirable for the augmentation methods of the process models used in this work, and the extent of the change should be configurable for the methods. This offers the advantage that the methods can be fine-tuned to each domain or the specific training set, ultimately increasing the applicability.

6.3.2 Notion of Correctness

The definitions of the *sequential NEST process model* and the corresponding *domain model* (see Sect. 6.2.1) lead to a challenge for augmentation methods to include this syntactical and semantic knowledge into their algorithms. Because the basic assumption of all data augmentation approaches [10] is also applicable here: The augmented data should be as close to the original data as possible to ensure a high quality of the augmented data as DL training data. This "closeness", however, is hard to measure and strongly depends on the domain. One way could be the computation of the similarity (see Sect. 6.2.3) between the original process model and the augmented process model, where a high similarity value is desired. However, the associated computation procedure is infeasible for large-scale augmentation due to the involved computational complexity, and it requires augmented process models that are valid in terms of all constraints by the definition of sequential NEST process models and the corresponding domain model. Especially the fulfillment of the latter requirement can make augmentation methods very complex and vastly increase the knowledge acquisition effort for their implementation. For this reason, we introduce the notion of *correctness*, on the one hand, as an approximation of the quality of an augmented process model and, on the other hand, as a mechanism to allow augmentation methods to steer the tradeoff between useful augmentation methods and their complexity and required knowledge. The correctness could also be considered a key criterion in addition to those described in Sect. 6.3.1, but it is described separately, as it serves the important purpose of structuring the different augmentation approaches. The correctness of an augmented process model, in the sense that we use it for the remainder of the paper, is further divided into *syntactic correctness* and *semantic correctness*:

A *syntactically correct process model* is a valid sequential NEST process model (as defined in Sect. 6.2.1) and, thus, follows the representation of a NEST graph as well as features a sequential data-flow and control-flow without missing semantic descriptions.

Syntactic correctness addresses the structure of a process model w. r. t. to its nodes and the edges between them. A violation of the syntactic correctness is equal to a violation of the constraints given by the definition of the NEST graph [4] and the additional rules of sequential NEST process models. For instance, a simple violation of syntactic correctness would be the removal of a control-flow edge between two tasks, which, in turn, breaks the control-flow within the process model. With the property of being syntactically correct, only the structural conformity of a process model is guaranteed, without any validity checks of the semantic annotations regarding the domain model. Because semantic information is a key aspect of sequential NEST process models in general and manufacturing process models in particular, semantic correctness is defined as follows:

A *semantically correct process model* is syntactically correct and consistent with the definitions and constraints of the PDDL domain model, thus executable in the manufacturing context (see Sect. 6.2.2).

Semantic correctness completes the aspects of syntactic correctness by incorporating crucial information from the domain model. For example, a manufacturing process model is syntactically but not semantically correct if it is a valid sequential NEST process model, but the workpiece is processed by machines in the wrong order. The underlying factor for determining semantic correctness is the domain model, as it holds the information that describes the proper content and relations of all semantic information of a process model.

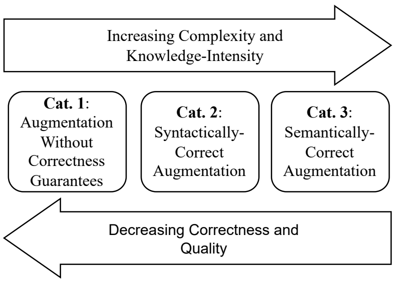

Increasing Complexity and Knowledge-Intensity
Cat. 1: Augmentation Without Correctness Guarantees
Cat. 2: Syntactically-Correct Augmentation
Cat. 3: Semantically-Correct Augmentation
Decreasing Correctness and Quality

Figure 6.3: Overview of the Categories of Augmentation Methods.

These two definitions lead to three levels of correctness that can be determined for an augmented process model: 1) A process model can be neither syntactically nor semantically correct, 2) a process model can be syntactically but not semantically correct, and 3) a process model can be semantically correct, and, thus, the process model is also syntactically correct. The following sections discuss augmentation approaches that are divided into three categories according to the level of correctness that they can guarantee for the resulting augmented process models. Figure 6.3 depicts these three categories.

We state that augmentation methods that preserve a higher level of correctness of the augmented process models are generally more complex and require more knowledge, i. e., are rather more knowledge-intensive. In turn, augmented process models with a higher level of correctness are typically more representative of their native process domain, and, thus, we assume them to be more useful for learning the properties of these processes. As a more accurate DL model also leads to better results when being applied as a similarity measure, the level of correctness is an important indication of the tradeoff between the quality of the augmented process models and the complexity and knowledge requirements of the respective augmentation techniques.

6.3.3 Augmentation Without Correctness Guarantees (Cat. 1)

Augmentation methods from this category are the least complex but hopefully still suitable for creating a large amount of augmented process models, representing the basic properties of the process domain. We propose to focus on augmentation based on *deletion* for this category. There are several reasons for this: These methods fit well to the main characteristics of this category, regarding complexity and ease-of-use. Another important factor refers to the knowledge required to implement methods based on deletion. Usually, these methods only require the original process model for the implementation, and no further information, such as data from other process models from a repository, in-depth knowledge about the process model representation or an in-depth understanding of the domain model. In the remainder of the section, we discuss the deletion of nodes, edges, and (parts of) semantic descriptions in more depth and examine the application of these methods for an example.

Randomly deleting nodes from the graph structure representing the process model is a well-known method from generic graph augmentation (see Sect. 6.2.5). The implementation is straightforward, but it might result in augmented process models that are not syntactically correct, mainly due to a broken sequential data-flow or control-flow (see Sect. 6.3.2). Similar to node deletion, process models can also be augmented by deleting edges. This method is also well-known from existing literature (see Sect. 6.2.5) and it is easy to understand and to implement. It is worth discussing nodes and edges separately, as their augmentations also have a different effect on the training procedure of the GEM (see Sect. 6.2.4). After the initial embedding of the nodes and edges in the embedder, the node information is propagated along the edge connections. This means that deleting a node removes the information from the propagation procedure and, ultimately, from the graph embedding. Deleting an edge changes the embedding by restricting the flow of information in the propagation layer of the GEM, which can have a significant influence on the resulting embedding.

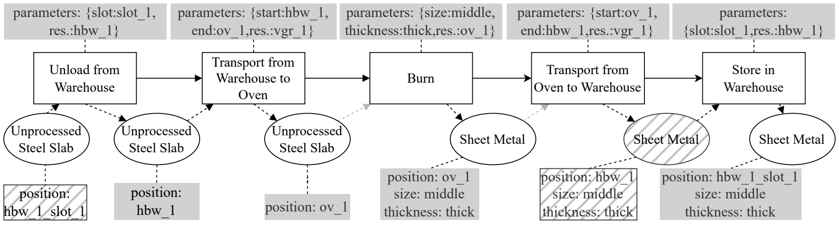

parameters: {slot:slot_1, res.:hbw_1}
parameters: {start:hbw_1, end:ov_1,res.:vgr_1}
parameters: {size:middle, thickness:thick,res.:ov_1}
parameters: {start:ov_1, end:hbw_1,res.:vgr_1}
parameters: {slot:slot_1,res.:hbw_1}
Unload from Warehouse
Transport from Warehouse to Oven
Burn
Transport from Oven to Warehouse
Store in Warehouse
Unprocessed Steel Slab
Unprocessed Steel Slab
Unprocessed Steel Slab
Sheet Metal
Sheet Metal
Sheet Metal
position: hbw_1_slot_1
position: hbw_1
position: ov_1
position: ov_1 size: middle thickness: thick
position: hbw_1 size: middle thickness: thick
position: hbw_1_slot_1 size: middle thickness: thick

Figure 6.4: Exemplary Syntactically Incorrect Process (based on Fig. 6.1).

Apart from the structural components, i. e., nodes and edges, the semantic descriptions are another integral part of the process model representation, which can also be addressed during augmentation. Thereby, the entire semantic description (or parts) of the semantic description of certain nodes or edges can be deleted. The resulting augmented process models are not syntactically correct, due to missing semantic descriptions (see Sect. 6.3.2). Although a node or an edge is mainly characterized by its semantic description in a sequential NEST process model, there are differences between deleting only the semantic description or the entire node or edge: On the one hand, the information on the type of node or edge is preserved and still part of the graph embedding procedure, which influences the resulting embedding. On the other hand, there is a node without semantic information, which will lead to a different behavior of the GNN during embedding and propagation. Therefore, it is useful to discuss both aspects separately.

An example of the application of three augmentation methods based on deletion is shown in Fig. 6.4. Thereby, the following three augmentations are applied successively to the exemplary process model shown in Fig. 6.1:

1. Random deletion of one data node
2. Random deletion of two data-flow edges
3. Random deletion of the semantic description of one data node

The changes made to the original process model are highlighted with hatched nodes and semantic descriptions, respectively, and edges in bright gray. The augmented process model no longer follows the sequential NEST process model definition (see Sect. 6.2.1) due to the broken sequential data-flow in three parts of the process model and a missing semantic description of the first unprocessed steel slab. It is also important to note that the state of the workpiece in the process model is lost due to the deleted data node and the semantic descriptions.

6.3.4 Syntactically Correct Augmentation (Cat. 2)

For this category of augmentation, we propose to focus on augmentation approaches that involve some kind of *replacement*. The main reason for this is the ability to reuse existing syntactically and semantically correct parts of the original process models to create augmented process models. Thus, it is possible to replace parts of a process model, e. g., a task node, with an equivalent part of the same or a different process model. Syntactic correctness is automatically ensured, since the structure of the augmented process model does not change compared to the respective original. The augmented process model differs from the original process model only on a semantic level. Existing generic graph augmentation methods from the literature can usually only hardly be reused in this regard (see Sect. 6.2.5) since they cannot specifically handle process models and, in particular, sequential NEST process models. The remainder of the section discusses replacements of task and data nodes, control-flow and data-flow edges, and (parts of) semantic descriptions in more depth, and examines the application of these methods for an example. We also discuss *swapping* as a special form of replacing, where the part to replace, e. g., a task node, and the replacement come from the same process model.

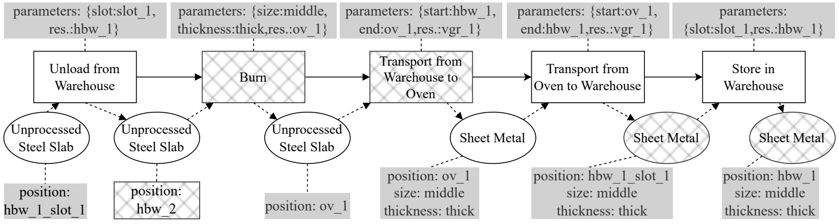


Figure 6.5: Exemplary Semantically Incorrect Process Model (based on Fig. 6.1).

The control-flow within a process model is given by the control-flow edges between the task nodes in the process model. It determines the order in which tasks are carried out on the data elements, e. g., a metal sheet is drilled. Augmentation can be aimed at the control-flow by replacing task nodes of the original process model with other task nodes, including their semantic description. Therefore, the replaced node receives all ingoing and outgoing edges of the former node after replacement (see tasks *Burn* and *Transport from Warehouse to Oven* in Fig. 6.5). Analogous to the control-flow, the data-flow within a process model, i. e., the data nodes that are consumed or produced along the control-flow, can also be targeted by augmentation similarly. In the manufacturing use case, this manipulates the input goods and the resulting goods of each manufacturing task in the process model. Such an augmentation replaces data nodes along the data-flow with other data nodes and their semantic descriptions, coming from the same or a different process model. As with the replaced task nodes, the connected edges are disconnected before replacing and then connected to the new nodes after the replacement.

Edges can also be replaced as a form of augmentation. The two most important types of edges to be augmented are control-flow edges and data-flow edges, due to the important NEST process model characteristics of sequential control-flow and data-flow (see Sect. 6.2.1).

The semantic descriptions of nodes and edges are also useful for replacements. Replacing the entire semantic description of a node or an edge (given the replacement node or edge has the same type) is equivalent to replacing the node or edge itself, as it is characterized and represented by its semantic description. Parts of semantic descriptions can also be replaced with values from other semantic descriptions of nodes or edges of the same type. Since the replaced parts of the semantic descriptions are from the same or other original process models and, therefore, semantically correct, the resulting process models might not be semantically correct due to the new context that the replaced node is now integrated in.

The influence of these augmentations on the training of the GEM is different from the deletions previously discussed (see Sect. 6.3.3). When comparing the original process model with one augmented by replacing nodes, edges, or parts of semantic descriptions, the structure of the process model remains unchanged, while the embedding procedure nonetheless changes. New information might be part of the process model if data from different process models of the repository is used for the replacement. If data from the same process model is used (i. e., swapping), then the augmented process model is handled differently by the GNN in the propagation and aggregation layers compared to the original one. This consideration between swapping and replacements from different process models might influence the quality of the resulting process models. On the one hand, swapping usually creates augmented process models that are relatively close to the original, since information within the process model is only moved along the control-flow or data-flow. Replacements with information from different process models, on the other hand, might increase the diversity of the training data with new, unseen process models.

An example of the application of three replacement-based augmentation methods is shown in Fig. 6.5. Thereby, the following three augmentations are applied successively to the exemplary process model shown in Fig. 6.1:

1. One random swap of two neighboring task nodes
2. One random swap of two neighboring data nodes
3. One random replacement of the item *position* in the semantic description of one data node

The changes made to the original process model are highlighted with cross-hatched nodes and semantic descriptions. The resulting augmented process model is still a valid sequential NEST process model (see Sect. 6.2.1) but it is not semantically correct. For instance, it is not valid to burn the workpiece before it was brought to the oven, as shown by the augmented process model.

6.3.5 Semantically Correct Augmentation (Cat. 3)

To create semantically correct augmented process models, it is important to do augmentation that fulfills the constraints given by the sequential NEST process model representation (see Sect. 6.2.1) and the domain model (see Sect. 6.2.2). For example, assume a manufacturing process with a wrong order of machines or wrong parameter settings for these machines. Training a DL model with these processes could lead to incorrect generalizations and, ultimately, poor model performance. To guarantee semantically correct augmented process models, automated planning techniques in combination with a planning domain description (see Sect. 6.2.2) are suitable.

Listing 6.2: Exemplary Planning Problem for Generating a Manufacturing Workflow.

```
(define (problem factory-problem) (:domain factory)
(:objects workflow_1 - processID)
(:init
...

(isReady vgr_1)  (isReady vgr_2)
(isReady hbw_1)  (isReady hbw_2)
(isReady ov_1)   (isReady ov_2)
(isReady mm_1)   (isReady mm_2)
(isReady wt_1)   (isReady wt_2)
(isReady sm_1)   (isReady sm_2)
(isReady pm_1)   (isReady dm_2)
(isReady hw_1)

; workflow_1 initial state
(bucketAt hbw_1_slot_1_pos)
(isSteelSlab workflow_1)
(at workflow_1 hbw_1_slot_1_pos)

(=(total-cost) 0)

)
(:goal (and (at workflow_1 hbw_1_slot_1_pos)
(isSheetMetal workflow_1)
(isSheetMetalWithSize middle workflow_1)
(isSheetMetalWithThickness thick workflow_1))

(:metric minimize (total-cost))
)
```

Automated planning [12, 17] is a technique from artificial intelligence and, thus, is often also called AI planning. AI planning can be used as a generative problem solver and is based on the concept of representing a planning problem as an initial state, a goal state that should be reached, and a set of actions that are used for state transitions. The goal of solving the planning problem is to find a sequence of actions so that the goal state is reached based on the facts given in the initial state. In this context, a sequence of actions is rather similar to a sequence of tasks in a process. For this reason, AI planning has also gained considerable interest in the BPM area in recent research works [36, 37, 39, 42, 43, 45, 58]. In Marrella [41], it has been examined in which phases of the BPM lifecycle, AI planning can be utilized. One use case for AI planning in BPM is the generation of process models in the *Design* phase. In this work, we generate new process models and transform existing process models to increase the set of semantically correct process models in the context of data augmentation.

To augment process models, a comprehensive planning domain model consisting of the available planning actions and a planning problem comprised of the initial state and the goal state that should be reached is required. For the manufacturing application scenario used in this work, we use a semantic service-based architecture for the smart factory [38, 64]. Based on this architecture, a domain expert creates a comprehensive planning domain description based on the PDDL [47] for the application scenario, as introduced in Sect. 6.2.2. With the planning domain, process models can be either augmented by *synthesis*, i. e., generated from scratch solely based on the planning domain and a randomly generated planning problem, or *transformation*, i. e., based on an original process model [for more information on this distinction, see 50].

In addition to the planning domain, a corresponding planning problem is required to generate process models for data augmentation. These planning problems can be created automatically and randomly in the context of the manufacturing application scenario. Given the problems, it is possible to solve them by a corresponding planner. Listing 6.2 represents a part of the planning problem to generate the workflow illustrated in Fig. 6.1. The random planning problem generator assumes that all machines are available in the smart factory (Lines 6–13) and specifies an object representing the workflow to be planned (Line 2). Subsequently, the initial state of the planning problem is specified by randomly choosing possible initial states (Lines 16–18). Furthermore, the total cost function is initialized (Line 20). To define the goal of the workflow, the goal state of the planning problem needs to be specified. For this purpose, the random planning problem generator determines for each possible property of the workpiece whether it should be included, e. g., drilled or not drilled, and, if so, in which concrete form, e. g., number and size of the holes. In this context, it is important to know that all properties, desired or not, are part of the planning problem. However, please note that Listing 6.2 only contains the desired ones due to space restrictions (Lines 23–26)[4].

Transformational augmentation is possible with AI planning similarly. Instead of directly manipulating the process models (as discussed in Sect. 6.3.3 and 6.3.4), the native planning problems or the configuration of the planner are augmented. In the first case, it is possible to create the initial state and the goal state of the origin process model and then augment the *initial state*, e. g., a change of the position of the workpiece, the *goal state*, e. g., a change in the thickness of the workpiece, or the *final metric*, e. g., a minimization of the number of tasks instead of the total cost. These changes are then processed by the planner and an augmented process model is created. A different way of creating augmented process models with AI planning is an augmentation of the planner configuration. By changing the heuristic or the configuration of the heuristic that the planner uses to solve the planning problem, it is also possible to create different resulting process models. The augmented process model and the original one are most likely similar for both variants, since they use a similar planning problem and planner configuration. In addition, the proposed approach can also be used for planning parts of process models, e. g., gaps in the control-flow between two activities or alternative routes for certain parts. This offers the advantage that the resulting process models are even more similar as only parts change, and they could be more effective as training examples.

6.3.6 Unsupervised Transfer Learning with Augmented Processes

Augmented process models pose a challenge in handling them in the training process of a supervised embedding model such as the GEM (introduced in Sect. 6.2.4) due to the required

[4] All PDDL files created for the paper are publicly available. See the statement on data availability at the end of the article for more information.

ground-truth similarities. On the one hand, it is unclear if the underlying ground-truth similarity measure can deal with syntactically or semantically incorrect process models. On the other hand, the procedure for computing the ground-truth similarities is usually computationally expensive and, when using measures based on A*, often entirely infeasible for large amounts of data (see Sect. 6.2.3 and Schuler *et al.* [62] for more information). Therefore, we propose to use augmentation in combination with a semi-supervised two-phase training approach, featuring an unsupervised pretraining phase and a supervised graph similarity learning phase [62]. This provides the advantage that the augmented process models are used in an unsupervised training procedure without the need for label acquisition, i. e., the computation of ground-truth similarities. The original, non-augmented process models are used in the supervised training procedure, where the model learns to predict the ground-truth similarities (see Sect. 6.2.4). Due to this separation, the augmented process models are still used to train the graph embedding model, but label acquisition is greatly reduced, compared to a fully supervised training setup.

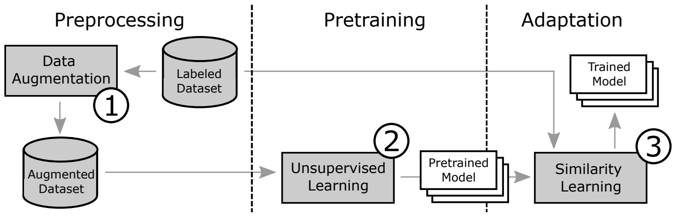


Figure 6.6: Architecture for Semi-Supervised Transfer Learning With Its Three Main Steps [Source: 62].

The unsupervised and the supervised training phases are unified by transfer learning (see Fig. 6.6 for an overview of the components and the main steps). The fundamental concept behind transfer learning is that the knowledge acquired during the pretraining phase from a source domain (see step 2) can be applied in the adaptation phase within a target domain (see step 3), as outlined in Kudenko [32] and Tan *et al.* [66]. In our specific context, a graph embedding model serves as this knowledge, transitioning from the unsupervised pretraining phase to the supervised adaptation phase.

The underlying assumption here is that the augmented training data is sufficient for the unsupervised pretraining to learn the characteristics of the process domain and, thus, to diminish the necessity for an extensive amount of labeled data during the subsequent supervised training. The existing literature [e. g., 70] supports this strategy.

Unsupervised Triplet Learning

During the pretraining phase, a *Siamese Neural Network (SNN)* with a triplet loss [20, 54, 61] is used to learn unsupervised graph embeddings (see Fig. 6.7). The specific graph embedding model is the GEM (introduced in Sect. 6.2.4). The concept revolves around creating embeddings that are alike for graphs with similarities, and distinct for graphs that differ significantly (in terms of vector space similarity). Given the absence of computed ground-truth similarities for the graphs within the training dataset, the loss function operates on graph triplets comprising an *anchor*, a *negative*, and a *positive*. Thereby, the similarity for the pair of the anchor and the positive is maximized, and the similarity for the pair of the anchor and the negative is minimized. The formal definition of this triplet loss is given in Eq. 6.1: Let $T = (x^a, x^p, x^n)$ be a triplet consisting of one input vector for anchor x^a, positive x^p, and negative x^n, and let $f(x_i) = m_i \in \mathbb{R}^n$ represent the embedding of an input vector x_i as the n-dimensional embedding m_i. The loss function L is minimized with α as a margin hyperparameter and N as the cardinality of a batch of triplets to train a model [61].

$$L = \sum_{j}^{N} \max\{0, \|m_j^a - m_j^p\|^2 - \|m_j^a - m_j^n\|^2 + \alpha\} \tag{6.1}$$

The main challenge for this training method in an unsupervised context is the composition of the triplets, as the aforementioned assumptions regarding the relationships of anchor and positive and anchor and negative should be satisfied. With a given anchor process model, the negative is selected randomly from the training set, which is straightforward. The positive is selected as an augmentation of the anchor process model, since the augmentation methods generally produce augmented process models that are similar to the original ones. Please note the special case when composing triplets of augmented process models generated by AI planning via synthesis (see Sect. 6.3.5). Since there is no original process model for each augmented one, it is possible to use the augmented process model as the anchor and the positive.

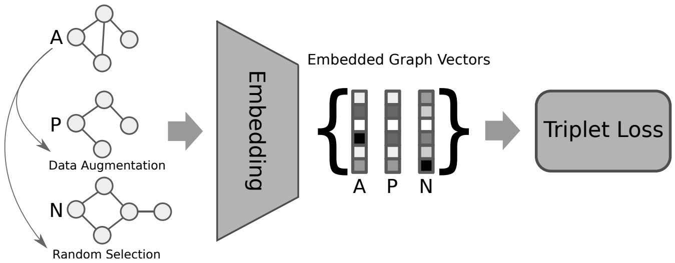


Figure 6.7: Unsupervised Triplet Graph Embedding using an Anchor (A), a Positive (P), and a Negative (N) [Source: 62].

Graph Embedding with the GEM

The GEM, as described in Sect. 6.2.4, serves a dual role in both the *pretraining* and *adaptation* phases. In the adaptation phase, its function aligns with its original purpose [see previous work in 23], involving the supervised embedding of graph pairs. Nevertheless, the pretraining phase, which employs the triplet learning method, necessitates specific adjustments to the model. Contrary to its original use with pairs of graphs, the GEM is trained with graph triplets. Each graph in the triplet is embedded using shared parameters. As the GEM supports processing tuples of graphs independent of their number out of the box, this change is more focused on the implementation. These refinements to the GEM enable the successful execution of unsupervised pretraining and the subsequent supervised adaptation of the model within the transfer learning framework.

6.4 Experimental Evaluation

The experimental evaluation aims to examine the effects of process model augmentation on similarity learning with the GEM. Therefore, the GEM is trained with different augmented training sets and is compared to the GEM trained with non-augmented data. The comparison between different GEMs is done according to several quality metrics, measuring the similarity prediction errors. The augmented training sets are created with different combinations and configurations of the three categories of augmentation, introduced in Sect. 6.3.

6.4.1 Hypotheses

We investigate the following hypotheses:

H1 Training the GEM on augmented process models increases the quality compared to the GEM being trained on non-augmented process models.

H2 Training the GEM on augmented process models of Cat. 2 (see Sect. 6.3.4) increases quality compared to the GEM being trained on augmented process models of Cat. 1 (see Sect. 6.3.3).

H3 Training the GEM on augmented process models of Cat. 3 (see Sect. 6.3.5) increases the quality compared to the GEM being trained on augmented process models of Cat. 1 or 2 (see Sect. 6.3.3 and 6.3.4).

H4 Training the GEM on process models augmented with methods from several categories increases quality compared to the GEM being trained on process models, augmented with the respective methods from the individual categories.

The hypotheses show the expectations of the authors regarding the experimental results. As a baseline, it is expected that using augmented data for training leads to a better model than the original (non-augmented) data. We additionally claim that the level of correctness influences the quality of the model, i. e., higher levels of correctness lead to better models. The final hypothesis addresses the combination of augmentation methods with different levels of correctness that are expected to show an increased quality compared to the respective individual methods.

6.4.2 Datasets

The main dataset (denoted as D_{100}) used throughout the experiments comprises workflows originating from a smart manufacturing IoT environment, representing sample production process models (see Sect. 6.2.1 or previous work in Malburg, Hoffmann, and Bergmann [37] for more details). The dataset contains 100 training cases, 20 validation cases, and 20 test cases. To measure the effects of augmentation on differently sized datasets, we also use D_{50} with 50 training, 10 validation, and 10 test cases, D_{200} with 200 training, 30 validation, and 30 test cases, and D_{300} with 300 training, 35 validation, and 35 test cases. For the cases contained in these datasets, $D_{50} \subset D_{100} \subset D_{200} \subset D_{300}$ holds. These datasets are created using the planning approach by synthesis introduced in Sect. 6.3.5 to reduce the manual effort to create a data set of process models. In a real-world scenario, a manually modeled dataset of process models would probably exist and could be used instead.

6.4.3 Augmentation Strategies

To examine the effects of augmentation on the quality of the GEM, the aforementioned datasets are augmented. The resulting datasets are presented in Tab. 6.1. Each dataset is augmented with different strategies: 1) methods without correctness guarantees (as introduced in Sect. 6.3.3), 2) syntactically correct methods (as introduced in Sect. 6.3.4), 3) semantically correct methods (as introduced in Sect. 6.3.5), 4) the combination of the strategies 1 and 2, and 5) the combination of the Strat. 1–3. Since Strat. 3 and Strat. 5 involve AI planning, we also investigate the differences between synthesis and transformational augmentation. Consequently, Strat. 3.1 and Strat. 5.1 use synthesis while Strat. 3.2 and Strat. 5.2 use transformations. The augmentation methods that are involved in each strategy are applied in parallel, that is, each augmentation method augments the original process model. The result is one original process model that leads to one

Strategy	Category	Methods	Factor	Sizes (D_{100}, D_{50}, D_{200}, D_{300})
1	1	Deletion of two control-flow edges; Deletion of one data node; Deletion of one item of the semantic description of two task nodes	4	400, 200, 800, 1200
2	2	Swap of two pairs of neighboring data nodes (in terms of data-flow); Swap of two pairs of neighboring task nodes (in terms of control-flow); Swap of one item of the semantic descriptions of three task nodes	4	400, 200, 800, 1200
3.1 & 3.2	3	Synthesis (3.1) and transformation augmentation (3.2) of process models via AI planning	4	400, 200, 800, 1200
4	1 and 2	Augmentation according to Strat. 1 and Strat. 2	7	700, 350, 1400, 2100
5.1 & 5.2	1, 2, and 3	Augmentation according to Strat. 1, Strat. 2, and Strat. 3.1 or Strat. 3.2	10	1000, 500, 2000, 3000

Table 6.1: Augmentation Strategies in the Experiments.

additional augmented process model. Given $n \in \mathbb{N}$ as the number of original non-augmented process models and $m \in \mathbb{N}$ as the number of augmentation methods, the number of process models after augmenting a dataset with one strategy equals $n + n \cdot m$. Therefore, the original process models as well as one augmented dataset for each augmentation method are part of the final augmented dataset of the strategy, leading to a factor of $m + 1$ for the multiplication of the training data.

We use three deletion-based augmentation methods for Strat. 1, 4, and 5, three augmentation methods based on replacements for Strat. 2, 4, and 5, and two augmentation methods based on AI planning for Strat. 3 and 5. All of these augmentation methods contain random aspects that can be seeded to allow reproducibility. The deletion-based methods work as follows:

Edge Deletion

This method randomly deletes edges, including their semantic description, in the original process model. The parameterization possibilities of the method comprise the number of edges to delete and the types of edges to delete. The method is configured to delete two control-flow edges in the experiments.

Node Deletion

This method randomly deletes nodes including their semantic description in the original process model. The method can be parameterized by the number of nodes to delete, the types of nodes to delete, and whether the connected edges should also be deleted. The configuration of the method in the experiments deletes one data node without deleting the connected edges.

Deletion Within Semantic Descriptions

Randomly deleting items from semantic descriptions is the core of this augmentation method. Therefore, an attribute of the semantic descriptions of a certain type of node or edge is selected first and then deleted for several nodes. The method can be parameterized by the number of nodes or edges to target and the respective type of node or edge. The method is configured to delete one item in the semantic descriptions of two task nodes in the experiments.

In addition to the methods based on deletion, the experiments also use methods based on replacements. These methods are implemented as follows:

Task Order Swapping

This method swaps pairs of task nodes that are neighbors according to the control-flow, i. e., task nodes that are directly connected by a control-flow edge. The swapped nodes keep their original semantic descriptions but replace the former node in all edge connections. The method can be configured by the number of swaps to perform, which is two in the experiments.

Data Order Swapping

Analogous to swapping the task order, this method swaps neighboring data nodes along the data-flow. Two data nodes are neighbors if one of those nodes is consumed and the other one is produced by the same task node. The number of swaps to perform can be set as a parameter, and this value is two in the experiments.

Swapping Within Semantic Descriptions

This augmentation method processes the semantic descriptions of nodes or edges and swaps contained items. Pairs of nodes or edges of the same type are first identified, an item of their semantic descriptions to swap is selected, and then this item is swapped. The method can be parameterized by the number of nodes or edges to target and the respective type of node or edge. The method is configured to swap one item in the semantic descriptions of three task nodes in the experiments.

For the augmentation methods of Strat. 3 and Strat. 5, we distinguish between synthesis (Strat. 3.1 and 5.1) and transformational augmentation (Strat. 3.2 and 5.2):

Synthesis

This method generates new process models by using the PDDL domain model and randomly setting an initial state and a goal state (see Sect. 6.3.5 for more information) for the search procedure of a planner. In our experiments, we generate the same number of process models as the augmentation methods of the other strategies to keep the size of the resulting datasets comparable. This means that a dataset augmented with Strat. 3.1 contains three times the number of generated process models compared to the original process models.

Transformation

Transformational augmentation via AI planning (see Sect. 6.3.5 for more information) in the experiments is conducted by using different planner heuristics for solving the planning problem of the original process model. Instead of utilizing A* search with the landmark-cut heuristic (*lmcut*) [18] that creates the original process models based on their planning problems, the augmented process models are created with three different variants of a lazy greedy best-first search: The first used heuristic is the context-enhanced additive heuristic (*hcea*) [19] and the second heuristic is fast-forward (*hff*) [21]. Both heuristics are also used in combination as a third variant (*hff-hcea*).

Tables 6.3 and 6.4 in the appendix give a statistical comparison of the original datasets and the respective augmented ones. The tables are concerned with the number of nodes and edges of all graphs in the respective datasets. In detail, we present the average, the minimum, and the maximum numbers. While Fig. 6.3 shows the absolute values, Fig. 6.4 gives a comparison of the average of these numbers w. r. t. the *Kolmogorov-Smirnov (KS)* statistic. The displayed value is the complement of the statistic, and it measures how well the values of two samples align. Thereby, a value of 1 indicates perfect alignment, meaning that both samples most likely come from the same distribution, and a value of 0 means the worst alignment.

6.4.4 Model Training and Metrics

The training procedure for each variant of the GEM is two-phased, following the proposed transfer learning approach (see Sect. 6.3.6). For each augmented dataset, a pretraining is carried out with the triplet learning approach. The resulting model is then used as the base model for adaptation. Each pretrained model is fine-tuned with the data of the respective base dataset. For example, the dataset D_{100} is augmented with Strat. 1, resulting in the augmented dataset D'. The data of D' is then used for pretraining, and the pretrained model is trained with the data of D_{100} in the adaptation phase. We use the same hyperparameters for all pretraining runs and adaptation runs, respectively, which improves their comparability. These hyperparameters are determined by hyperparameter tuning [22] that is done once for the pretraining of D_{100} augmented with Strat. 1 and the adaptation of the resulting model. All trainings run for a fixed number of 40 epochs, and the best model is selected afterward according to the validation results of each epoch (similar to early stopping). Each of the adaptation training runs is done three times, and the evaluation results for these three models are averaged to get more consistent results.

Different variants of the GEM are compared by using them as a similarity measure for the scenario of a search in a process model repository. Given a single query process model from the test set, similar process models are searched based on the pairwise similarity between the query process model and all process models in the repository. The repository thereby contains all process models from the training set. This setup reflects real-world scenarios where a certain repository exists and new, unknown process models are used as queries. The inspected metrics for the comparison of different variants of the GEM are the MSE, the MAE, and the *Root Mean Squared Error (RMSE)*. All three metrics measure the error between the predicted similarity and the ground-truth similarity. While the predicted similarity stems from the corresponding GEM variant, we use ground-truth similarities that are computed by an A* graph matching algorithm (see Sect. 6.2.3 and Bergmann and Gil [4]). The MSE computes the error as the average squared error of the difference between the predicted and the ground-truth similarity. The MAE does not use the square of these differences, but rather their absolute value. The RMSE is the square root of the MSE. Lower values of all three metrics are generally better.

6.4.5 Experimental Results

The results of the experiments are presented in Tab. 6.2. Two tables are provided with the absolute values of the error metrics in Tab. 6.2a and the relative differences in Tab. 6.2b. The tables cover the computed metrics of the different variants of the GEM for all datasets. These variants include the standard, non-augmented model and the models augmented with different strategies, as introduced in Sect. 6.4.2–6.4.4. The tables also include a variant of the A* graph matching similarity measure (see Sect. 6.2.3 and Bergmann and Gil [4]) that is also used to compute the ground-truth similarities. This variant is included to connect the results of the augmented models to previous work, where embedding-based similarity measures are compared to measures based on A* graph matching (see Sect. 6.2.4 and Hoffmann and Bergmann [23] and Hoffmann *et al.* [26] for more details). This A* measure differs from the one used to compute the ground-truth similarities in the sense that it is restricted in the maximum number of solution candidates to maintain during the search procedure. The value is chosen such that the average computation time is approximately equal to that of the other measures to allow a fair comparison with a similar time budget. The same strategy is also used in previous work [23, 26].

	Model	100			50			200			300		
		MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓
	Standard	0.1772	0.1851	0.0384	0.1603	0.1690	0.0308	0.1087	0.1205	0.0157	0.1234	0.1326	0.0216
Augmented	**Strat. 1**	0.1204	0.1311	0.0187	**0.0746**	**0.0841**	**0.0083**	0.0775	0.0908	0.0101	0.0984	0.1068	0.0139
	Strat. 2	0.1031	0.1142	0.0146	0.0802	0.0904	0.0098	**0.0747**	**0.0894**	**0.0091**	0.1163	0.1229	0.0179
	Strat. 3.1	**0.0911**	**0.0996**	**0.0117**	0.0917	0.1009	0.0125	0.0888	0.1039	0.0117	**0.0972**	**0.1048**	**0.0139**
	Strat 3.2	0.2754	0.3162	0.1167	0.1253	0.1398	0.0243	0.3195	0.3344	0.1422	0.2439	0.2524	0.1051
	Strat. 4	0.1118	0.1248	0.0173	0.1774	0.1963	0.0450	0.0863	0.1032	0.0130	0.1204	0.1310	0.0195
	Strat 5.1	0.1769	0.1969	0.0438	0.1583	0.1706	0.0339	0.1378	0.1584	0.0281	0.1005	0.1118	0.0153
	Strat. 5.2	0.1566	0.1732	0.0329	0.1363	0.1495	0.0264	0.1096	0.1258	0.0177	0.1729	0.1832	0.0664
	A* measure	0.1656	0.1709	0.0298	0.1764	0.1799	0.0329	0.1816	0.1844	0.0344	0.1560	0.1618	0.0270

(a) Absolute Errors.

	Model	100			50			200			300		
		MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓	MAE ↓	RMSE ↓	MSE ↓
	Standard	0.1772	0.1851	0.0384	0.1603	0.1690	0.0308	0.1087	0.1205	0.0157	0.1234	0.1326	0.0216
Augmented	**Strat. 1**	-32.1%	-29.2%	-51.3%	**-53.5%**	**-50.2%**	**-73.0%**	-28.7%	-24.6%	-35.7%	-20.3%	-19.5%	-35.5%
	Strat. 2	-41.8%	-38.3%	-62.0%	-50.0%	-46.5%	-68.3%	**-31.3%**	**-25.8%**	**-41.8%**	-5.7%	-7.3%	-17.0%
	Strat. 3.1	**-48.6%**	**-46.2%**	**-69.4%**	-42.8%	-40.3%	-59.4%	-18.3%	-13.7%	-25.3%	**-21.2%**	**-20.9%**	**-35.7%**
	Strat 3.2	55.4%	70.8%	204.3%	-21.9%	-17.2%	-21.0%	194.0%	177.5%	805.7%	97.7%	90.4%	387.5%
	Strat. 4	-36.9%	-32.6%	-55.0%	10.7%	16.2%	46.4%	-20.6%	-14.4%	-17.2%	-2.4%	-1.2%	-9.6%
	Strat 5.1	-0.2%	6.4%	14.2%	-1.3%	0.9%	10.2%	26.7%	31.5%	78.8%	-18.6%	-15.7%	-28.9%
	Strat. 5.2	-11.6%	-6.4%	-14.2%	-15.0%	-11.5%	-14.3%	0.8%	4.4%	12.5%	40.1%	38.2%	207.7%
	A* measure	-6.6%	-7.7%	-22.3%	10.0%	6.5%	6.9%	67.1%	53.0%	119.1%	26.5%	22.1%	25.2%

(b) Relative Errors.

Table 6.2: Evaluation Results.

The first observation in the data is that all three metrics behave very similar when comparing the same model, with only small deviations in some cases. Therefore, in the following, we do not describe the data w. r. t. each metric individually but rather as a combination. It is apparent that the best-performing model is never the non-augmented one but rather one of Strat. 1–3.1 for all domains. While these models are generally better than the non-augmented model (indicated by negative percentages in Tab. 6.2b) this is not the case for Strat. 3.2–5.2. There, we can observe an increased error for some domains, e. g., Strat. 5 of D_{50}. The metric values for the Strat. 1–3.1 are similar across all domains, and the observed variations do not follow a clear trend between different domains. The data also shows that similar amounts of training data lead to models of similar quality, regardless of the data quality. This becomes apparent, for instance, when comparing the model augmented with Strat. 3.1 of domain D_{50} with the non-augmented model of D_{200}. Both are trained with 200 training process models and, with an MAE of approximately 0.1, the performance is very similar. However, training with more data does not necessarily lead to better results, as shown by the models of Strat. 3.2 and 5 compared to the models of Strat. 1–3.1 of D_{200}. The comparison between synthesis and transformational augmentation with AI planning reveals mixed results. Strat. 3.2 seems to perform worse than Strat. 3.1, on average, and Strat. 5.2 seems to generally outperform Strat. 5.1. It is noticeable that Strat. 3.2 performs very poorly for all domains except D_{50}. The measure based on A* is outperformed in almost all comparisons with the embedding-based measures which is in line with the experimental results of previous work [23, 26]. It also appears that the larger domains, i. e., more training data, favors the embedding-based measure more.

When interpreting the results of the embedding-based measures, a great potential of the process augmentation approach is revealed. The augmented datasets lead to a reduction of the prediction error of up to 53% in terms of MAE. This effect weakens with larger datasets, but it is still significant, i. e., a reduction of 21% for the largest domain D_{300}. More original training data is expected to reduce the need for augmented data, which is in line with our results. Furthermore, the results indicate that the augmentation methods of the different categories do not lead to trained models with different levels of performance. The observable differences between the augmented models of the Strat. 1–3 follow no trend across the domains, and are most likely variation. The error reductions appear to be possible with augmented process models of higher and lower quality. Additionally, the results indicate that the highly augmented datasets, i. e., the datasets augmented with Strat. 4 and 5, perform worse than the datasets of Strat. 1–3 on average. This is contrary to the common expectation that more (augmented) training data improves the performance of DL models. And while this expectation holds for the comparison between the differently sized domains, the reason for the adverse effect of Strat. 4 and 5 might be a drift between the original datasets and the highly augmented ones. This means that the augmented data does not adequately represent the original data anymore. Furthermore, the different results of synthesis and transformational augmentation via AI planning might be a result of the different ways the augmented datasets are created. Tab. 6.4 indicates that most of the statistical numbers point to a close alignment between the augmented datasets and the respective original ones, except for Strat. 3.2 and 5.2. It appears that the applied method of augmenting process models by using different heuristics to solve the same planning problem leads to large variations in the resulting process models, thus the low values of the KS statistic. In contrast, Strat. 3.1 and 5.1 have much higher KS values that indicate an advantage of synthesis in this scenario.

To accept or reject the hypotheses, the experimental results are assessed statistically and qualitatively. The statistical assessment is conducted by paired, two-sample, one-sided t-tests[5] between the similarity predictions of two individual models to check for unequal mean values. For instance, to get a statistical confidence for Hypothesis H1, the p-value of the t-tests of all pairs of one augmented model and the standard model in all domains (28 in total) are checked to see if the augmented models predict similarities that are different from the standard model. As the t-tests only check for differences of the mean values, the hypotheses are also checked qualitatively for better or worse predictions, given by Tab. 6.2. The following conclusions for the hypotheses are drawn based on the statistical and qualitative results: Hypothesis H1 is accepted, due to statistical significance ($p < 0.05$) and since we can see a strong positive influence of the augmented datasets on the model performance compared to the original data. Although some models perform worse than the baseline, a better performing model can still be found in a simple model selection or hyperparameter tuning procedure. The Hypotheses H2 and H3 are rejected, as there are no clear trends between models being trained on the augmented data of different categories of augmentation methods. The statistical results for both hypotheses are mostly statistically significant ($p < 0.05$) apart for the comparisons of Strat. 1 and 2 as well as Strat. 2 and 3.1 for domain D_{50}. The t-tests for Hypothesis H4 show statistical significance ($p < 0.05$) except for the comparison between Strat. 5.1 and 2 in domain D_{300}. The hypothesis is still rejected due to the generally worse performance of the Strat. 4 and 5 compared to the Strat. 1–3.

6.4.6 Discussion

One of the key outcomes of the experiments is the observation that DL models trained on augmented process models with a guaranteed correctness do not consistently outperform DL models trained on augmented process models without correctness guarantees. This appears to be con-

[5] The detailed results of the statistical tests are part of the supplementary data, as referenced in the data availability statement at the end of the paper.

trary to the common expectation that training data with a higher quality leads to trained models of noticeably better quality. We think that this effect cannot be seen here due to the following reason: There are enough augmented process models created such that the individual incorrect parts do not lead to a lowered quality of the trained model, and the important characteristics of the original process models are still represented by the majority of the augmented data. The following example demonstrates this: An augmented process model where a control-flow node is deleted does not allow the trained model to learn the control-flow as it should be. If the other augmented process models of the training dataset are augmented differently, the trained model can still learn the correct characteristics of the control-flow, as the combined characteristics of the training examples represent all the needed information to learn. That is, the (hypothetical) average augmented training example represents the same distribution as the original non-augmented training example. This could be the reason, the different categories of augmentation methods perform similarly. While this is generally positive, there is a risk for small datasets as well as datasets with very distinct individual examples because in these scenarios, each example is more valuable to the training procedure and a significant amount of information might be lost during augmentation by methods without correctness guarantees.

Another noticeable observation refers to the Strat. 4-5 that combine multiple augmentations from the other strategies. The models trained on these augmented process models perform mainly worse than the other augmented models. We interpret the data as an indication of a drift between the non-augmented and augmented datasets due to the highly diverse training sets. It might be the case that too much augmentation with different methods shows the opposite of the effect that was discussed before, ultimately reducing the quality of the trained model. Another factor that might play a role, especially in the training procedure with large amounts of training data, is overfitting. Although the effects are mitigated by early stopping in every training run, there are still hints of it when inspecting the validation loss in comparison to the training loss in the training statistics. The decision to perform hyperparameter optimization only once and reuse the results for all other training runs (see the details in Sect. 6.4.4) was made deliberately to keep the experimental results between different models comparable. Nevertheless, there might be room for improvement by performing an individual hyperparameter optimization for each training procedure.

In summary of the experiments, we recommend utilizing augmentation to enlarge the data available for the training procedure of a DL model. The ratio between original and augmented data should be chosen carefully. Larger original training data sets appear to be more robust to augmented data of low quality, but the gain expected by using this augmented data in the training procedure appears to be lower. Analogously, smaller original training data sets might have a higher risk of augmented data of low quality disrupting the training procedure, but the potential gains by training with a larger data set are higher.

6.5 Limitations and Transferability to Other Domains

In this section, we discuss the limitations and main assumptions of the proposed approach and, particularly, examine the transferability of the approach to other domains.

Sequential NEST Process Model Representation

One of the main assumptions of the approach is the restriction of the discussed categories of augmentation methods to sequential NEST process models, without any rich control-flow elements such as AND or XOR elements (see Sect. 6.3.2). This restriction enables this paper to have a proper evaluation of different augmentation methods w. r. t. the correctness of the resulting augmented process models and, thus, the influence of training data with a higher correctness on the quality of the trained embedding models. Transferring the same principles onto other, arbitrary domains seems challenging at first glance, but we still expect it to be possible. For example, it is possible to split the whole process model into several smaller process models, e. g., one subprocess before the control-flow element, one subprocess with the control-flow elements and their branches, and one subprocess with the rest of the process model. The augmentation methods can then recursively be executed on these (sequential) subprocesses to augment them. Thereafter, the processes are put together in a single combined augmented process model.

Selection of Augmentation Methods and Planning Domain Definition

The presented augmentation categories and the augmentation methods from these categories are selected to fit the assumptions on correctness and the described key criteria. However, we still expect the described augmentation methods to be universally applicable to other domains, especially when correctness is not relevant. The main determining factor to deciding on the concrete methods is most likely the effort that can be put into the implementation of the augmentation and training procedures described in this work. As the evaluation shows no noticeable differences between the augmentation methods of the three categories, even simple augmentations can perform well. But, we are not sure if this also applies to other domains and, therefore, encourage not to disregard methods with correctness guarantees. The use of AI planning in particular can be challenging, even though AI planning is already being used successfully in the BPM area [36, 37, 39, 42, 43, 45, 58] and in different lifecycle phases [41]. However, for using AI planning, a comprehensive formal planning domain is required, which is difficult to achieve in real-world application scenarios and practice [39, 51]. To remedy this problem, already available encoded knowledge, e. g., in the form of an ontology, can be reused and converted to a formal planning domain definition [39]. In addition, there exist approaches [for a comprehensive overview, see 28] that try to remedy the efforts for creating a formal planning domain definition. For example, in McCluskey *et al.* [46] and in Zhuo, Muñoz-Avila, and Yang [80] the authors use demonstrations or traces, i. e., in our case available process models or event logs of executed process instances, to automatically derive the planning domain. In addition, special developed user interfaces [e. g., 71] or generative models such as ChatGPT can be used to remedy these efforts [e. g., 16, 35, 67]. However, and even if domain experts or already encoded and formal knowledge or demonstrations and traces are available, it means a certain amount of effort that needs to be investigated. If this effort is too high or there is no other formal knowledge available that can be used to create a formal planning domain, augmentation methods of Cat. 1 and Cat. 2 should favorably be used, as they are universally applicable independent of the concrete domain.

Similarity Learning Use Case

The presented approach and its evaluation is built around the use case of similarity predictions between pairs of process models. This use case is chosen, as previous work of the authors [e. g., 23] also deals with it, and it is relevant to BPM literature [e. g., 60]. However, the approach is designed to be applicable to other use cases where DL models are utilized to improve BPM applications. That is, the augmentations (see Sect. 6.3.3 to 6.3.5) are not dependent on the use case of similarity prediction but can rather be used to enlarge the training data for arbitrary use cases. Additionally, the transfer learning approach (see Sect. 6.3.6) is also rather generic and can be applied to other DL models and training procedures as well. The core feature is the combination of the unsupervised and the supervised training procedures that is not specific to similarity learning. In fact, previous work [62] experiments with a very similar transfer learning setup and shows its suitability across multiple domains. Furthermore, the main contributions might also be interesting to BPM applications that usually deal with process instances or process traces, e. g., predictive business process monitoring [56]. The processes used there can be represented in a graph form and also be augmented with the discussed methods, and training can be enhanced with the proposed transfer learning approach.

6.6 Conclusion and Future Work

This paper presents an approach for using augmentation to enhance DL models in the context of similarity learning between semantic processes. The approach is motivated by an analysis of existing literature on process augmentation from BPM research, which shows that the topic of augmentation is largely unexplored. The proposed augmentation methods are organized in three categories that represent their properties regarding complexity and knowledge-intensity and correctness and quality. Additionally, a semi-supervised transfer learning procedure is described to combine training with augmented and non-augmented data. The experimental evaluation compares graph embedding models that are trained on augmented data with models trained on non-augmented data. The results indicate a large potential of the augmented datasets but also strong variations, depending on the specific augmentation methods used and the size of the augmented datasets.

In future work, we aim to extend and refine the proposed approach. Due to our focus on the manufacturing domain in this work, it is not apparent how the approach performs for process models of other domains (see Sect. 6.5). In this regard, the applicability of the approach in the domain of event logs should also be examined. It could complement other literature, such as Käppel and Jablonski [29] or Grüger *et al.* [14]. In this context, we also want to investigate how *Large Language Models (LLMs)* can be combined and used with other AI methods such as automated planning [e. g., 16, 35, 67] to support process augmentation further. Furthermore, the usage of other GNNs than the GEM, e. g., the *Graph Matching Network (GMN)* [24, 26], is part of future work. This provides insights on the effects of the proposed augmentation methods for other DL models. In addition, these approaches can be used in combination with process model adaptation approaches [36, 37, 49, 76], as self-learning process adaptation requires large amounts of data. Finally, it is planned to apply the proposed approach in a greater comparison with related work on process matching, e. g., other process similarity measures [2, 8, 27, 73].

Acknowledgements

We thank our colleague Martin Kuhn for his valuable feedback and his help with selecting the statistical methods used in the paper.

Disclosure Statement

The authors report there are no competing interests to declare.

Data Availability Statement

The source code, the datasets, the trained models, and other supplementary resources are available online at https://gitlab.rlp.net/iot-lab-uni-trier/aai-journal-2024. The data was last accessed on August 8$^{\text{th}}$, 2024.

References

1. Adjeisah, M., Zhu, X., Xu, H., Ayall, T.A.: Towards data augmentation in graph neural network: An overview and evaluation. Comput. Sci. Rev. **47**, 100527 (2023). https://doi.org/10.1016/J.COSREV.2022.100527

2. Assy, N., van Dongen, B.F., van der Aalst, W.M.P.: Similarity resonance for improving process model matching accuracy. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018, pp. 86–93. ACM (2018). https://doi.org/10.1145/3167132.3167138

3. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Springer (2002)

4. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014). https://doi.org/10.1016/J.IS.2012.07.005

5. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Kapetanakis, S., Borck, H. (eds.) Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Reasoning co-located with the Twenty-seventh International Conference on Case-Based Reasoning (ICCBR 2019), Otzenhausen, Germany, September 8-12, 2019. CEUR Workshop Proceedings, pp. 156–161. CEUR-WS.org (2019)

6. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.E.: A Simple Framework for Contrastive Learning of Visual Representations. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event. Proceedings of Machine Learning Research, pp. 1597–1607. PMLR (2020)

7. Chen, Y., Xie, Y., Song, L., Chen, F., Tang, T.: A Survey of Accelerator Architectures for Deep Neural Networks. Engineering **6**(3), 264–274 (2020). https://doi.org/10.1016/j.eng.2020.01.007

8. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Inf. Syst. **36**(2), 498–516 (2011). https://doi.org/10.1016/J.IS.2010.09.006

9. Ding, K., Xu, Z., Tong, H., Liu, H.: Data Augmentation for Deep Graph Learning: A Survey. SIGKDD Explor. **24**(2), 61–77 (2022). https://doi.org/10.1145/3575637.3575646

10. van Dyk, D.A., Meng, X.-L.: The Art of Data Augmentation. Journal of Computational and Graphical Statistics **10**(1), 1–50 (2001)

11. Francescomarino, C.D., Ghidini, C.: Predictive Process Monitoring. In: Process Mining Handbook. Ed. by W.M.P. van der Aalst and J. Carmona, pp. 320–346. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_10

12. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016)

13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural Message Passing for Quantum Chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, pp. 1263–1272. PMLR (2017)

14. Grüger, J., Geyer, T., Jilg, D., Bergmann, R.: SAMPLE: A Semantic Approach for Multi-perspective Event Log Generation. In: Montali, M., Senderovich, A., Weidlich, M. (eds.) Process Mining Workshops - ICPM 2022 International Workshops, Bozen-Bolzano, Italy, October 23-28, 2022, Revised Selected Papers. LNBIP, vol. 468, pp. 328–340. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-27815-0_24

15. Grumbach, L., Bergmann, R.: Using Constraint Satisfaction Problem Solving to Enable Workflow Flexibility by Deviation (Best Technical Paper). In: Bramer, M., Petridis, M. (eds.) Artificial Intelligence XXXIV - 37th SGAI International Conference on Artificial Intelligence, AI 2017, Cambridge, UK, December 12-14, 2017, Proceedings. LNCS, vol. 10630, pp. 3–17. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-71078-5_1

16. Guan, L., Valmeekam, K., Sreedharan, S., Kambhampati, S.: Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. CoRR **abs/2305.14909** (2023). https://doi.org/10.48550/ARXIV.2305.14909. arXiv: 2305.14909

17. Haslum, P., Lipovetzky, N., Magazzeni, D., Muise, C.: An Introduction to the Planning Domain Definition Language. Morgan & Claypool Publishers (2019)

18. Helmert, M., Domshlak, C.: Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In: Gerevini, A., Howe, A.E., Cesta, A., Refanidis, I. (eds.) Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009. AAAI (2009)

19. Helmert, M., Geffner, H.: Unifying the Causal Graph and Additive Heuristics. In: Rintanen, J., Nebel, B., Beck, J.C., Hansen, E.A. (eds.) Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008, pp. 140–147. AAAI (2008)

20. Hermans, A., Beyer, L., Leibe, B.: In Defense of the Triplet Loss for Person Re-Identification. CoRR **abs/1703.07737** (2017). arXiv: 1703.07737

21. Hoffmann, J.: FF: The Fast-Forward Planning System. AI Mag. **22**(3), 57–62 (2001). https://doi.org/10.1609/AIMAG.V22I3.1572

22. Hoffmann, M., Bergmann, R.: Improving Automated Hyperparameter Optimization with Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 273–288. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_18

23. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

24. Hoffmann, M., Bergmann, R.: Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133039

25. Hoffmann, M., Malburg, L., Bach, N., Bergmann, R.: GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 240–255. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_16

26. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

27. Jabeen, F., Leopold, H., Reijers, H.A.: How to Make Process Model Matching Work Better? An Analysis of Current Similarity Measures. In: Abramowicz, W. (ed.) Business Information Systems - 20th International Conference, BIS 2017, Poznan, Poland, June 28-30, 2017, Proceedings. LNBIP, vol. 288, pp. 181–193. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-59336-4_13

28. Jilani, R.: "Automated Domain Model Learning Tools for Planning." In: Knowledge Engineering Tools and Techniques for AI Planning. Ed. by M. Vallati and D. Kitchin. Cham: Springer International Publishing, 2020, pp. 21–46. ISBN: 978-3-030-38561-3. https://doi.org/10.1007/978-3-030-38561-3_2.

29. Käppel, M., Jablonski, S.: Model-Agnostic Event Log Augmentation for Predictive Process Monitoring. In: Indulska, M., Reinhartz-Berger, I., Cetina, C., Pastor, O. (eds.) Advanced Information Systems Engineering - 35th International Conference, CAiSE 2023, Zaragoza, Spain, June 12-16, 2023, Proceedings. LNCS, vol. 13901, pp. 381–397. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-34560-9_23

30. Käppel, M., Schönig, S., Jablonski, S.: Leveraging Small Sample Learning for Business Process Management. Inf. Softw. Technol. **132**, 106472 (2021). https://doi.org/10.1016/J.INFSOF.2020.106472

31. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 188–203. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_13

32. Kudenko, D.: Special Issue on Transfer Learning. Künstliche Intelligenz **28**(1), 5–6 (2014). https://doi.org/10.1007/s13218-013-0289-5

33. Lenz, M., Ollinger, S., Sahitaj, P., Bergmann, R.: Semantic Textual Similarity Measures for Case-Based Retrieval of Argument Graphs. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 219–234. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_15

34. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf. Syst. **56**, 235–257 (2016). https://doi.org/10.1016/J.IS.2015.07.003

35. Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., Stone, P.: LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. CoRR **abs/2304.11477** (2023). https://doi.org/10.48550/ARXIV.2304.11477. arXiv: 2304.11477

36. Malburg, L., Brand, F., Bergmann, R.: Adaptive Management of Cyber-Physical Workflows by Means of Case-Based Reasoning and Automated Planning. In: Sales, T.P., Proper, H.A., Guizzardi, G., Montali, M., Maggi, F.M., Fonseca, C.M. (eds.) Enterprise Design, Operations, and Computing. EDOC 2022 Workshops - IDAMS, SoEA4EE, TEAR, EDOC Forum, Demonstrations Track and Doctoral Consortium, Bozen-Bolzano, Italy, October 4-7, 2022, Revised Selected Papers. LNBIP, vol. 466, pp. 79–95. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-26886-1_5

37. Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. **61**(1), 83–111 (2023). https://doi.org/10.1007/S10844-022-00766-W

38. Malburg, L., Klein, P., Bergmann, R.: Semantic Web Services for AI-Research with Physical Factory Simulation Models in Industry 4.0. In: Panetto, H., Madani, K., Smirnov, A.V. (eds.) Proceedings of the International Conference on Innovative Intelligent Industrial Production and Logistics, IN4PL 2020, Budapest, Hungary, November 2-4, 2020, pp. 32–43. SCITEPRESS (2020)

39. Malburg, L., Klein, P., Bergmann, R.: Converting semantic web services into formal planning domain descriptions to enable manufacturing process planning and scheduling in industry 4.0. Eng. Appl. Artif. Intell. **126**, 106727 (2023). https://doi.org/10.1016/J.ENGAPPAI.2023.106727

40. Malburg, L., Seiger, R., Bergmann, R., Weber, B.: Using Physical Factory Simulation Models for Business Process Management Research. In: del-Río-Ortega, A., Leopold, H., Santoro, F.M. (eds.) Business Process Management Workshops - BPM 2020 International Workshops, Seville, Spain, September 13-18, 2020, Revised Selected Papers. LNBIP, vol. 397, pp. 95–107. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-66498-5_8

41. Marrella, A.: Automated Planning for Business Process Management. J. Data Semant. **8**(2), 79–98 (2019). https://doi.org/10.1007/S13740-018-0096-0

42. Marrella, A., Mecella, M., Sardiña, S.: Intelligent Process Adaptation in the SmartPM System. ACM Trans. Intell. Syst. Technol. **8**(2), 25:1–25:43 (2017). https://doi.org/10.1145/2948071

43. Marrella, A., Mecella, M., Sardiña, S.: Supporting adaptiveness of cyber-physical processes through action-based formalisms. AI Commun. **31**(1), 47–74 (2018). https://doi.org/10.3233/AIC-170748

44. Marrium, M., Mahmood, A.: Data Augmentation for Graph Data: Recent Advancements. CoRR **abs/2208.11973** (2022). https://doi.org/10.48550/ARXIV.2208.11973. arXiv: 2208.11973

45. Masellis, R.D., Francescomarino, C.D., Ghidini, C., Tessaris, S.: Solving reachability problems on data-aware workflows. Expert Syst. Appl. **189**, 116059 (2022). https://doi.org/10.1016/J.ESWA.2021.116059

46. McCluskey, T.L., Cresswell, S., Richardson, N.E., West, M.M.: Automated Acquisition of Action Knowledge. In: Filipe, J., Fred, A.L.N., Sharp, B. (eds.) ICAART 2009 - Proceedings of the International Conference on Agents and Artificial Intelligence, Porto, Portugal, January 19 - 21, 2009, pp. 93–100. INSTICC Press (2009)

47. McDermott, D., Ghallab, M., Howe, A.E., Knoblock, C.A., Ram, A., Veloso, M.M., Weld, D.S., Wilkins, D.E.: PDDL - The Planning Domain Definition Language: Technical Report CVC TR-98-003/DCS TR-1165, (1998).

48. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. In: Bengio, Y., LeCun, Y. (eds.) 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013)

49. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer, Wiesbaden (2018)

50. Mumuni, A., Mumuni, F.: Data augmentation: A comprehensive survey of modern approaches. Array **16**, 100258 (2022). https://doi.org/10.1016/J.ARRAY.2022.100258

51. Nguyen, T., Sreedharan, S., Kambhampati, S.: Robust planning with incomplete domain models. Artif. Intell. **245**, 134–161 (2017). https://doi.org/10.1016/J.ARTINT.2016.12.003

52. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Mach. Learn. **107**(11), 1875–1893 (2018). https://doi.org/10.1007/S10994-018-5702-8

53. Ontañón, S.: An Overview of Distance and Similarity Functions for Structured Data. CoRR **abs/2002.07420** (2020). arXiv: 2002.07420

54. Ott, F., Rügamer, D., Heublein, L., Bischl, B., Mutschler, C.: Cross-Modal Common Representation Learning with Triplet Loss Functions. CoRR **abs/2202.07901** (2022). arXiv: 2202.07901

55. Pfeiffer, P., Lahann, J., Fettke, P.: Multivariate Business Process Representation Learning Utilizing Gramian Angular Fields and Convolutional Neural Networks. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings. LNCS, vol. 12875, pp. 327–344. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-85469-0_21

56. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep Learning for Predictive Business Process Monitoring: Review and Benchmark. IEEE Trans. Serv. Comput. **16**(1), 739–756 (2023). https://doi.org/10.1109/TSC.2021.3139807

57. Richter, M.M.: Foundations of Similarity and Utility. In: Wilson, D., Sutcliffe, G. (eds.) Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7-9, 2007, Key West, Florida, USA, pp. 30–37. AAAI Press (2007)

58. Rodríguez-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating planning and scheduling in workflow domains. Expert Syst. Appl. **33**(2), 389–406 (2007). https://doi.org/10.1016/J.ESWA.2006.05.027

59. von Rüden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Pfrommer, J., Pick, A., Ramamurthy, R., Walczak, M., Garcke, J., Bauckhage, C., Schuecker, J.: Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. IEEE Trans. Knowl. Data Eng. **35**(1), 614–633 (2023). https://doi.org/10.1109/TKDE.2021.3079836

60. Schoknecht, A., Thaler, T., Fettke, P., Oberweis, A., Laue, R.: Similarity of Business Process Models - A State-of-the-Art Analysis. ACM Comput. Surv. **50**(4), 52:1–52:33 (2017). https://doi.org/10.1145/3092694

61. Schroff, F., Kalenichenko, D., Philbin, J.: FaceNet: A unified embedding for face recognition and clustering. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pp. 815–823. IEEE Computer Society (2015). https://doi.org/10.1109/CVPR.2015.7298682

62. Schuler, N., Hoffmann, M., Beise, H., Bergmann, R.: Semi-supervised Similarity Learning in Process-Oriented Case-Based Reasoning. In: Bramer, M., Stahl, F.T. (eds.) Artificial Intelligence XL - 43rd SGAI International Conference on Artificial Intelligence, AI 2023, Cambridge, UK, December 12-14, 2023, Proceedings. LNCS, vol. 14381, pp. 159–173. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-47994-6_12

63. Seiger, R., Huber, S., Heisig, P., Aßmann, U.: Toward a framework for self-adaptive workflows in cyber-physical systems. Softw. Syst. Model. **18**(2), 1117–1134 (2019). https://doi.org/10.1007/S10270-017-0639-0

64. Seiger, R., Malburg, L., Weber, B., Bergmann, R.: Integrating process management and event processing in smart factories: A systems architecture and use cases. Journal of Manufacturing Systems **63**, 575–592 (2022). https://doi.org/10.1016/j.jmsy.2022.05.012

65. Shu, J., Xu, Z., Meng, D.: Small Sample Learning in Big Data Era. CoRR **abs/1808.04572** (2018). arXiv: 1808.04572

66. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A Survey on Deep Transfer Learning. In: Kurková, V., Manolopoulos, Y., Hammer, B., Iliadis, L.S., Maglogiannis, I. (eds.) Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III. LNCS, vol. 11141, pp. 270–279. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01424-7_27

67. Valmeekam, K., Sreedharan, S., Marquez, M., Hernandez, A.O., Kambhampati, S.: On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark). CoRR **abs/2302.06706** (2023). https://doi.org/10.48550/ARXIV.2302.06706. arXiv: 2302.06706

68. Venkateswaran, P., Muthusamy, V., Isahagian, V., Venkatasubramanian, N.: Robust and Generalizable Predictive Models for Business Processes. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings. LNCS, vol. 12875, pp. 105–122. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-85469-0_9

69. Weidlich, M., Dijkman, R.M., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13094-6_37

70. Weiss, K.R., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. J. Big Data **3**, 9 (2016). https://doi.org/10.1186/S40537-016-0043-6

71. Wickler, G., Chrpa, L., McCluskey, T.L.: Ontological Support for Modelling Planning Knowledge. In: Fred, A.L.N., Dietz, J.L.G., Aveiro, D., Liu, K., Filipe, J. (eds.) Knowledge Discovery, Knowledge Engineering and Knowledge Management - 6th International Joint Conference, IC3K 2014, Rome, Italy, October 21-24, 2014, Revised Selected Papers. Communications in Computer and Information Science, pp. 293–312. Springer (2014). https://doi.org/10.1007/978-3-319-25840-9_19

72. Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P.: Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. CoRR **abs/2004.09602** (2020). arXiv: 2004.09602

73. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I. LNCS, vol. 6426, pp. 60–77. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16934-2_8

74. Yu, S., Huang, H., Dao, M.N., Xia, F.: Graph Augmentation Learning. In: Laforest, F., Troncy, R., Simperl, E., Agarwal, D., Gionis, A., Herman, I., Médini, L. (eds.) Companion of The Web Conference 2022, Virtual Event / Lyon, France, April 25 - 29, 2022, pp. 1063–1072. ACM (2022). https://doi.org/10.1145/3487553.3524718

75. Zeyen, C., Bergmann, R.: A*-Based Similarity Assessment of Semantic Graphs. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 17–32. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_2

76. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 388–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_26

77. Zhao, T., Jin, W., Liu, Y., Wang, Y., Liu, G., Günnemann, S., Shah, N., Jiang, M.: Graph Data Augmentation for Graph Machine Learning: A Survey. IEEE Data Eng. Bull. **46**(2), 140–165 (2023)

78. Zhou, J., Xie, C., Wen, Z., Zhao, X., Xuan, Q.: Data Augmentation on Graphs: A Technical Survey. CoRR **abs/2212.09970** (2023). arXiv: 2212.09970

79. Zhou, Y., Zeng, J.: Massively Parallel A* Search on a GPU. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, pp. 1248–1255. AAAI Press (2015). https://doi.org/10.1609/AAAI.V29I1.9367

80. Zhuo, H.H., Muñoz-Avila, H., Yang, Q.: Learning hierarchical task network domains from partially observed plan traces. Artif. Intell. **212**, 134–157 (2014). https://doi.org/10.1016/J.ARTINT.2014.04.003

6.7 Appendix: Statistics

Domain	Dataset	Nodes									Edges								
		All			Task			Data			All			Control-Flow			Dataflow		
		Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
50	Original	25.7	18.0	39.0	11.6	8.0	17.0	12.2	9.0	18.0	59.3	40.0	94.0	11.9	7.0	24.0	22.7	16.0	34.0
	Strat. 1	25.4	17.0	39.0	11.6	8.0	17.0	12.0	8.0	18.0	58.1	37.0	94.0	11.4	5.0	24.0	22.2	14.0	34.0
	Strat. 2	25.7	18.0	39.0	11.6	8.0	17.0	12.2	9.0	18.0	59.3	40.0	94.0	11.9	7.0	24.0	22.7	16.0	34.0
	Strat. 3.1	25.6	12.0	39.0	11.7	5.0	18.0	12.3	6.0	18.0	58.9	25.0	94.0	11.6	4.0	24.0	22.7	10.0	34.0
	Strat. 3.2	30.0	18.0	58.0	14.3	8.0	27.0	14.1	9.0	24.0	69.5	40.0	140.0	14.2	7.0	35.0	26.3	16.0	48.0
	Strat. 4	25.5	17.0	39.0	11.6	8.0	17.0	12.1	8.0	18.0	58.6	37.0	94.0	11.6	5.0	24.0	22.4	14.0	34.0
	Strat. 5.1	25.5	12.0	39.0	11.7	5.0	18.0	12.1	6.0	18.0	58.6	25.0	94.0	11.6	4.0	24.0	22.5	10.0	34.0
	Strat. 5.2	27.3	17.0	58.0	12.7	8.0	27.0	12.8	8.0	24.0	62.9	37.0	140.0	12.6	5.0	35.0	24.0	14.0	48.0
100	Original	26.2	18.0	39.0	11.8	8.0	18.0	12.4	9.0	19.0	60.6	40.0	94.0	12.4	7.0	24.0	23.0	16.0	36.0
	Strat. 1	26.0	17.0	39.0	11.8	8.0	18.0	12.1	8.0	19.0	59.4	37.0	94.0	11.9	5.0	24.0	22.6	14.0	36.0
	Strat. 2	26.2	18.0	39.0	11.8	8.0	18.0	12.4	9.0	19.0	60.6	40.0	94.0	12.4	7.0	24.0	23.0	16.0	36.0
	Strat. 3.1	26.2	12.0	39.0	11.9	5.0	18.0	12.4	6.0	19.0	60.4	25.0	94.0	12.1	4.0	24.0	23.1	10.0	36.0
	Strat. 3.2	31.0	18.0	60.0	14.7	8.0	27.0	14.5	9.0	26.0	72.1	40.0	146.0	14.9	7.0	35.0	27.2	16.0	52.0
	Strat. 4	26.1	17.0	39.0	11.8	8.0	18.0	12.2	8.0	19.0	59.9	37.0	94.0	12.1	5.0	24.0	22.8	14.0	36.0
	Strat. 5.1	26.1	12.0	39.0	11.9	5.0	18.0	12.3	6.0	19.0	60.0	25.0	94.0	12.1	4.0	24.0	22.9	10.0	36.0
	Strat. 5.2	28.0	17.0	60.0	13.0	8.0	27.0	13.1	8.0	26.0	64.7	37.0	146.0	13.2	5.0	35.0	24.5	14.0	52.0
200	Original	26.0	15.0	43.0	11.8	7.0	18.0	12.3	7.0	19.0	60.1	32.0	104.0	12.3	6.0	26.0	22.9	12.0	36.0
	Strat. 1	25.8	14.0	43.0	11.8	7.0	18.0	12.0	6.0	19.0	58.9	29.0	104.0	11.8	4.0	26.0	22.4	10.0	36.0
	Strat. 2	26.0	15.0	43.0	11.8	7.0	18.0	12.3	7.0	19.0	60.1	32.0	104.0	12.3	6.0	26.0	22.9	12.0	36.0
	Strat. 3.1	26.3	12.0	43.0	12.0	5.0	18.0	12.5	6.0	19.0	60.8	25.0	104.0	12.2	4.0	26.0	23.3	10.0	36.0
	Strat. 3.2	30.8	15.0	67.0	14.6	7.0	31.0	14.4	7.0	30.0	71.6	32.0	163.0	14.8	6.0	39.0	27.1	12.0	58.0
	Strat. 4	25.9	14.0	43.0	11.8	7.0	18.0	12.1	6.0	19.0	59.5	29.0	104.0	12.0	4.0	26.0	22.6	10.0	36.0
	Strat. 5.1	26.0	12.0	43.0	11.9	5.0	18.0	12.3	6.0	19.0	59.9	25.0	104.0	12.0	4.0	26.0	22.9	10.0	36.0
	Strat. 5.2	27.8	14.0	67.0	12.9	7.0	31.0	13.0	6.0	30.0	64.2	29.0	163.0	13.1	4.0	39.0	24.4	10.0	58.0
300	Original	26.1	15.0	43.0	11.8	7.0	18.0	12.4	7.0	19.0	60.3	32.0	104.0	12.2	6.0	26.0	23.0	12.0	36.0
	Strat. 1	25.8	14.0	43.0	11.8	7.0	18.0	12.1	6.0	19.0	59.1	29.0	104.0	11.7	4.0	26.0	22.6	10.0	36.0
	Strat. 2	26.1	15.0	43.0	11.8	7.0	18.0	12.4	7.0	19.0	60.3	32.0	104.0	12.2	6.0	26.0	23.0	12.0	36.0
	Strat. 3.1	26.3	12.0	43.0	12.0	5.0	18.0	12.5	6.0	19.0	60.9	25.0	104.0	12.2	4.0	26.0	23.3	10.0	36.0
	Strat. 3.2	31.2	15.0	67.0	14.9	7.0	34.0	14.6	7.0	32.0	72.7	32.0	163.0	15.0	6.0	39.0	27.5	12.0	62.0
	Strat. 4	25.9	14.0	43.0	11.8	7.0	18.0	12.2	6.0	19.0	59.6	29.0	104.0	11.9	4.0	26.0	22.8	10.0	36.0
	Strat. 5.1	26.1	12.0	43.0	11.9	5.0	18.0	12.3	6.0	19.0	60.0	25.0	104.0	12.0	4.0	26.0	22.9	10.0	36.0
	Strat. 5.2	28.0	14.0	67.0	13.1	7.0	34.0	13.2	6.0	32.0	64.7	29.0	163.0	13.1	4.0	39.0	24.6	10.0	62.0

Table 6.3: Statistics on Amounts of Nodes and Edges of the Datasets in the Experiments.

Domain	Attribute	Compared Strategy						
		1	2	3.1	3.2	4	5.1	5.2
50	No. Nodes (avg.)	0.970	1.000	0.940	0.725	0.983	0.976	0.896
	No. Task Nodes (avg.)	1.000	1.000	0.940	0.000	1.000	0.976	0.008
	No. Data Nodes (avg.)	0.945	1.000	0.955	0.020	0.969	0.970	0.020
	No. Edges (avg.)	0.920	1.000	0.940	0.125	0.954	0.966	0.050
	No. Control-Flow Edges (avg.)	0.925	1.000	0.940	0.140	0.957	0.968	0.140
	No. Dataflow Edges (avg.)	0.950	1.000	0.955	0.585	0.971	0.978	0.698
100	No. Nodes (avg.)	0.970	1.000	0.943	0.745	0.983	0.968	0.899
	No. Task Nodes (avg.)	1.000	1.000	0.943	0.010	1.000	0.977	0.017
	No. Data Nodes (avg.)	0.943	1.000	0.958	0.020	0.967	0.970	0.020
	No. Edges (avg.)	0.935	1.000	0.940	0.153	0.963	0.952	0.061
	No. Control-Flow Edges (avg.)	0.920	1.000	0.943	0.170	0.954	0.966	0.177
	No. Dataflow Edges (avg.)	0.955	1.000	0.958	0.565	0.974	0.972	0.703
200	No. Nodes (avg.)	0.978	1.000	0.936	0.738	0.987	0.979	0.895
	No. Task Nodes (avg.)	1.000	1.000	0.936	0.014	1.000	0.975	0.023
	No. Data Nodes (avg.)	0.946	1.000	0.941	0.019	0.969	0.983	0.024
	No. Edges (avg.)	0.938	1.000	0.936	0.144	0.964	0.970	0.064
	No. Control-Flow Edges (avg.)	0.923	1.000	0.936	0.169	0.956	0.977	0.174
	No. Dataflow Edges (avg.)	0.958	1.000	0.944	0.556	0.976	0.978	0.706
300	No. Nodes (avg.)	0.975	1.000	0.941	0.715	0.986	0.982	0.887
	No. Task Nodes (avg.)	1.000	1.000	0.941	0.018	1.000	0.976	0.027
	No. Data Nodes (avg.)	0.950	1.000	0.948	0.025	0.971	0.982	0.028
	No. Edges (avg.)	0.933	1.000	0.941	0.163	0.961	0.979	0.071
	No. Control-Flow Edges (avg.)	0.923	1.000	0.941	0.152	0.956	0.977	0.161
	No. Dataflow Edges (avg.)	0.958	1.000	0.950	0.550	0.976	0.984	0.697

Table 6.4: Kolmogorov-Smirnov-Statistics of the Average Number of Nodes and Edges of the Datasets in the Experiments.

Chapter 7

Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning Using Graph Neural Networks and Transfer Learning

Bibliographic Information

Pauli, J., Hoffmann, M., Bergmann, R.: Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning Using Graph Neural Networks and Transfer Learning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133040

Abstract

Similarity-based retrieval of semantic graphs is a crucial task of *Process-Oriented Case-Based Reasoning (POCBR)* that is usually complex and time-consuming, as it requires some kind of inexact graph matching. Previous work tackles this problem by using *Graph Neural Networks (GNNs)* to learn pairwise graph similarities. In this paper, we present a novel approach that improves on the GNN-based case retrieval with a *Transfer Learning (TL)* setup, composed of two phases: First, the *pretraining* phase trains a model for assessing the similarities between graph nodes and edges and their semantic annotations. Second, the pretrained model is then integrated into the GNN model by either using *fine-tuning*, i. e., the parameters of the pretrained model are further trained, or *feature extraction*, i. e., the parameters of the pretrained model are converted to constants. The experimental evaluation examines the quality and performance of the models based on TL compared to the GNN models from previous work for three semantic graph domains with various properties. The results show the great potential of the proposed approach for reducing the similarity prediction error and the training time.

Keywords

Case-Based Reasoning · Deep Learning · Process-Oriented Case-Based Reasoning · Transfer Learning

Copyright Notice

7.1 Introduction

Case-based reasoning (CBR) [1] is a problem-solving paradigm that solves a new problem (*query*) by using specific knowledge from previously experienced, concrete problem situations (bundled as *cases* in a *case base*). This procedure relies heavily on similarity computations between the query and each case of the case base to find best-matching cases for the current problem situation (so-called *case retrieval*), based on the assumption that similar problems have similar solutions. This paper focuses on the subdomain of *Process-Oriented CBR (POCBR)* [5, 27] where the CBR methodology is applied to procedural knowledge in the form of workflows and process descriptions. The cases in POCBR are commonly represented as semantic graphs [5] with a relational structure of nodes and edges of different types and with individual semantic annotations. When computing pairwise similarities between semantic graphs, a single *global similarity* is computed from individual *local similarities* between nodes and edges of the graph and their types and semantic annotations. The similarity measures to compute local similarities usually involve manually-modeled similarity knowledge such as data types, value ranges, and ontologies, and use functions such as weighted sums or set overlaps. The global similarity measures, on the other hand, usually involve some kind of inexact graph matching to determine how well the nodes and edges of the query graph fit the nodes and edges of the case graph. However, finding an optimal mapping is an NP-complete problem [7, 12, 35, 36], which can result in long retrieval times and have negative effects on the overall system performance and, ultimately, on the user experience.

To mitigate these issues, several approaches have been proposed [6, 15, 29], with the most recent one by Hoffmann and Bergmann [11] using embedding methods based on *Graph Neural Networks (GNNs).* GNNs transform semantic graphs to low-dimensional vector representations (so-called *embeddings*) that can be transformed to graph similarities with vector similarity measures or *Multi-Layer Perceptrons (MLPs).* The models are trained on the prediction error between the predicted graph similarities and the labeled ground-truth similarities, aiming to approximate the manually-modeled ground-truth graph similarity measure based on graph matching. However, the used GNNs are only trained to predict the labeled global graph similarity without any explicit consideration of local similarities, despite this information being available as a result of the labeling process anyway. This can be problematic, as current work is not capable of training or applying local models separate from the global model. Enabling this might increase *flexibility*, as local models and global models can be used in different application scenarios or a single local model can be used in multiple global models. It might also improve *performance*, because the local similarity labels can be used to reduce the prediction error and the training time.

This paper aims to address these issues by using *Transfer Learning (TL)* [18, 32] in a two-phased approach: In the *pretraining* phase, a model is trained to predict the local similarities between nodes and edges, and in the *adaptation* phase[1], the pretrained model is integrated into the training process of the graph embedding model to predict global pairwise graph similarities. Therefore, the currently used graph embedding models [11] are broken up into two models, one

[1] Please note that the term "adaptation" refers to its meaning in the context of TL in the remainder of the paper and is not referring to the reuse phase in CBR.

for embedding nodes and edges and for computing local similarities, and another one for embedding graphs and for computing graph similarities. The goal is to improve the currently used GNNs [11] and ultimately increase the similarity prediction quality, reduce the overall training time, and improve the flexibility of the training process. The paper is organized as follows: Section 7.2 describes foundations on the used semantic graph representation, the graph similarity computation as well as current graph embedding models and related work. Furthermore, our approach to use TL for improving semantic graph embedding and similarity prediction is presented in Sect. 7.3. Section 7.4 evaluates the approach and compares it against existing models. Finally, Section 5 summarizes the work and identifies areas for future work.

7.2 Foundations and Related Work

The foundations include the semantic workflow representation that is used in the concept and the experiments, as well as the similarity assessment between pairs of these workflows. Further, work on approximating pairwise graph similarities with GNNs and related work is presented.

7.2.1 Semantic Workflow Representation

We represent all workflows as semantically annotated directed graphs referred to as *NEST* graphs, introduced by Bergmann and Gil [5]. More specifically, a *NEST* graph is a quadruple $W = (N,E,S,T)$ that is composed of a set of nodes N and a set of edges $E \subseteq N \times N$. Each node and each edge has a specific type from Ω that is indicated by the function $T : N \cup E \rightarrow \Omega$. Additionally, the function $S : N \cup E \rightarrow \Sigma$ assigns a semantic description from Σ (*semantic metadata language*, e. g., an ontology) to nodes and edges. Whereas nodes and edges are used to build the structure of each workflow, types and semantic descriptions are additionally used to model semantic information. Figure 7.1 shows an exemplary *NEST* graph that represents a sandwich recipe. The mayo-gouda sandwich is prepared by executing the cooking steps `coat` and `layer` (task nodes) with the ingredients `mayo`, `baguette`, `sandwich dish`, and `gouda` (data nodes). All components are linked by edges that indicate relations, e. g., `mayo` is consumed by `coat`. Semantic descriptions of task nodes and data nodes are used to further specify semantic information belonging to the workflow components, e. g., the semantic description of the task node `coat` states that a spoon and a baguette knife are needed to execute the task (`Auxiliaries`) and the estimated time that the task takes is two minutes (`Duration`).

7.2.2 Similarity Assessment

Determining the similarity between two *NEST* graphs, i. e., a query workflow QG and a case workflow CG, requires a similarity measure that assesses the link structure of nodes and edges as well as their semantic descriptions and types. Bergmann and Gil [5] propose a semantic similarity measure that determines a global similarity between two graphs based on local similarities, i. e., the pairwise similarities of nodes and edges. The similarity between two nodes with identical types is defined as the similarity of the semantic descriptions of these nodes. The similarity between two edges with identical types does not only consider the similarity of the semantic descriptions of the edges, but in addition, the similarity of the connected nodes as well. In order to put together a global similarity by aggregating local similarities, the domain's similarity model has to define similarity measures for all components of the semantic description, i. e., $sim_\Sigma : \Sigma \times \Sigma \rightarrow [0,1]$. The global similarity of the two workflows $sim(QG, CG)$ is finally calculated by finding an injective partial mapping m that maximizes $sim_m(QG, CG)$.

$$sim(QG, CG) = max\{sim_m(QG, CG) \mid \text{admissible mapping } m\} \tag{7.1}$$

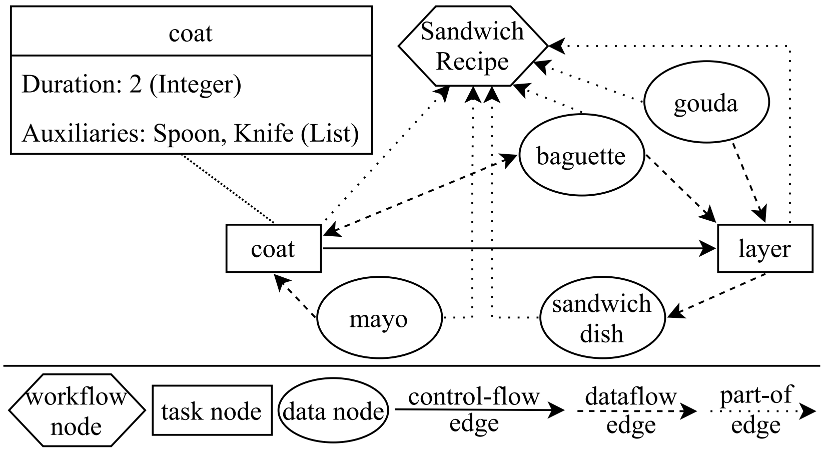


Figure 7.1: Exemplary Cooking Recipe represented as a NEST Graph

A mapping is admissible if all mapped nodes as well as all mapped edges with their source and destination nodes are of the same type. The complex process of finding a mapping that maximizes the global similarity between a query QG and a single case CG is tackled by utilizing an A^* search algorithm (see [5] for more details). However, A* search is usually time-consuming and can lead to long retrieval times [12, 36] which motivates the use of automatic learning methods such as GNNs [11].

7.2.3 Neural Networks for Semantic Graph Embedding

For the purpose of similarity-based retrieval in POCBR, Hoffmann and Bergmann [11] adapt two Siamese GNNs, namely the *Graph Embedding Model (GEM)* and the *Graph Matching Network (GMN)*, which are shown in Fig. 7.2. Both models learn to predict pairwise graph similarities by embedding the graph structure and the semantic annotations and types to a whole-graph vector representation, before aggregating two of these vectors to compute a single similarity

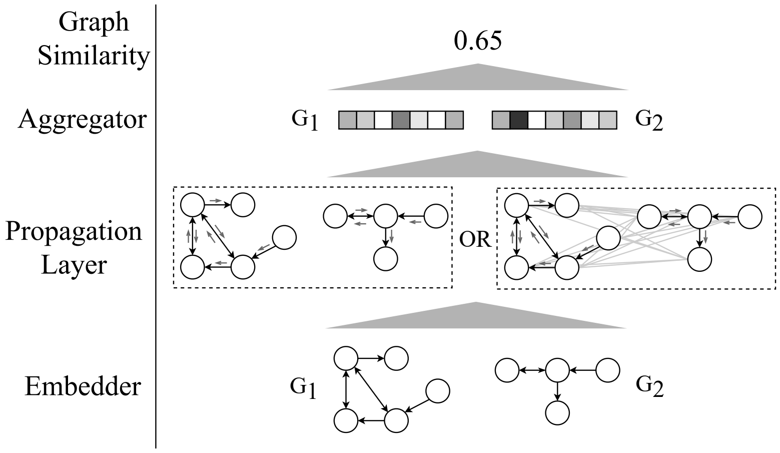


Figure 7.2: GEM (left branch) and GMN (right branch) (taken from Hoffmann and Bergmann [11])

value. Thereby, both models share the same general model architecture that consists of four parts: The *embedder* maps the node and edge features to the initial node and edge vectors in the vector space. The *propagation layer* then accumulates information for each node in its local neighborhood via message passing. More specifically, the vector representation of a single node is updated by merging the vector representation of this node with the vector representations of the neighboring nodes, connected by an incoming edge. After multiple propagation iterations, the *aggregator* converts the set of node representations into a whole-graph representation. The *graph similarity* of two graphs can finally be computed based on their representations in the vector space, e. g., with the help of a vector similarity measure such as cosine similarity. Both models differ in the concrete realization of the propagation layer and the graph similarity. These varieties induce a trade-off between expressiveness and performance, with the GMN being more expressive than the GEM but also computationally more expensive (see Hoffmann and Bergmann [11] for more details). A general shortcoming of both models, however, is the missing explicit integration of local similarity knowledge, motivating this work for using GEM and GMN in a TL setup to improve similarity predictions.

7.2.4 Related Work

We consider approaches as related that deal with TL in the environment of CBR and specifically POCBR, or apply Deep Learning (DL) methods in the research field of CBR. Approaches using TL in CBR are of limited number. The work of Klenk, Aha, and Molineaux [16] discusses the perspective that solution reuse as part of the CBR cycle can be considered as TL, since it involves the transfer of knowledge from previous cases to new problems. In this context, other methods for case reuse, e. g., [28], can be seen as contributing to TL in this respect. Herold and Minor [10] investigate the feasibility of using ontologies for TL in POCBR. They try to find an automated ontology-based learning approach for knowledge transfer between two domains, while earlier work [26] is based on the manual creation of ontologies. There are also approaches, e. g., [2], where CBR methods are used to support or improve TL methods, which we want to mention for the sake of completeness.

A greater research interest can be observed for the use of DL methods in CBR [19]. Mathisen, Bach, and Aamodt [25] and Amin *et al.* [3] utilize Siamese neural networks in CBR retrieval to learn similarity measures used in aquaculture and natural language processing tasks, respectively. Leake, Ye, and Crandall [21], Liao, Liu, and Chao [22], and Ye, Leake, and Crandall [34] discuss the application of DL methods in the reuse phase of CBR. Leake, Wilkerson, and Crandall [20] use neural networks to learn features in CBR tasks. In a more narrow sense, regarding the integration of DL methods in the retrieval of POCBR, the work of Klein, Malburg, and Bergmann [15] and the work of Hoffmann and Bergmann [11], on which the present work builds, are related. They have in common that they learn vector representations of semantic graphs and evaluate the learned similarity function in a retrieval scenario. The approach of Klein, Malburg, and Bergmann [15] only learns on structural properties of semantic graphs, while Hoffmann and Bergmann [11] additionally consider the semantic information and types of nodes and edges. The proposed approach differs from the aforementioned works in the sense that it is the first work to combine the above topics. In particular, TL is used to improve the generalization capability of the DL models developed by Hoffmann and Bergmann [11] to improve on the task of similarity-based retrieval.

7.3 Transfer Learning for Improving Semantic Graph Embedding Models

The global similarity between two semantic graphs can be determined from the local similarities, that is, their node and edge similarities. However, the computation of graph similarities using GEM or GMN lacks the opportunity to take this teaching input into account, possibly reducing the prediction quality. The approach proposed in this paper aims at integrating local similarity knowledge into GEM and GMN by utilizing a *network-based deep TL* strategy [18, 30, 32]. Thereby, a neural network trained on the *source task*, i. e., to predict local similarities, is transferred and adapted to the *target task*, i. e., to predict pairwise graph similarities with GEM and GMN, to aim at improving prediction quality of the GNNs. While pairwise graph similarity prediction consists of a single step with only one training procedure, our approach is composed of two separate phases, i. e., *pretraining* and *adaptation*, that both perform a training step (see Fig. 7.3). The knowledge transfer is conducted in the adaptation phase, where the neural network, which is pretrained to predict local similarities, becomes a part of GEM or GMN in the target domain [31]. In this scenario, we extract the embedder from GEM and GMN and then pretrain it to predict local similarities (see Fig. 7.2). The embedder is well-suited, as it is the only part of the GNN architecture that processes semantic descriptions and types of nodes and edges, which are used to compute the local similarities (see Sect. 7.2.2). As already mentioned in the motivation, training the embedder individually has three key advantages: First, it can be used standalone to predict local similarities in various scenarios, increasing flexibility. For instance, it can be used with other similarity measures such as feature-based approaches [6] or to predict local similarities in a matching algorithm [5]. Second, the trained embedder can be reused for multiple model instances of GEM or GMN, thus, reducing the training effort for these models. Third, the overall precision of the similarity predictions can be improved, as the embedder is directly trained with labeled local similarities, instead of being indirectly trained as part of the whole GNN via the labeled global similarity.

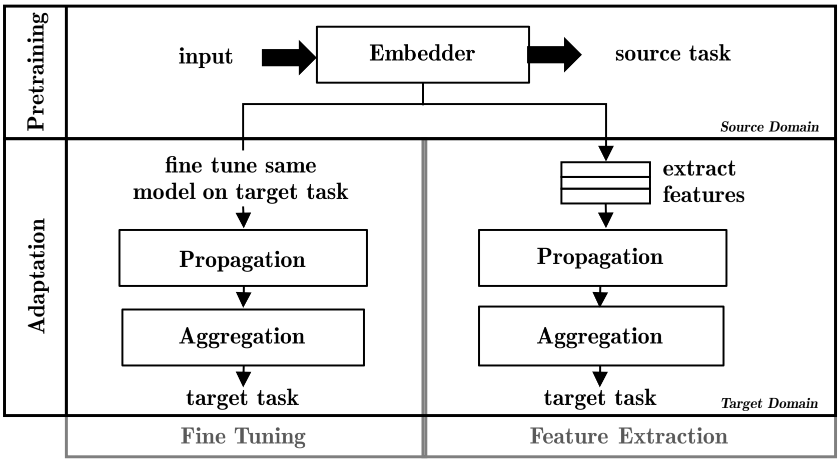


Figure 7.3: Pretraining and Adaptation Phases of Deep Transfer Similarity Learning.

7.3.1 Pretraining

In the pretraining phase, the embedder is trained to predict local similarities of nodes and edges by, first, embedding their semantic descriptions and types and, second, aggregating the embedding vectors to pairwise similarities. The encoding methods of the semantic descriptions and types can be reused from GEM and GMN (see Hoffmann and Bergmann [11] for more details). We propose to train two separate embedders, one for nodes and one for edges, since their semantic descriptions may be structured differently.

Analogous to GEM and GMN, the pretrained embedder is trained as a Siamese neural network, which is the de-facto standard in metric learning tasks (see Chicco [8] for an overview). This means that local similarities are learned by embedding two nodes or edges with identical networks and aggregating the two embedding vectors to a single similarity value. The parameters are shared between the two networks and trained by minimizing a loss function, which measures the difference between the ground-truth local similarity and the predicted similarity in the vector space. The embedding procedure can be performed with any kind of neural network architecture and mainly depends on the domain of the semantic descriptions. In previous work [11], embedders for sequence and graph-structured data in the form of *Recurrent Neural Networks (RNNs)* and GNNs are presented, which are both applicable in this scenario. The final step of computing a local similarity between the embedding vectors is conducted by an MLP, as also used in the GMN, since it is expressive and easy to compute.

The training examples for pretraining are composed of pairs of semantic descriptions as well as types of nodes and edges. Each training pair is thereby labeled with the ground-truth similarity value in the range of [0,1], which is determined using local similarity measures (see Sect. 7.2.2). Please note that these local similarities are already available from the process of generating labeled data for GEM and GMN and, thus, result in no additional cost. The training is a regression-like problem, where any standard regression loss function such as the *Mean Squared Error (MSE)* can be used. To optimize the parameters of the neural network according to the MSE loss value, any optimizer using stochastic gradient descent, e. g., the Adam optimizer [14], can be used.

7.3.2 Adaptation

In the adaptation phase of the TL approach, the pretrained embedder is integrated into GEM and GMN by replacing the standard, untrained embedder (see Sect. 7.2.3). This adaptation to the task of the target domain is mainly implemented in two ways in the literature [31], either by *feature extraction* or by *fine-tuning.* In feature extraction, similar to classical feature-based approaches [17], the trainable parameters of the pretrained embedder are converted to constants ("frozen"), and the model will not be further trained in the training procedure of GEM and GMN. Alternatively in a fine-tuning setup, the parameters of the pretrained embedder are used as non-random initializations [9] and are further trained, i. e., fine-tuned, during the training procedure of GEM and GMN. On the one hand, the general advantage of using feature extraction over fine-tuning is the reduced training effort of the graph embedding models, as the "frozen" weights and biases do not have to be trained. On the other hand, there can be further potential for improvement of prediction quality when fine-tuning the pretrained embedders w. r. t. the source task. Either way, the adaptation is expected to reduce the training time due to a non-random initialization of the neural network parameters. Since there are many factors that affect the choice of the adaption method (see Peters, Ruder, and Smith [31] for more information), both methods are further examined and evaluated in the remainder of this paper.

7.4 Experimental Evaluation

To evaluate the impact of integrating local similarities into GEM and GMN, the results of the model training with and without the use of TL are compared. Therefore, we conduct an experiment that consists of two parts: The first part is concerned with pretraining different embedder variants to find the best-performing one. The best-performing variant is then used in the second phase of the experiment, where it is adapted to GEM and GMN via TL. We examine the quality of the predictions in terms of the MSE between the predicted similarities and the ground-truth similarities. Furthermore, the training time is measured to quantify the impact of TL on the performance of the training procedure, which is important across all DL scenarios. The following hypotheses are investigated:

H1 The integration of local similarities into GEM and GMN based on TL generally improves the prediction quality of the similarities.

H2 The integration of local similarities into GEM and GMN based on TL generally reduces the training time.

7.4.1 Experimental Setup

The experiments are performed with three case bases from different domains, namely workflows of a cooking domain CB-I [13], a data mining domain CB-II [37], and a manufacturing domain CB-III [24]. CB-I consists of 40 manually modeled cooking recipes that are extended to 800 workflows by generalization and specialization of ingredients and cooking steps (see Müller [28] for more details), resulting in 660 training cases, 60 validation cases, and 80 test cases. The workflows of the data mining domain (CB-II) are built from sample processes that are delivered with RapidMiner (see Zeyen, Malburg, and Bergmann [37] for more details), resulting in 509 training cases, 40 validation cases, and 60 test cases. CB-III contains workflows that stem from a smart manufacturing IoT environment and represent sample production processes. The case base consists of 75 training cases, nine validation cases, and nine test cases.

While the GNNs in the adaption phase use pairs of semantic graphs and their respective ground-truth similarity as training input, the embedders are pretrained based on the semantic annotations and the types of the nodes[2] from each graph. The examples of the pretraining phase sum up to 14259 training cases, 1439 test cases, and 1219 validation cases for CB-I, 8539 training cases, 1032 test cases, and 670 validation cases for CB-II, and 2136 training cases, 242 test cases, and 265 validation cases for CB-III. The number of graph pairs and node pairs used as examples for training, testing, and validation is exactly the square of the respective numbers of graphs and nodes given before, as there is a ground-truth similarity value calculated for each possible graph and node pair, respectively.

The training examples are used as training input for the neural networks, while the validation examples are used to monitor the training process. The models using TL use the same hyperparameter settings as the respective base models. The training of all models and for both phases, i. e., pretraining and adaption, runs for 40 epochs, and a snapshot of the model is exported after each epoch. The models with the lowest validation error within these 40 exported models are used as reference. The MSE determines the out-of-sample prediction accuracy of the GNNs by computing the squared difference between the predicted values of the GNNs and the actual similarity values for all pairs of graphs in the test data set. In addition, the training time is determined as the time interval between the start of training and the time when the validation loss permanently falls below a certain threshold ϵ. Measuring this time indicates how well a model converges, and enables a comparison regarding training effort between different models.

We evaluate three different model architectures for the embedders, mainly based on related work [11]: (1) A *Recurrent Neural Network (RNN)* processes the encoded node information as sequence data. (2) An *Attention Neural Network (ANN)* [33] that uses attention mechanisms

[2] We do not use the edges for pretraining, since they are not semantically annotated in the evaluated domains. The approach, however, supports doing this.

to give more weight to certain elements of the node information than others (see Bahdanau, Cho, and Bengio [4] and Vaswani *et al.* [33] for more details on attention). (3) A GNN that processes the node information in a tree-based structure. All experiments are computed on a single NVIDIA Tesla V100-SXM2 GPU with 32 GB graphics RAM.

7.4.2 Experimental Results

Table 7.1 shows the validation loss w. r. t. the MSE of all evaluated, pretrained embedders for all three case bases. The embedders with the lowest MSE values are highlighted in bold font and are used within the adaptation phase for the respective domain. For both CB-I and CB-III, the best MSE can be observed by the RNN while the GNN achieves the lowest MSE value for CB-II. This shows variations between the embedder architectures for the different domains, which is in-line with other experimental results, e. g., [11].

Table 7.1: Evaluation Results of the Pretraining Phase.

	RNN	ANN	GNN
CB-I	**0.0004**	0.0215	0.0080
CB-II	0.0197	0.0645	**0.0170**
CB-III	**0.0045**	0.0183	0.0078

Table 7.2 shows the results of the experiments for the adaptation phase. The models using TL are labeled with superscripts indicating the methods used to adapt the pretrained embedder to the downstream task, namely feature extraction ($^{\text{FE}}$) and fine-tuning ($^{\text{FT}}$). The table shows the loss on the test data w. r. t. the MSE and the time span (in seconds) to fall below a certain threshold ϵ for all evaluated models and for all three case bases. The value of ϵ is set to be the lowest validation loss that is reached by all compared models of the same domain. The models are grouped according to their underlying model architecture. This results in a different threshold for each model architecture and all domains. The lowest MSE and time span values are highlighted in bold font, grouped by base model and domain.

Table 7.2: Evaluation Results of the Adaptation Phase.

		GEM			GMN		
		GEM	GEM$^{\text{FE}}$	GEM$^{\text{FT}}$	GMN	GMN$^{\text{FE}}$	GMN$^{\text{FT}}$
CB-I			$\epsilon = 0.022$			$\epsilon = 0.006$	
	Training Time (s)	13273	**12070**	15078	123432	**59523**	137524
	MSE	0.0767	0.0712	**0.0661**	0.0028	**0.0017**	0.0024
CB-II			$\epsilon = 0.084$			$\epsilon = 0.008$	
	Training Time (s)	8970	**6707**	8306	81724	**24012**	73960
	MSE	0.0882	0.0841	**0.0709**	0.0068	**0.0054**	0.0063
CB-III			$\epsilon = 0.215$			$\epsilon = 0.033$	
	Training Time (s)	260	171	**47**	1452	**598**	745
	MSE	0.2035	0.1867	**0.1838**	0.0237	0.0206	**0.0193**

The qualities w. r. t. MSE show a significant influence of TL on the base models. Both for GEM and GMN, the models using TL consistently outperform the base models, regardless of the domain. Thereby, the two adaptation methods feature extraction and fine-tuning show different effects for the three case bases. The use of fine-tuning in GEMs leads to an MSE decrease between 10 % and 20 %, while feature extraction decreases the MSE between 5 % and 8 %. The results of the GMN variants are also promising: GMN^{FE} has the best overall MSE values for CB-I and CB-II with an MSE decrease over the base model of about 39 % and 21 %, respectively. GMN^{FT}

decreases the MSE by about 14 % for CB-I and 7 % for CB-II. In terms of CB-III, fine-tuning outperforms feature extraction with a 5 % lower MSE.

The training times also show promising results of the TL models. Thereby, feature extraction reduces the training time independent of the model architecture in all domains. GEM^{FE}, for instance, undercuts the threshold ϵ between 9 % and 34 % faster than the base models. GMN^{FE} follows this trend and undercuts ϵ between 52 % and 71 % faster than the base models. Fine-tuning has a positive effect on training time only in CB-II and CB-III, whereas in CB-I it has a negative effect. GEM^{FT} shows a 7 % and 82 % and GMN^{FT} a 10 % and 49 % decrease in training time compared to the base models for CB-II and CB-III, respectively. Hence, for GMN, feature extraction outperforms fine-tuning in terms of training time, while for GEM this is only true for CB-I and CB-II.

In conclusion, the experiments indicate the potential of the proposed approach for reducing the prediction error and the training time. The improved prediction error is likely caused by the explicit use of additional label local similarities that are not used by the base models. In most cases, fine tuning also outperforms feature extraction which suggests that GEM and GMN can benefit from the additional training of parameters in the adaptation phase. Hypothesis H1 is therefore accepted due to a positive effect of TL across all evaluated domains and base models. The results regarding training time are overall very positive, except for GEM$^{\mathrm{FT}}$ and GMN$^{\mathrm{FT}}$ of CB-I. It is not clear why these two cases lead to worse results when using TL. As a rule of thumb, feature extraction should be preferred for minimizing training time. Thus, we only partly accept H2 but report a high potential for a reduction of training time.

7.5 Conclusion and Future Work

This paper explores the potential of using TL to predict GNN-based similarities of semantic graphs. The proposed approach splits two existing GNN architectures, that is the GEM and the GMN, into two parts that are trained individually to predict local and global similarities. The resulting models can be used in a similarity-based retrieval, in the context of POCBR, to predict the similarities of pairs of semantic graphs. The models with and without TL are evaluated, focusing on changes in retrieval quality and training time. The experimental results indicate that the integration of local similarities can improve the similarity evaluation quality of both models and reduce training times.

The evaluated implementation of the proposed approach pretrains separate models for each domain, which can lead to a significant training effort for multi-domain setups or when frequently changing the domain definition. Future work could investigate performing GNN model pretraining in the sense of unsupervised learning on a large collection of unlabeled graph data, deriving generic, transferable, and domain-independent knowledge that encodes intrinsic graph properties. Further, analogous to Lu *et al.* [23], one can argue that there is a divergence between the two steps of pretraining and adaptation due to different optimization goals targeting the source domain and the target domain, respectively. The pretraining process might currently ignore the need to quickly adapt to the downstream task with a few fine-tuning updates, which could be further analyzed in future work. Additionally, future approaches could consider both node and edge-level and graph-level tasks for pretraining since solely pretraining on nodes and edges may be suboptimal for graph-level similarity assessment, as performed with GEM and GMN, where capturing structural similarities of local neighborhoods may be more important than capturing pairwise local similarities. In addition, a focus of future work should be to perform a more comprehensive evaluation of the proposed approach, and particularly the different embedder variants, to measure the impact for a higher number of domains and DL setups.

Acknowledgements

This work is funded by the German Federal Ministry for Economic Affairs and Climate Action under grant No. 22973 *SPELL*.

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994). https://doi.org/10.3233/AIC-1994-7104
2. Aha, D.W., Molineaux, M., Sukthankar, G.: Case-Based Reasoning in Transfer Learning. In: McGinty, L., Wilson, D.C. (eds.) Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009, Seattle, WA, USA, July 20-23, 2009, Proceedings. LNCS, vol. 5650, pp. 29–44. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02998-1_4
3. Amin, K., Kapetanakis, S., Polatidis, N., Althoff, K., Dengel, A.: DeepKAF: A Heterogeneous CBR & Deep Learning Approach for NLP Prototyping. In: Ivanovic, M., Yildirim, T., Trajcevski, G., Badica, C., Bellatreche, L., Kotenko, I.V., Badica, A., Erkmen, B., Savic, M. (eds.) International Conference on INnovations in Intelligent SysTems and Applications, INISTA 2020, Novi Sad, Serbia, August 24-26, 2020, pp. 1–7. IEEE (2020). https://doi.org/10.1109/INISTA49547.2020.9194679
4. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
5. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014). https://doi.org/10.1016/J.IS.2012.07.005
6. Bergmann, R., Stromer, A.: MAC/FAC Retrieval of Semantic Workflows. In: Boonthum-Denecke, C., Youngblood, G.M. (eds.) Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida, USA, May 22-24, 2013. AAAI Press (2013)
7. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognit. Lett. **19**(3-4), 255–259 (1998). https://doi.org/10.1016/S0167-8655(97)00179-7
8. Chicco, D.: Siamese Neural Networks: An Overview. In: Artificial Neural Networks - Third Edition. Ed. by H.M. Cartwright, pp. 73–94. Springer (2021). https://doi.org/10.1007/978-1-0716-0826-5_3
9. Dai, A.M., Le, Q.V.: Semi-supervised Sequence Learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pp. 3079–3087 (2015)
10. Herold, M., Minor, M.: Ontology-based Representation of Workflows for Transfer Learning. In: Gemulla, R., Ponzetto, S.P., Bizer, C., Keuper, M., Stuckenschmidt, H. (eds.) Proceedings of the Conference ”Lernen, Wissen, Daten, Analysen”, LWDA 2018, Mannheim, Germany, August 22-24, 2018. CEUR Workshop Proceedings, pp. 139–149. CEUR-WS.org (2018)

11. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

12. Hoffmann, M., Malburg, L., Bach, N., Bergmann, R.: GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 240–255. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_16

13. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

14. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

15. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 188–203. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_13

16. Klenk, M., Aha, D.W., Molineaux, M.: The Case for Case-Based Transfer Learning. AI Mag. **32**(1), 54–69 (2011). https://doi.org/10.1609/AIMAG.V32I1.2331

17. Koehn, P., Och, F.J., Marcu, D.: Statistical Phrase-Based Translation. In: Hearst, M.A., Ostendorf, M. (eds.) Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003. The Association for Computational Linguistics (2003)

18. Kudenko, D.: Special Issue on Transfer Learning. Künstliche Intelligenz **28**(1), 5–6 (2014). https://doi.org/10.1007/s13218-013-0289-5

19. Leake, D., Crandall, D.J.: On Bringing Case-Based Reasoning Methodology to Deep Learning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 343–348. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_22

20. Leake, D., Wilkerson, Z., Crandall, D.: Extracting Case Indices from Convolutional Neural Networks: A Comparative Study. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 81–95. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_6

21. Leake, D., Ye, X., Crandall, D.J.: Supporting Case-Based Reasoning with Neural Networks: An Illustration for Case Adaptation. In: Martin, A., Hinkelmann, K., Fill, H., Gerber, A., Lenat, D., Stolle, R., van Harmelen, F. (eds.) Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021. CEUR Workshop Proceedings. CEUR-WS.org (2021)

22. Liao, C., Liu, A., Chao, Y.: A Machine Learning Approach to Case Adaptation. In: First IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018, Laguna Hills, CA, USA, September 26-28, 2018, pp. 106–109. IEEE Computer Society (2018). https://doi.org/10.1109/AIKE.2018.00023

23. Lu, Y., Jiang, X., Fang, Y., Shi, C.: Learning to Pre-train Graph Neural Networks. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pp. 4276–4284. AAAI Press (2021). https://doi.org/10.1609/AAAI.V35I5.16552

24. Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. **61**(1), 83–111 (2023). https://doi.org/10.1007/S10844-022-00766-W

25. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. **51**(11), 8107–8118 (2021). https://doi.org/10.1007/S10489-021-02251-3

26. Minor, M., Bergmann, R., Müller, J., Spät, A.: On the Transferability of Process-Oriented Cases. In: Goel, A.K., Díaz-Agudo, M.B., Roth-Berghofer, T. (eds.) Case-Based Reasoning Research and Development - 24th International Conference, ICCBR 2016, Atlanta, GA, USA, October 31 - November 2, 2016, Proceedings. LNCS, vol. 9969, pp. 281–294. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-47096-2_19

27. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014). https://doi.org/10.1016/J.IS.2013.06.004

28. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer, Wiesbaden (2018)

29. Müller, G., Bergmann, R.: A Cluster-Based Approach to Improve Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, pp. 639–644. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-639

30. Pan, S.J., Yang, Q.: A Survey on Transfer Learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010). https://doi.org/10.1109/TKDE.2009.191

31. Peters, M.E., Ruder, S., Smith, N.A.: To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks. In: Augenstein, I., Gella, S., Ruder, S., Kann, K., Can, B., Welbl, J., Conneau, A., Ren, X., Rei, M. (eds.) Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019, pp. 7–14. Association for Computational Linguistics (2019). https://doi.org/10.18653/V1/W19-4302

32. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A Survey on Deep Transfer Learning. In: Kurková, V., Manolopoulos, Y., Hammer, B., Iliadis, L.S., Maglogiannis, I. (eds.) Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III. LNCS, vol. 11141, pp. 270–279. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01424-7_27

33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)

34. Ye, X., Leake, D., Crandall, D.: Case Adaptation with Neural Networks: Capabilities and Limitations. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 143–158. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_10

35. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing Stars: On Approximating Graph Edit Distance. Proc. VLDB Endow. **2**(1), 25–36 (2009). https://doi.org/10.14778/1687627.1687631

36. Zeyen, C., Bergmann, R.: A*-Based Similarity Assessment of Semantic Graphs. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 17–32. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_2

37. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 388–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_26

Chapter 8

Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning

Bibliographic Information

Hoffmann, M., Bergmann, R.: Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133039

Abstract

In *Process-Oriented Case-Based Reasoning (POCBR)*, experiential knowledge from previous problem-solving situations is retrieved from a case base to be reused for upcoming problems. The task of retrieval is approached in previous work by using *Graph Neural Networks (GNNs)* to learn workflow similarities which are, in turn, used to find similar workflows w.r.t. a query workflow. This paper is motivated by the fact that these GNNs are mostly used for predicting the similarity between two workflows (query and case), while the retrieval in CBR is only concerned with the ranking of the most similar workflows from the case base w.r.t. the query. Thus, we propose a novel approach to extend the GNN-based workflow retrieval by a *Learning-to-Rank (LTR)* component where rankings instead of similarities between cases are predicted. The main contribution of this paper addresses the changes to the GNNs from previous work, such that their model architecture predicts pairwise preferences between cases w.r.t. a query and that they can be trained using labeled preference data. In order to transform these preferences into a case ranking, we also describe *rank aggregation* methods with different levels of computational complexity. The experimental evaluation compares different models for predicting similarities and rankings in case retrieval scenarios. The results indicate the potential of our ranking-based approach in significantly improving retrieval quality with only small impacts on the performance.

Keywords

Case-Based Reasoning · Deep Learning · Process-Oriented Case-Based Reasoning · Learning-to-Rank

Copyright Notice

8.1 Introduction

Hybrid approaches of *Case-Based Reasoning (CBR)* [1] and *Deep Learning (DL)* are becoming more and more popular across many CBR research groups, e. g., [3, 14, 29]. The common goal is to support or implement different parts of CBR applications with appropriate DL methods [20]. Thereby, most of these efforts are focused on *case retrieval*, i. e., a case base of best-practice problem-solution-pairs is queried to find solution candidates for a given problem based on the similarity between the new problem and past problems. Supporting this task with DL methods is commonly approached by automatically learning similarity measures, for instance, to predict the similarity between the query and cases of the case base [3, 21, 25]. In this process, the role of the similarity measure is to control the retrieval process by declaring the most similar cases of the case base as the most useful solution candidates to the query. This leaves the concrete similarity values mainly as a criterion for *case ranking*. Especially when using DL-based similarity measures, predicting similarities instead of rankings might lead to imprecise retrieval results, since small similarity prediction errors can have great effects on the final ranking of the cases. A more intuitive approach in this scenario would be to bring the model predictions closer to the underlying task by predicting *pairwise preferences* such as "*case A is more relevant to the query than case B*". This reformulation has two advantages: First, the number of training examples increases as each pair of cases from the case base combined with a query case is one training example. When predicting similarities, on the other hand, a training example consists of the query and a single case, which effectively squares the number of training examples when training with pairwise preferences. Second, it redefines the learning task as a binary classification which utilizes different metrics, loss functions, and training parameters compared to a regression. This enables a new way of training the models, which might lead to more accurate predictions or shorter training times.

The described technique is part of the *Learning to Rank (LTR)* methodology [23] that is also known as preference learning [11]. LTR deals with ranking items according to a given query by means of machine learning methods. It covers crucial aspects of the ranking procedure, such as suitable loss functions and training procedures. The use of LTR methods in CBR is not thoroughly investigated, but its applicability is indicated by the shared goal of ranking cases according to a query [6] as well as the way experts think about cases which much more resembles preferences than similarity values [18]. Therefore, we investigate the potential of LTR for similarity-based retrieval in this paper. We use the context of *Process-Oriented Case-Based Reasoning (POCBR)* [26]. POCBR is a subfield of CBR that deals with procedural knowledge, such as semantic graphs [5]. We discuss existing DL-based methods w. r. t. their contribution to LTR and integrate new LTR principles into the *Graph Neural Networks (GNNs)* from literature [14, 16] with the goal of improving the quality and performance of these models for the task of case retrieval in POCBR. The remainder of the paper is structured as follows: Section 8.2 shows foundations of the used semantic graph representation and gives an introduction to LTR before presenting related work. Further, Sect. 8.3 presents the concept for integrating LTR principles into GNNs for similarity assessment in POCBR. Section 8.4 presents the experimental evaluation before Sect. 8.5 concludes the paper and shows directions of future work.

8.2 Foundations and Related Work

The foundations include the semantic workflow representation that is used in the concept and the experiment evaluation, as well as the similarity assessment between pairs of these workflows. Additionally, the basic strategies of LTR are introduced, and related work is examined.

8.2.1 Semantic Workflow Representation and Similarity Assessment

We represent all workflows as semantically annotated directed graphs referred to as *NEST* graphs, introduced by Bergmann and Gil [5]. A *NEST* graph is defined as a quadruple $W = (N,E,S,T)$ that is composed of a set of nodes N and a set of edges $E \subseteq N \times N$. Each node and each edge has a specific type from Ω that is indicated by the function $T : N \cup E \rightarrow \Omega$. Additionally, the function $S : N \cup E \rightarrow \Sigma$ assigns a semantic description from Σ (*semantic metadata language*, e. g., an ontology) to nodes and edges. Whereas nodes and edges represent the structure of each workflow, types and semantic descriptions model additional semantic information. Figure 8.1 shows an exemplary *NEST* graph that represents a cooking recipe of a mayo-gouda sandwich. The sandwich is prepared by executing the cooking steps `coat` and `layer` (represented as task nodes) with the ingredients `mayo`, `baguette`, `sandwich dish`, and `gouda` (represented as data nodes). All components are connected by edges indicating relations, e. g., `coat` consumes `mayo`. Semantic information belonging to the workflow components is further specified by using semantic annotations for nodes and edges. For instance, the semantic description of the task node `coat` provides the information that a spoon and a baguette knife are needed to execute the task and that the estimated time is two minutes.

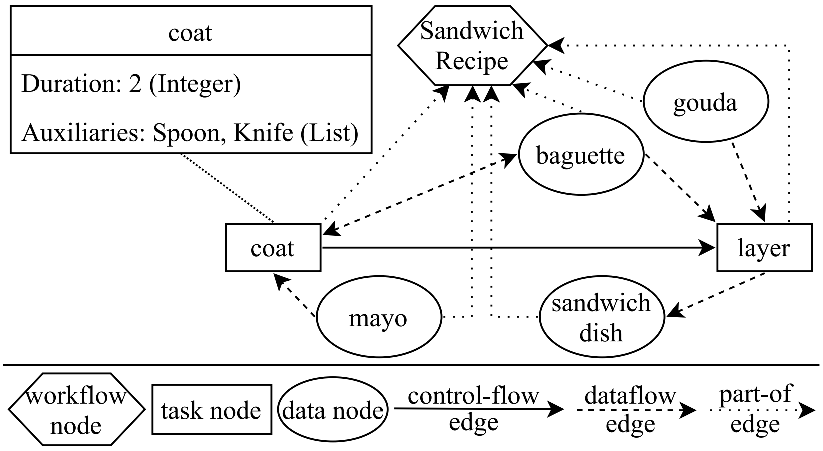


Figure 8.1: Exemplary Cooking Recipe represented as NEST Graph

Since NEST graphs are used in the context of POCBR, Bergmann and Gil [5] also propose a semantic similarity measure between two *NEST* graphs, i. e., a query workflow QG and a case workflow CG. The similarity is determined based on the local-global principle such that a global similarity, i. e., the similarity between two graphs, is composed of local similarities,

i. e., the pairwise similarities of nodes and edges. Nodes with the same type can be mapped onto each other, and their similarity corresponds to the similarity of their semantic descriptions. The similarity between two edges with identical types does not only consider the similarity of the semantic descriptions of the edges, but in addition, the similarity of the connected nodes, as well. A mapping is, hence, admissible if all mapped nodes as well as all mapped edges with their source and destination nodes are of the same type. When assuming that the domain defines a similarity model with a similarity measure $sim_\Sigma : \Sigma \times \Sigma \rightarrow [0,1]$ for all components of the semantic description, the global similarity of the two workflows $sim(QG, CG)$ is calculated by finding an injective partial mapping m that maximizes $sim_m(QG, CG)$.

$$sim(QG, CG) = max\{sim_m(QG, CG) \mid \text{admissible mapping } m\}$$

The complex process of finding a mapping that maximizes the global similarity between a query QG and a single case CG is tackled by an A* search algorithm (see Bergmann and Gil [5] for more details). However, A* search is usually time-consuming and can lead to long retrieval times [15, 19, 30] which motivates using automatic learning methods for the task of case retrieval.

8.2.2 Learning to Rank

In LTR, a ranked list of items is determined based on the relevance of these items w. r. t. a query [23]. A well-known application scenario is information retrieval, where the most relevant documents regarding some query are ranked and returned to a user. LTR approaches are commonly divided into three categories based on the way rankings are determined, i. e., pointwise, pairwise, listwise LTR [23]: *Pointwise LTR* approaches use existing metrics to compute a relevance score for each item. For instance, retrieving similar cases based on a query in a case retrieval in CBR could be associated to this category. These methods have the advantage that they can be developed and implemented rather quickly due to the metrics already existing, but the quality might not be sufficient, since the used metrics were probably not intended to be used in an LTR context. Furthermore, *pairwise LTR* or *label ranking* [11] approaches determine the ranked list of items by evaluating preferences between pairs of items w. r. t. the query. A preference in this sense states whether one or the other item of a pair is more relevant to the query. To get a ranked list of items, *rank aggregation* is used and transforms the pairwise preferences. Pairwise methods are, compared to pointwise approaches, harder to conceptualize and implement due to the two involved components, but could probably lead to better overall results due to their focus on predicting actual rankings. Although the proposed approach only covers pointwise and pairwise methods, *listwise LTR*, also known as *object ranking* [11], is still explained for the sake of completeness. This category of approaches is the most straightforward, as it returns a complete ranked list of items according to the query. There is no need to derive the ranked list of items from different metrics as in the other categories. Thus, listwise LTR methods promise better results than other methods due to their direct prediction of ranked lists but, in turn, are also generally harder to implement and train.

8.2.3 Related Work

Related work in the context of CBR or POCBR dealing with rankings or preferences explicitly is rather scarce. One of the earliest approaches in this regard is proposed by Avesani, Ferrari, and Susi [4]. They describe an interactive approach where users set preferences to help a system configure a model to answer queries based on the user's preferences. Brinker and Hüllermeier [7] discuss pairwise ranking in the CBR context conceptually. They mainly focus on implementing a case retrieval when given preferences as labels. Hüllermeier and Schlegel [18] extend this concept and propose a methodological framework for problem-solving in CBR with preferences. It describes the representation of experiential knowledge in the form of pairwise preferences between cases and case retrieval in this setup. While the aforementioned approaches are primarily concerned with a conceptualization of rankings in the CBR methodology, there are also approaches that use rankings in more application-oriented scenarios. For instance, Li and Sun [22] use rankings and CBR to compute similarities between cases in finance applications. Glöckner and Weis [12] determine case rankings with a combination of machine learning and CBR for question answering. Bichindaritz [6] combines CBR and information retrieval for scenarios from biomedicine. Our approach is novel regarding related work as it is a combination of the POCBR context, domain independence, and the use of DL methods to predict preferences.

8.3 Similarity-Based Retrieval by Using Learning to Rank Methods

Case retrieval with GNNs in POCBR applications returns the most similar cases according to the similarity predictions of the used neural networks. We propose an alternative approach to case retrieval that predicts preferences of cases with GNNs to form the retrieval result. The following sections, first, place GNN-based retrieval in the context of pointwise LTR and, second, describe the approach of retrieval by predicting pairwise case preferences.

8.3.1 Pointwise Ranking with GNNs

For the purpose of similarity-based retrieval in POCBR, Hoffmann *et al.* [16] adapt two Siamese GNNs which are called *Graph Embedding Model (GEM)* and the *Graph Matching Network (GMN)* (see Fig. 8.2). Both models learn embeddings of semantic graphs by processing the nodes and the edges between those nodes, incl. their types and semantic annotations. These embeddings are then further aggregated to a single pairwise similarity value by using cosine vector similarity or a *Multi-Layer Perceptron (MLP)*. Thereby, both models have the same fundamental architecture that consists of four parts that the data passes through in succession: the *embedder*, the *propagation layer*, the *aggregator*, and the *graph similarity*. The embedder maps the types

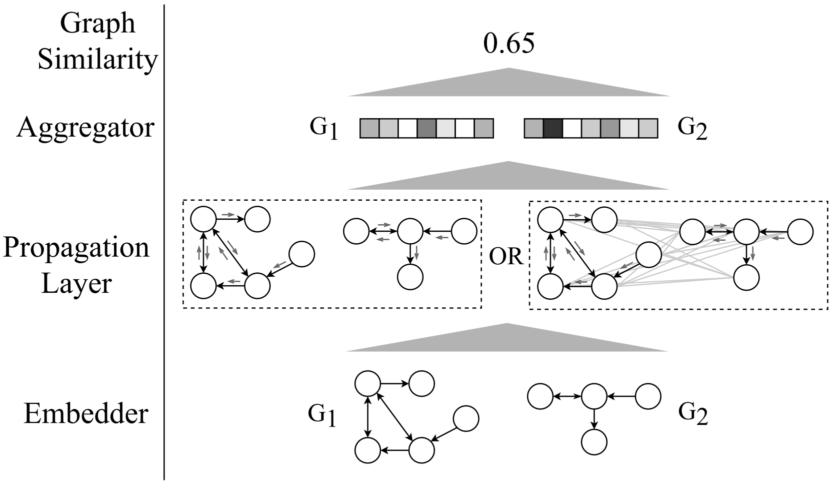


Figure 8.2: GEM (left branch) and GMN (right branch) (taken from Hoffmann and Bergmann [14])

and semantic annotations of nodes and edges to their initial representations in the vector space. The propagation layer then propagates information about each node in its local neighborhood via message passing, where the vector representation of a single node is updated by merging it with the vector representations of all nodes that are connected by an incoming edge. The GEM computes the graph representations for each graph independently, and the node-level information is propagated exclusively in the respective graph. In contrast, the GMN uses a cross-graph matching mechanism that propagates information between two graphs based on node attentions. After multiple propagation iterations, the aggregator joins representations of all nodes to a single graph representation. The graph similarity of two graphs can finally be computed with their respective representations in the vector space, by using a cosine vector similarity in GEM and an MLP in GMN. The architectural differences between GEM and GMN induce a trade-off between expressiveness and performance, with the GMN being more expressive than the GEM but also computationally more expensive.

According to the categorization in Sect. 8.2.2, this approach is a pointwise ranking measure. The most relevant cases of the case base w. r. t. a query are retrieved by predicting pairwise similarities with GEM or GMN and sorting the cases according to their similarity. Thus, the GNN-based similarity measure can be used interchangeably with other similarity measures in POCBR applications. However, it might be too strict in retrieval-only applications, where only the case ranks are of interest and not the concrete similarity values.

8.3.2 Predicting Preferences with GNNs

In order to realize a case retrieval, we propose to adjust the similarity-predicting GNNs to predict pairwise preferences of case pairs. Let $G_q, G_1, G_2 \in \mathbb{G}$ be a query and two semantic case graphs from a graph space $\mathbb{G}$, we want to define a function $p : ((\mathbb{G},\mathbb{G}),(\mathbb{G},\mathbb{G})) \longrightarrow [0,1]$ that determines a probabilistic preference between the pairs (G_q, G_1) and (G_q, G_2). A result of $p((G_q, G_1),(G_q, G_2)) \geq 0.5$ means that G_1 is more relevant to the query than G_2, and vice versa. The architectural changes to use GEM and GMN in this context address the part about the graph similarity that is replaced with a preference function. The embedder, the propagation layer, and the aggregator can be reused to embed the graphs into a latent space vector representation. Let GNN : $(\mathbb{G},\mathbb{G}) \longrightarrow (\mathbb{R}^n, \mathbb{R}^n)$ be a function that uses either GEM or GMN to embed two graphs to their n-dimensional vector representations, $(v_q, v_1) = \text{GNN}(G_q, G_1)$, and $(v_q, v_2) = \text{GNN}(G_q, G_2)$, then $p((G_q, G_1),(G_q, G_2))$ is defined as follows:

$$p((G_q, G_1),(G_q, G_2)) = \sigma(\text{MLP}_p([v_q - v_1, v_q - v_2]))$$

The pairwise preference between these two graph pairs is computed by element-wise subtracting the vectors v_1 and v_2 from v_q, concatenating the results, processing the results by MLP_p, and activating the final value with a sigmoid activation. MLP_p can be an arbitrary MLP with the constraint that it has to have a single output neuron in the last layer in order to have a single preference value. Training GEM and GMN in this setup also differs from the original goal of predicting similarities. Instead of training on batches of graph pairs, the models are trained on batches of pairs of graph pairs, as function p also suggests. Instead of computing a *Mean Squared Error (MSE)* loss between predicted and label similarity, the predicted preferences are evaluated by computing a binary cross-entropy loss between the predictions and the labels. Please note that the proposed pairwise ranking approach can also be directly trained on label preferences between cases, for instance determined by direct user feedback. This is not possible with the pointwise models, but useful in the real world, since it is more natural for human experts to determine a preference between two alternatives than a similarity value [18].

Algorithm 1 Greedy Rank Aggregation

Data: case base *CB*; query workflow *QG*; GNN-based preference function $p(\cdot,\cdot)$
Result: ranked list of cases

```
 1: $V \longleftarrow CB$;    $result \longleftarrow$ empty list
 2: for each $v \in V$ do
 3:     $s_1(v) \longleftarrow \sum_{u \in V} p(v,u) - \sum_{u \in V} p((QG,v),(QG,u))$
 4: end for
 5: while $|V| > 0$ do
 6:     $v_{\text{pot}} \longleftarrow argmax_{u \in V}\, s_1(u)$
 7:     add $v_{\text{pot}}$ to the end of $result$
 8:     $V = V \setminus \{v_{\text{pot}}\}$
 9:     for each $v \in V$ do
10:         $s_1(v) \longleftarrow s_1(v) + p((QG,v_{\text{pot}}),(QG,v)) - p(v,v_{\text{pot}})$
11:     end for
12: end while
13: return $result$
```

In order to transform the predicted pairwise preferences to actual ranks which can, in turn, be used as the retrieval result, it is necessary to perform *rank aggregation* (as introduced in Sect. 8.2.2). There are overviews of rank aggregators available in the literature [2, 17], where most of the methods can be used in this approach. For our use case, we want to present a slightly modified variant of the approach by Cohen, Schapire, and Singer [10] (see Alg. 1) and another algorithm that is simpler and faster to compute (see Alg. 2).

Algorithm 1 is provided with the case base *CB*, the query *QG*, and the preference function $p(\cdot,\cdot)$ introduced before. The rank aggregation starts by creating a temporary set of cases V, where each case is initialized with its *potential ranking score*, denoted by the function $s_1(\cdot)$ (see lines 2 – 4). The value of this function for any case v is initialized as the number of comparisons where v is preferred over some other case u, minus the number of opposed comparisons. The result of the algorithm is a ranked result list of the cases in descending order of relevance (see lines 5 – 12). The result list is put together in a loop, where the case v_{pot} with the highest value of $s_1(v)$ is extracted from V and added to the end of the list. Afterward, the values of function s_1 are updated for each case in V according to the comparisons with the extracted case v_{pot}. The algorithm resembles a greedy search through the graph of pairwise preferences. It has a quadratic computational complexity and is therefore faster than optimal search algorithms such as A*. Cohen, Schapire, and Singer [10] also prove that the algorithm's agreement is at least half the agreement of the optimal ranking order.

Algorithm 2 is derived from Alg. 1 and simplifies two aspects to reduce computation time. First, the computation of the potential ranking score function $s_2(v)$ only includes the preferences where v is preferred over another case u. The opposed preference relations where u is preferred over another case v are not counted, hence the naming as *positive* rank aggregation. Second, the creation of the result list is also done in a loop, but the values of s_2 are not continuously updated. This means that the preferences of cases extracted from V are not removed from the scores of the cases still in V. This rank aggregation algorithm also has a quadratic computational complexity, but requires less computation steps than the greedy one. The performance of both methods is evaluated in the experimental evaluation.

Algorithm 2 Positive Rank Aggregation

Data: case base *CB*; query workflow *QG*; GNN-based preference function $p(\cdot,\cdot)$
Result: ranked list of cases

```
$V \longleftarrow CB$;    $result \longleftarrow$ empty list
for each $v \in V$ do
    $s_2(v) \longleftarrow \sum_{u \in V} p((QG, v), (QG, u))$
end for
while $|V| > 0$ do
    $v_{\text{pot}} \longleftarrow argmax_{u \in V}\, s_2(u)$
    add $v_{\text{pot}}$ to the end of $result$
    $V = V \setminus \{v_{\text{pot}}\}$
end while
return $result$
```

8.4 Experimental Evaluation

We conduct retrievals to compare pairwise ranking and pointwise ranking with GEM and GMN (see Sect. 8.3.1). All different GNN variants also include a version with sequence embedding and one with tree embedding to examine the effects of these semantic annotation embedding techniques, introduced by Hoffmann and Bergmann [14]. The models based on pairwise ranking also compare the results of the different rank aggregators, i. e., the greedy and the positive approach. This gives a total of twelve evaluated retrievers (see Tab. 8.1 for the entire enumeration). We measure the quality of several retrieval runs by comparing the order of the results, as predicted by the pointwise and pairwise ranking models, with the ground-truth ordering. In addition, we look at a comparison between the different retrievers regarding retrieval time. The following hypotheses are investigated:

H1 The retrievers based on pairwise ranking generally outperform the retrievers based on pointwise ranking w. r. t. the retrieval quality.

H2 The retrievers based on pairwise ranking generally show reduced retrieval times compared to the retrievers based on pointwise ranking.

8.4.1 Experimental Setup

The experiments are performed with three case bases from different domains, that is of a recipe domain CB-I [16], a data mining domain CB-II [31], and a manufacturing domain CB-III [24]. CB-I consists of 40 manually modeled cooking recipes that are extended to 800 workflows by adaptation techniques described in Müller [27], resulting in 660 training cases, 60 validation cases, and 80 test cases. The workflows of the data mining domain CB-II are built from processes stemming from RapidMiner (see Zeyen, Malburg, and Bergmann [31] for more details), resulting in 509 training cases, 40 validation cases, and 60 test cases. CB-III contains workflows that represent production processes which are used in a smart manufacturing environment. The case base is composed of 75 training cases, nine validation cases, and nine test cases.

When training the neural networks with the data of these three case bases, all semantic graphs are encoded in a numeric vector format (see Hoffmann and Bergmann [14] for more information). The pointwise ranking models, i. e., the base models, are trained to predict pairwise graph similarities. The number of pairwise graph similarities for training, testing, and validation is precisely the square of the numbers given before, as there is a ground-truth similarity value calculated for each possible case pair. The pairwise ranking models, i. e., the proposed approach, are trained on preferences between graph pairs, computed based on the ground-truth graph similarities. A single training iteration uses a batch of pairs of graph pairs, and the preferences are predicted between each possible pair of graph pairs. This way, the number of different preferences to train the pairwise ranking models is much higher than the number of pairwise graph similarities to train the pointwise ranking models.

The training cases are used as training input for the neural networks, while the validation cases are used to monitor the training process. The proposed models using pairwise ranking use the same hyperparameter settings as the respective base models using pointwise ranking. The training of all models using the GEM architecture runs for 20 epochs, and the training of all models using the GMN architecture runs for 40 epochs. These settings were identified as the approximate points of convergence in pretests. A snapshot of the model is exported after each training epoch and the models with the lowest validation error within these 20 or 40 exported models, respectively, are used in the experiments. The examined metrics cover retrieval quality and performance: Quality is measured in terms of *correctness* (see Cheng *et al.* [9] for more details), retrieval hits, and *k-NN quality* (see Müller and Bergmann [28] for more details). The correctness (ranged between -1 and 1) describes the conformity of the ranking positions in the predicted ranking to the ground-truth ranking. Given two arbitrary workflow pairs $GP_1 = (QG, CG_1)$ and $GP_2 = (QG, CG_2)$, the correctness is decreased if GP_1 is ranked before GP_2 in the predicted ranking although GP_2 is ranked before GP_1 in the ground-truth ranking or vice versa. The retrieval hits measure the number of cases that are within the top-25 retrieved results in the predicted rankings and the ground-truth rankings. The k-NN quality (ranged between 0 and 1) extends the retrieval hits by considering what the similarity of the cases within the top-25 retrieval results is. Therefore, the 25 most similar cases from the ground-truth rankings are compared with the predicted rankings. Each case from the most similar cases that is missing in the predicted rankings decreases the quality, with highly relevant cases affecting the quality more than less relevant cases. In addition, the retrieval time in milliseconds is determined to compare the performance of the individual models. All experiments are computed on a machine, running Ubuntu 22.04, an AMD Ryzen 5700x, an NVIDIA RTX 3070 with 8 GB graphics RAM and 64 GB system RAM. Please note that a comparison between retrievers using A* search and retrievers using GNNs is presented by Hoffmann and Bergmann [14] and, thus, not included in these experiments.

8.4.2 Experimental Results

Table 8.1 shows the results of the experimental evaluation for all models on all domains regarding the metrics introduced before. The evaluated models are grouped regarding the used embedder, i. e., a tree embedder or a sequence embedder. The values highlighted in bold font mark the best metric values among all models for each domain. The results w. r. t. training time show a clear dominance of the pointwise GEM models, since they can cache embedding vectors due to their parallel model architecture (see Hoffmann and Bergmann [14] for more details). There is a general time advantage of GEM as opposed to GMN, which can be seen for pointwise as well as pairwise variants. When directly comparing the pointwise variants with the pairwise variants, it is apparent that the pairwise variants in most cases have longer execution times with an advance between 154 and 1600 times for the GEM variants and 0.92 and 3.36 times for the GMN variants. The GMNs of CB-III do not follow this trend, since they are consistently faster than their respective base models. Additionally, the results also show that most of the pairwise GEMs retrieve faster than the pointwise GMNs, which indicates that the comparison of retrieval time should be done with similarly performing models in terms of quality. Comparing the two rank aggregators, it can be seen that positive rank aggregation is almost always faster than greedy rank aggregation, which is to be expected given the different levels of computational complexity.

Table 8.1: Evaluation Results.

	Retriever	CB-I				CB-II				CB-III			
		Time (ms)	Corr.	Hits	Quality	Time (ms)	Corr.	Hits	Quality	Time (ms)	Corr.	Hits	Quality
Tree Emb.	GEM Pointwise	0.63	0.035	0.225	0.476	0.53	0.330	2.867	0.439	0.13	0.197	10.333	0.543
	GMN Pointwise	462.11	0.444	0.963	0.490	811.77	0.571	7.033	0.539	171.95	0.584	16.556	0.731
	GEM Pairwise (Positive)	695.63	0.490	3.688	0.546	394.78	0.507	3.567	0.449	20.50	0.470	15.222	0.682
	GEM Pairwise (Greedy)	799.49	0.478	**4.050**	**0.555**	431.98	0.506	3.650	0.451	20.27	0.478	15.333	0.687
	GMN Pairwise (Positive)	1473.28	0.550	2.150	0.519	1348.08	0.616	6.550	0.524	161.04	0.628	**17.333**	**0.760**
	GMN Pairwise (Greedy)	1570.79	**0.551**	2.150	0.519	1391.45	0.615	6.283	0.515	165.19	**0.629**	**17.333**	0.759
Sequence Emb.	GEM Pointwise	**0.48**	0.055	0.013	0.471	**0.44**	0.357	6.150	0.522	**0.08**	0.389	11.778	0.600
	GMN Pointwise	741.83	0.340	0.225	0.476	1110.58	0.615	6.933	0.532	297.90	0.600	15.889	0.710
	GEM Pairwise (Positive)	698.21	0.398	2.125	0.515	398.52	0.524	4.300	0.468	18.32	0.488	15.111	0.678
	GEM Pairwise (Greedy)	797.86	0.385	2.175	0.516	437.91	0.522	4.100	0.463	19.08	0.495	15.111	0.679
	GMN Pairwise (Positive)	1747.54	0.493	2.825	0.535	1674.35	**0.737**	9.600	0.601	284.88	0.555	14.444	0.674
	GMN Pairwise (Greedy)	1841.79	0.491	2.738	0.533	1712.71	0.736	**9.650**	**0.602**	285.98	0.563	14.556	0.677

The results regarding the quality metrics show that there is a general trend of pairwise models having a higher correctness and k-NN quality and a higher number of hits compared to pointwise models. Except for CB-I, the data also shows a superior quality of the GMN-based models over the GEM-based models. For instance, the differences in correctness between the pairwise models and the respective pointwise model are between 1.2 and 13.9 times for CB-I, 1.07 and 1.53 times for CB-II, and 0.92 and 2.41 times for CB-III. As with training time, some pairwise GMNs do not follow this trend and achieve worse quality values than the respective pointwise models. A comparison among the quality metrics for individual models shows an agreement on average with variations in some cases, e. g., a comparison of the pairwise GEM and GMN with greedy aggregation and tree embedding in CB-I shows a lower correctness of the GEM but, in turn, a higher k-NN quality. When comparing the average results of the two rank aggregators, it can be seen that greedy rank aggregation is usually on-par with positive rank aggregation, with small differences in favor of one or the other approach in some comparisons.

Overall, the results show consistent improvements in quality when using pairwise compared to pointwise ranking approaches. However, these improvements mostly come at the cost of higher retrieval times of pairwise ranking approaches. It leads to a trade-off between these two aspects that should be considered regarding the use case. There are still noteworthy examples where a pairwise GEM model outperforms a pointwise GMN model in terms of quality and performance, leading to a great potential for future use. The performance and the quality results are statistically significant according to a paired two-sample t-test ($p < 0.01$) and we accept hypothesis H1 and reject hypothesis H2.

8.5 Conclusion and Future Work

The proposed approach deals with case retrieval by using GNN-based predictions of case rankings in POCBR. Previous work on similarity-based case retrieval with GNNs is extended to use the neural networks for predicting pairwise preferences between cases and the query. To transform multiple preferences to a case ranking, two rank aggregation methods are discussed. The experimental evaluation indicates that the ranking-based models significantly increase the quality of the predicted rankings with a small increase in retrieval time compared to similarity-based models.

Future research for the proposed approach should examine concepts of listwise ranking methods [8] for similarity-based retrieval in POCBR, since it is, in general, more powerful than pointwise and pairwise ranking [23]. Furthermore, the proposed approach could be used with a wider variety of rank aggregators, which could include more complex or even simpler rank aggregators for different domains or retrieval scenarios [2, 17]. Additionally, the approach could be extended to factor in uncertainty, e. g., by not only predicting a pairwise preference, but also the degree of uncertainty for this prediction. Existing methods such as Guiver and Snelson [13] could then be used to do rank aggregation and determine the final ranking.

Acknowledgements

This work is funded by the German Federal Ministry for Economic Affairs and Climate Action under grant No. 22973 *SPELL*.

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994). https://doi.org/10.3233/AIC-1994-7104

2. Allwein, E.L., Schapire, R.E., Singer, Y.: Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. J. Mach. Learn. Res. **1**, 113–141 (2000)

3. Amin, K., Kapetanakis, S., Polatidis, N., Althoff, K., Dengel, A.: DeepKAF: A Heterogeneous CBR & Deep Learning Approach for NLP Prototyping. In: Ivanovic, M., Yildirim, T., Trajcevski, G., Badica, C., Bellatreche, L., Kotenko, I.V., Badica, A., Erkmen, B., Savic, M. (eds.) International Conference on INnovations in Intelligent SysTems and Applications, INISTA 2020, Novi Sad, Serbia, August 24-26, 2020, pp. 1–7. IEEE (2020). https://doi.org/10.1109/INISTA49547.2020.9194679

4. Avesani, P., Ferrari, S., Susi, A.: Case-Based Ranking for Decision Support Systems. In: Ashley, K.D., Bridge, D.G. (eds.) Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR 2003, Trondheim, Norway, June 23-26, 2003, Proceedings. LNCS, vol. 2689, pp. 35–49. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45006-8_6

5. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014). https://doi.org/10.1016/J.IS.2012.07.005

6. Bichindaritz, I.: Memory Organization as the Missing Link between Case Based Reasoning and Information Retrieval in Biomedicine. In: Brüninghaus, S. (ed.) 6th International Conference on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Workshop Proceedings, pp. 18–31 (2005)

7. Brinker, K., Hüllermeier, E.: Label Ranking in Case-Based Reasoning. In: Weber, R., Richter, M.M. (eds.) Case-Based Reasoning Research and Development, 7th International Conference on Case-Based Reasoning, ICCBR 2007, Belfast, Northern Ireland, UK, August 13-16, 2007, Proceedings. LNCS, vol. 4626, pp. 77–91. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74141-1_6

8. Cao, Z., Qin, T., Liu, T., Tsai, M., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: Ghahramani, Z. (ed.) Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007. ACM International Conference Proceeding Series, pp. 129–136. ACM (2007). https://doi.org/10.1145/1273496.1273513
9. Cheng, W., Rademaker, M., de Baets, B., Hüllermeier, E.: Predicting Partial Orders: Ranking with Abstention. In: Cellular automata. Ed. by S. Bandini, S. Manzoni, H. Umeo, and G. Vizzari, pp. 215–230. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-15880-3_20
10. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to Order Things. J. Artif. Intell. Res. **10**, 243–270 (1999). https://doi.org/10.1613/JAIR.587
11. Fürnkranz, J., Hüllermeier, E.: Preference Learning: An Introduction. In: Preference Learning. Ed. by J. Fürnkranz and E. Hüllermeier, pp. 1–17. Springer (2010). https://doi.org/10.1007/978-3-642-14125-6_1
12. Glöckner, I., Weis, K.: An Integrated Machine Learning and Case-Based Reasoning Approach to Answer Validation. In: 11th International Conference on Machine Learning and Applications, ICMLA, Boca Raton, FL, USA, December 12-15, 2012. Volume 1, pp. 494–499. IEEE (2012). https://doi.org/10.1109/ICMLA.2012.90
13. Guiver, J., Snelson, E.L.: Learning to rank with SoftRank and Gaussian processes. In: Myaeng, S., Oard, D.W., Sebastiani, F., Chua, T., Leong, M. (eds.) Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008, pp. 259–266. ACM (2008). https://doi.org/10.1145/1390334.1390380
14. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027
15. Hoffmann, M., Malburg, L., Bach, N., Bergmann, R.: GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 240–255. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_16
16. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15
17. Hüllermeier, E., Fürnkranz, J.: Comparison of Ranking Procedures in Pairwise Preference Learning. In: Bouchon-Meunier, B., Coletti, G., Yager, R. (eds.) Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04), Perugia, Italy (2004)
18. Hüllermeier, E., Schlegel, P.: Preference-Based CBR: First Steps toward a Methodological Framework. In: Ram, A., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 19th International Conference on Case-Based Reasoning, ICCBR 2011, London, UK, September 12-15, 2011. Proceedings. LNCS, vol. 6880, pp. 77–91. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23291-6_8

19. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 188–203. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_13

20. Leake, D., Crandall, D.J.: On Bringing Case-Based Reasoning Methodology to Deep Learning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 343–348. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_22

21. Leake, D., Ye, X.: Harmonizing Case Retrieval and Adaptation with Alternating Optimization. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) Case-Based Reasoning Research and Development - 29th International Conference, ICCBR 2021, Salamanca, Spain, September 13-16, 2021, Proceedings. LNCS, vol. 12877, pp. 125–139. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-86957-1_9

22. Li, H., Sun, J.: Predicting business failure using forward ranking-order case-based reasoning. Expert Syst. Appl. **38**(4), 3075–3084 (2011). https://doi.org/10.1016/J.ESWA.2010.08.098

23. Liu, T.-Y.: Learning to Rank for Information Retrieval. Springer, Berlin and Heidelberg (2011)

24. Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. **61**(1), 83–111 (2023). https://doi.org/10.1007/S10844-022-00766-W

25. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. **51**(11), 8107–8118 (2021). https://doi.org/10.1007/S10489-021-02251-3

26. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014). https://doi.org/10.1016/J.IS.2013.06.004

27. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer, Wiesbaden (2018)

28. Müller, G., Bergmann, R.: A Cluster-Based Approach to Improve Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, pp. 639–644. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-639

29. Ye, X., Leake, D., Crandall, D.: Case Adaptation with Neural Networks: Capabilities and Limitations. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS, vol. 13405, pp. 143–158. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_10

30. Zeyen, C., Bergmann, R.: A*-Based Similarity Assessment of Semantic Graphs. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 17–32. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_2

31. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings. LNCS, vol. 11680, pp. 388–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29249-2_26

Chapter 9

Improving Automated Hyperparameter Optimization with Case-Based Reasoning

Bibliographic Information

Hoffmann, M., Bergmann, R.: Improving Automated Hyperparameter Optimization with Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS,. Vol. 13405, pp. 273–288. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_18

Abstract

The hyperparameter configuration of machine learning models has a great influence on their performance. These hyperparameters are often set either manually w. r. t. to the experience of an expert or by an *Automated Hyperparameter Optimization (HPO)* method. However, integrating experience knowledge into HPO methods is challenging. Therefore, we propose the approach HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) that uses *Case-Based Reasoning (CBR)* to improve the optimization of hyperparameters. HypOCBR is used as an addition to HPO methods and builds up a case base of sampled hyperparameter vectors with their loss values. The case base is then used to retrieve hyperparameter vectors given a query vector and to make decisions whether to proceed trialing with this query or abort and sample another vector. The experimental evaluation investigates the suitability of HypOCBR for two deep learning setups of varying complexity. It shows its potential to improve the optimization results, especially in complex scenarios with limited optimization time.

Keywords

Case-Based Reasoning · Automated Hyperparameter Optimization · Machine Learning · Deep Learning

Copyright Notice

9.1 Introduction

In recent years, machine learning and especially *Deep Learning (DL)* models have been used as a method of choice for solving various tasks, e. g., in decision support systems [18] and in helpdesk scenarios [2]. An important part of these models are the *hyperparameters* that are used to configure the model architecture or the learning process such as the number of layers or the learning rate. Hyperparameters are numerous in modern DL setups and there is an ongoing trend towards even more complex models, making it increasingly difficult to find suitable configurations for the large number of hyperparameters. Despite the existence of *Automated Hyperparameter Optimization (HPO)* methods [7, 9, 16], it is still common practice to tune these hyperparameters manually based on user experience. This task is, on the one hand, challenging since it requires in-depth knowledge of the underlying task and DL setup to set hyperparameter values appropriately. On the other hand, it is time-consuming as the process involves manually triggered iterations of selecting a hyperparameter configuration, training the parameterized model, and validating its performance. Whereas the latter aspect can be automated even by the simplest HPO methods, expert knowledge of the DL setup to be tuned can be hardly considered in current HPO methods (e. g., [8, 12, 15]). For instance, there might be knowledge in the form of a statement such as this: *The second convolutional layer has a great influence on the overall results.* To use this knowledge in current HPO approaches, it would be necessary to transform it into meta-parameters of the HPO method, such as the number of sampled hyperparameter vectors. However, this transformation is not trivial and requires in-depth knowledge of the HPO method, which the neural network modeler might be lacking.

To address these shortcomings, we propose an approach called HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) for combining HPO methods with a *Case-Based Reasoning (CBR)* [1] system that allows an explicit integration of expert knowledge into optimization procedures. The main assumption is that similar hyperparameter vectors lead to similar optimization results. Based on this assumption and modeled domain and similarity knowledge that expresses an expert's experience knowledge, HypOCBR uses a CBR system to filter sampled hyperparameter vectors from an HPO method and make decisions to proceed training with them or abort them. The proposed approach is designed to be used as an extension of existing HPO methods without required modifications to their core functionality. The remainder of the paper is structured as follows: Section 9.2 describes the fundamentals of HPO, including the hyperparameters of an example DL model and random search as a popular HPO method. Additionally, this section discusses related work about HPO in CBR research. Section 9.3 describes our approach by introducing the proposed system architecture and the used retrieval and decision-making process. The presented approach is evaluated in Sect. 9.4 where optimization procedures with and without CBR methods are compared on two DL setups of varying complexity. Finally, Sect. 9.5 concludes the paper and examines future research directions.

9.2 Automated Hyperparameter Optimization of Deep Learning Models

Our approach aims at optimizing *Deep Learning (DL)* models with the help of *Case-Based Reasoning (CBR)* methods that are built on top of existing *Automated Hyperparameter Optimization (HPO)* methods. In this section, we introduce the formal concepts and terminology of HPO (see Sect. 9.2.1). Additionally, different types of hyperparameters (see Sect. 9.2.2) are examined and random search, a popular HPO method (see Sect. 9.2.3), is presented. We further discuss related work in Sect. 9.2.4.

9.2.1 Formal Definition

Hyperparameter optimization involves two basic components in its simplest form, i. e., a training setup of a DL model including its hyperparameters and training data, and a search procedure for selecting a hyperparameter vector from the hyperparameter search space. Our introduction and notation of these components follows Feurer and Hutter [9]: A DL model $\mathcal{A}$ is parameterized by a set of hyperparameters N, where each hyperparameter $n \in N$ is defined by its domain Λ_n. The overall hyperparameter search space is given by $\Lambda = \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_n$. We denote $\lambda \in \Lambda$ as a hyperparameter vector that represents a distinct sample from the hyperparameter search space. Instantiating the model $\mathcal{A}$ with this sample is denoted as $\mathcal{A}_\lambda$. The optimization procedure is defined as follows:

$$\lambda^* = \operatorname*{argmin}_{\lambda \in \Lambda} \mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}) \tag{9.1}$$

Thereby, the optimal hyperparameter vector $\lambda^* \in \Lambda$ is determined by selecting a hyperparameter vector $\lambda \in \Lambda$ that minimizes the loss function $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D})$. The loss value is the result of training $\mathcal{A}_\lambda$ and validating its performance on the dataset $\mathcal{D}$. Therefore, $\mathcal{D}$ is usually split up into a training dataset for training $\mathcal{A}_\lambda$ and a validation dataset for computing the loss value. We refer to the training, validation, and loss computation with a single hyperparameter vector by the term *trial*. Please note that instead of loss values, which are always minimized, also other metrics, e. g., classification accuracy, can be used for optimization according to Eq. 9.1. The only necessary change is to compute argmax($\cdot$) instead of argmin($\cdot$) in some cases. When referring to loss values in this paper, we mean both types, i. e., metrics to minimize and to maximize.

9.2.2 Hyperparameter Search Spaces

The computational effort of an optimization as shown in Eq. 9.1 is mainly determined by the number of possible hyperparameter vectors $|\Lambda|$ and the time needed to perform trials with the parameterized models. But, even if the training time is small and it is a simple DL setup, Λ can become very large and the whole optimization process very slow. Consider the following example[1]: A DL setup for image classification is made up of two consecutive layers of convolution followed by pooling. The convoluted images are then flattened, i. e., reshaped to a one-dimensional vector, and fed into several fully-connected layers. The final output is a vector of ten elements that represents the probabilities for the individual classes of a classification task. The model is trained by processing batches of images and applying stochastic gradient descent

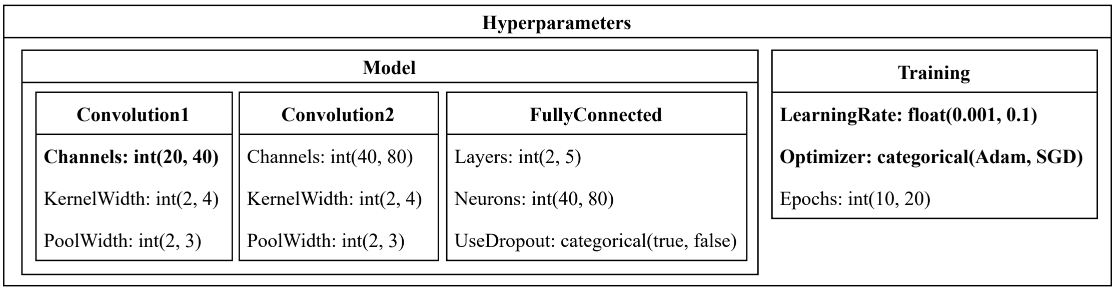


Figure 9.1: Example Hyperparameter Search Space

[1] The example is derived from an introduction on convolutional neural networks, accessible at https://www.tensorflow.org/tutorials/images/cnn.

according to the computed loss. A possible hyperparameter search space for this example setup is illustrated in Fig. 9.1. The hyperparameters are structured hierarchically, with model and training hyperparameters forming the main groups. Each hyperparameter belongs to a certain type of search space (Λ_n) and is constrained to certain values. For instance, the number of channels in the first convolution is an integer value between 20 and 40 and the learning rate is a float value between 0.001 and 0.1. These types of search spaces influence the number of possible values of each hyperparameter. For instance, categorical and integer hyperparameters such as the optimizer or the number of channels represent a definitive set of possible values. Continuous search space types such as the learning rate, in turn, have an infinite number of theoretically possible values, assuming an infinitely small step size. This makes it infeasible to trial with every possible hyperparameter vector to compute $\text{argmin}(\cdot)$ and to find λ^* [6].

9.2.3 Random Search

Therefore, most search algorithms only examine a small subset of Λ that does not guarantee finding the optimal hyperparameter vector [9]. Random search (see Alg. 3 and cf. [6] for more information) is a simple example of these algorithms. The algorithm is an approximation of the

Algorithm 3 Random Search

```
$\lambda \leftarrow \text{selectFrom}(\Lambda)$
$\lambda^+ \leftarrow \lambda$
while not isOptimized($\lambda^+, \mathcal{L}$) do
    $\lambda \leftarrow \text{selectFrom}(\Lambda)$
    if $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}) < \mathcal{L}(\mathcal{A}_\lambda^+, \mathcal{D})$ then
        $\lambda^+ \leftarrow \lambda$
    end if
end while
return $\lambda^+$
```

function $\text{argmin}(\cdot)$ from Eq. 9.1 and works in the following way: Random hyperparameter vectors λ are selected from Λ by the function $\text{selectFrom}(\cdot)$ and trialed. If the loss of the selected sample is smaller than the loss of the current best hyperparameter vector λ^+, λ will be assigned to λ^+. This loop repeats until the search is finished (function $\text{isOptimized}(\cdot, \cdot)$) and the best selected sample λ^+ is returned. The used termination criterion is dependent on the underlying scenario and can be, for instance, the maximum search time, the maximum number of selected samples, or a threshold of the loss value. If random search has an unlimited budget, it will converge towards λ^+ being equal to the optimal hyperparameter vector λ^*.

9.2.4 Related Work

We discuss related work that covers popular HPO methods and CBR approaches in the field of HPO. Literature from the former category is discussed in detail in several survey publications (e. g., [9, 16, 26]), which is why we only want to highlight two popular example approaches: Multi-fidelity optimization aims at approximating the actual loss value of a setup by conducting low-fidelity and high-fidelity optimizations with a specific budget allocation, e. g., a maximum number of training iterations. Hyperband [15] uses an iterative process where the budget for each trial is determined based on the results of this run in the last iteration. Well-performing runs are allocated more resources and poorly-performing runs are terminated, until only the best-performing optimization is left. Model-based black box optimization methods aim to improve on other methods, e. g., Hyperband, that usually sample random hyperparameter vectors. Instead, promising hyperparameter vectors are chosen based on a surrogate model that is built during the optimization to copy the hyperparameter distribution of the black box method to optimize, e. g., a DL model. Bayesian optimization [24] methods use a surrogate model that is based on probabilistic methods, e. g., Gaussian processes. Our approach is compatible with both categories of HPO methods and, additionally, provides the opportunity of expert knowledge integration into the optimization process. Expert knowledge is directly integrated as CBR domain and similarity knowledge instead of being indirectly modeled in existing HPO methods.

The joint applications of CBR and HPO methods in literature are mostly concerned with hyperparameter tuning of CBR algorithms, e. g., feature weighting [25]. The opposite relation of CBR used for HPO is, to the best of our knowledge, not commonly discussed. We highlight some selected approaches: Pavón *et al.* [20] use CBR to enable automatic selection of Bayesian model configurations for individual problem instances. Their application builds up a case base of configurations, which is used to find the best-matching configuration for an upcoming problem instance. This idea is further pursued by Yeguas *et al.* [27] and applied to the task of tuning parameters of evolutionary optimization algorithms. Auslander, Apker, and Aha [3] explore a case-based approach of setting parameter values in the scenario of multi-agent planning. They retrieve parameter settings from a case base to parameterize upcoming planning problems. Molina *et al.* [19] use a set of past decision tree executions to predict suitable parameters for the application of the decision tree on new datasets. The parameter settings are retrieved based on the characteristics of the datasets. In contrast to the approach in this paper, most of the presented approaches can be classified as AutoML methods [13] that are used as a replacement for hyperparameter optimization. These methods automatically select a model configuration according to the training data or the learning task instead of tuning parameters from scratch. Our approach is novel in the sense that it aims to integrate CBR within the optimization procedure to make expert knowledge about the hyperparameter search space available and guide the optimization by eliminating unpromising hyperparameter vectors.

9.3 Automated Hyperparameter Optimization with Case-Based Reasoning

This section introduces our approach HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) that aims at improving *Automated Hyperparameter Optimization (HPO)* of *Deep Learning (DL)* hyperparameters with *Case-Based Reasoning (CBR)* methods. Our goal is to accelerate the optimization process by utilizing HypOCBR as a filter for the hyperparameter vectors that classifies whether a vector should be considered for trialing or not. Thereby, the strategy is to disregard hyperparameter vectors in situations where they are highly similar to known low-performing examples that are collected in a case base throughout the optimization procedure. Figure 9.2 shows an architectural overview of the three involved components, i. e., the CBR system, the HPO method, and HypOCBR. The HPO method carries out the optimization procedure by sampling hyperparameter vectors from the search space, performing training, and validating the model's performance (see Sect. 9.2.3). The architecture, in this regard, allows using arbitrary HPO methods, since their functionality does not have to be fitted to HypOCBR. The CBR system supports the optimization procedure by maintaining a case base with completed trials of hyperparameter vectors and their corresponding loss values as well as domain and similarity models to include domain expert knowledge. HypOCBR connects the CBR system and the HPO method by making use of a lightweight implementation of the 4R cycle [1]: After a new hyperparameter vector is sampled during the optimization, HypOCBR makes a decision to proceed with this vector or abort it early and sample another vector. Each sampled vector serves

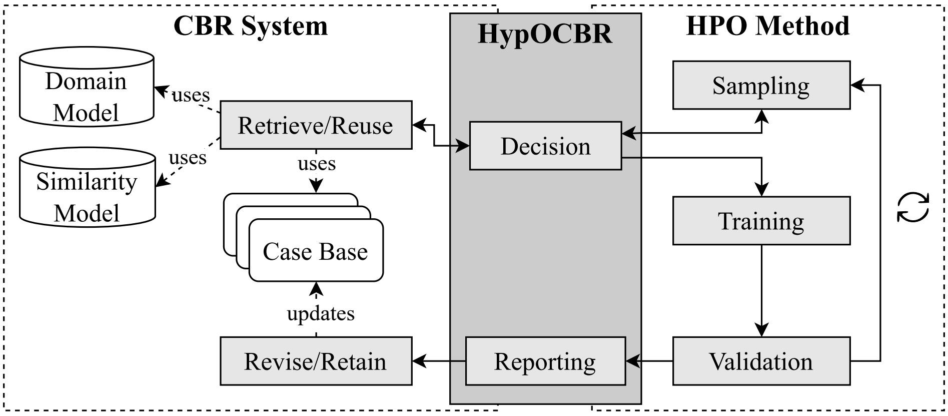


Figure 9.2: Architecture of the CBR system, the HPO method, and HypOCBR

as a query to the CBR system based on which the case base is searched for similar hyperparameter vectors and their corresponding loss values, i. e., retrieving trials. HypOCBR then reuses the most similar trials and makes the decision based on their loss values. Please note that the retrieved hyperparameters are not adapted but their loss values are used to assess the potential of the queried hyperparameter vector. Each optimization gradually builds up a case base to use by starting with an empty case base and retaining each completed trial therein. Thereby, we do not consider an explicit revision of cases in the approach but rather store all reported validation results from the HPO component. Please note that this can lead to large case bases in long optimization runs, such that methods for speeding up retrieval (e. g., [17]) or decreasing the size of the case base (e. g., [21]) can be necessary but are not further discussed in this work. In the following, we will examine how a retrieval in the CBR system is performed by inspecting the case representation as well as the similarity definitions (see Sect. 9.3.1). The retrieval is part of the decision-making process of HypOCBR that is explained in Sect. 9.3.2.

9.3.1 Experience-Based Retrieval: Case Representation and Similarity Definition

The case representation models trials as pairs of their hyperparameter vector and the respective loss value. The case base, in this regard, stores information on the quality of hyperparameter vectors that were sampled during the optimization. The hyperparameter vectors are represented as object-oriented cases, as shown by the example in Fig. 9.1. The domain model describing these vectors is one point to integrate expert knowledge. The concrete form is highly dependent on the structure of the modeled hyperparameter vectors. For instance, simple knowledge about the data types, value ranges, and expected distributions of individual hyperparameters (see Sect. 9.2.2) can build the baseline. It can be extended by knowledge about the structure of individual hyperparameters (e. g., taxonomies of optimizer types) or the constraints among hyperparameters (e. g., the width of the kernel has to be less than the image width and height). Even more complex knowledge such as ontologies can be used to model the cases. The loss value that is also part of each case, usually is a single numeric value, e. g., accuracy, but can also be represented as a more complex object, e. g., a list of accuracy values from each epoch of training.

During retrieval, similarities are computed between the hyperparameter vectors of the cases and the hyperparameter vector that is given as the query by HypOCBR. We do not specify the similarity measures to use, but rather allow them to be customized regarding the use case and the available expert knowledge. Thereby a global object-oriented similarity between two hyperparameter vectors is usually put together from multiple local similarities between the individual hyperparameters, e. g., numeric measures for integer and float search spaces, binary measures for boolean search spaces, and tables or taxonomies for categorical values (cf. [4, pp. 93f.] for more information). This enables a precise customization of the global pairwise similarity based on the expert's experience: For instance, the weights of all hyperparameters from the training category (see Fig. 9.1) can be increased to focus more on this part of the hyperparameter vectors for similarity assessment.

9.3.2 Making a Decision for New Trials

The decision task of HypOCBR applies an algorithm to make a decision between *proceeding* with a sampled hyperparameter vector or *aborting* the trial. The used approach is conceptualized in Alg. 4 and can be parameterized according to the well-known principles of *exploration* and *exploitation* (see, for instance, Leake and Schack [14] for more information). This means that it can be configured towards favoring hyperparameter vectors that are different from all previously encountered vectors (exploration) or similar to known good hyperparameter vectors (exploitation). The meta-parameters to influence the behavior of the algorithm are given in Tab. 9.1.

The algorithm's first step is to check if the case base has reached a certain size (lines 1 – 3) which is given as the meta-parameter θ_1. This check is necessary as a case base is built up throughout the optimization to improve the quality of upcoming decisions. The case base, however, suffers from the cold start problem [23] which can cause undesired behavior in CBR systems with little or no data at the launch of the application (e. g., [22]). If the case base has reached the desired size, it can be used for retrieving r (line 4) with the query λ_q, the case base CB, and the similarity measure for case pairs sim. The retrieval incorporates the meta-parameter θ_4 that defines the number of nearest neighbors to retrieve. Larger values of θ_4 help

Algorithm 4 Decision-Making for New Trials

Input: case base CB, similarity function $\mathrm{sim}(\cdot,\cdot)$, query λ_q
Output: PROCEED or ABORT
Meta-Parameters: θ_1, θ_2, θ_3, θ_4

```
 1: if $|CB| < \theta_1$ then
 2:     return PROCEED                      ▷ Minimum size of case base not reached
 3: end if
 4: $r \leftarrow \mathrm{retrieve}(CB, \mathrm{sim}, \lambda_q, \theta_4)$
 5: $s \leftarrow \mathrm{aggregateSimilarities}(r)$
 6: if $s \geq \theta_2$ then
 7:     $l \leftarrow \mathrm{aggregateLosses}(r)$
 8:     $p \leftarrow \mathrm{percentile}(l, CB, \theta_3)$
 9:     if $p = true$ then
10:         return PROCEED                  ▷ Exploitation
11:     else
12:         return ABORT                    ▷ Similar to underperforming cases
13:     end if
14: else
15:     return PROCEED                      ▷ Exploration
16: end if
```

to prevent overfitting and make the retrieved cases more robust to outliers. These cases are then used to compute an aggregate similarity value s that measures how similar λ_q is to the retrieved cases from the case base r (line 5). The aggregated similarity is computed to ensure that the retrieved cases are sufficiently similar to make a decision. The retrieval is also the main part of the algorithm where expert knowledge is used. It determines the computed similarities, which direct the algorithm to the hyperparameter vectors and their loss values that, in turn, best match the queried vector. As the retrieved cases are the basis of the decision, the knowledge directly influences the measured potential of the query vector. If s does not exceed θ_2 (line 6) then the exploration strategy is followed and `PROCEED` is returned (see line 15). Otherwise, the retrieved cases are further inspected in lines 7 – 13. The loss values of the retrieved cases are aggregated to l (line 7) to get an average loss of the cases that are similar to the query. Given l, the case base *CB*, and the meta-parameter θ_3 as parameters, the function percentile$(\cdot,\cdot,\cdot)$ in line 8 returns a boolean value that expresses if the average loss value l is in the percentile range given by θ_3. For instance, setting θ_3 to a value of 55 expresses that p is true if l is among the best 45 % of the loss values in the case base. This gives a hint towards the potential quality of λ_q which is based on the average quality of cases similar to λ_q (retrieved cases r) w. r. t. to all cases in the case base *CB*. Eventually, `PROCEED` is returned if p is true (exploitation strategy; line 9) and `ABORT` is returned if p is false. The algorithm only aborts in one particular scenario, which highlights the main goal of the filter process, i. e., to abort trials if they are expected to lead to bad results.

Table 9.1: Meta-Parameters of HypOCBR

Meta-Parameter	**Description**	**Constraints**
θ_1	Minimum number of cases in the case base to avoid cold start	$\theta_1 > 0$
θ_2	Minimum similarity threshold of aggregated retrieved cases	$\theta_2 \in [0,1]$
θ_3	Minimum percentile of aggregated metric of retrieved cases	$\theta_3 \in [0,100]$
θ_4	Number of cases to retrieve	$\theta_4 <= \theta_1$

Furthermore, the meta-parameters (see Tab. 9.1) provide a straightforward way of configuring the behavior of the algorithm w. r. t. certain goals. For instance, it can be configured towards exploration by increasing θ_2 which requires a higher similarity of retrieved cases. A parameterization towards an `ABORT` decision is also possible by decreasing θ_2 and increasing θ_3. Additionally, pretests of an implementation of Alg. 4 with standard similarity measures of the process-oriented CBR framework ProCAKE [5] have shown that the performance is adequate, enabling working with case bases containing hundreds or thousands of trials within a few milliseconds.

9.4 Experimental Evaluation

The experimental evaluation compares the results and procedure of hyperparameter optimization with and without the usage of the proposed *Case-based Reasoning (CBR)* component HypOCBR. To include optimization tasks with different levels of complexity, we evaluate a rather simple baseline setup of an image classification task and a much more complex similarity learning task based on *Graph Neural Networks (GNNs)*. The goal is to analyze the influence of HypOCBR on the number of sampled hyperparameter vectors and the loss values of the trialed cases. The following hypotheses are examined:

H1 The computation overhead introduced by HypOCBR is negligible.

H2 The usage of HypOCBR in an optimization procedure leads to better results compared to an identical optimization without it.

H3 The benefit of integrating HypOCBR in an optimization procedure increases with the complexity, i. e., longer running trials with less overall optimization budget, of the setup to optimize.

Hypothesis H1 addresses the computation overhead introduced by the CBR system, which is expected to not influence the completed number of trials compared to HPO without HypOCBR. Hypothesis H2 examines the potential of the presented approach to conduct better trials w. r. t. their loss values. Hypothesis H3 aims at investigating the influence of the setup complexity on the benefit of HypOCBR.

9.4.1 Experimental Setup

The experiments feature two *Deep Learning (DL)* setups. The baseline setup (S - I) is a convolutional neural network classifying CIFAR10 images (derived from the example in Sect. 9.2.2). The model is fully trained for ten epochs within one optimization run with the goal of maximizing the percentage of correctly classified images (*accuracy*). The hyperparameter vectors of the model contain 11 parameters with only integer search spaces. The number of possible hyperparameter combinations, i. e., the number of unique hyperparameter vectors, is approx. $3 \cdot 10^7$. Additionally, we also want to look at a more complex problem by incorporating a second setup (S - II) that uses GNNs. S - II is based on our previous work on using graph embeddings for similarity learning [10, 11]. Due to the required time for a full training run being approx. 12 – 18 hours, the iterations in each epoch are limited and the model is only trained for ten epochs in trials. Each hyperparameter vector contains 26 hyperparameters, i. e., 18 integer hyperparameters, five float hyperparameters, and 3 categorical hyperparameters. This results in a total number of approx. $2 \cdot 10^{37}$ possible hyperparameter vectors. The target metric is the *Mean Absolute Error (MAE)* between the predicted similarities and the label similarities that should be minimized (see more details in [10, 11]). Both setups are validated on a part of the dataset that is disjoint from the data that is used for training. We use simple domain models for both setups that include the data types and value ranges of all hyperparameters as well as an object-oriented structure of the vectors. The similarity model is also simple with linear numeric measures for integers and floats and binary measures for categorical hyperparameters. The similarity between two hyperparameter vectors is computed by a weight-adjusted average of the object-oriented structure of the hyperparameters.

The HPO method used in the experiments is random search (see Sect. 9.2.3) as it is a baseline method that is often used and performs reasonably well among other state-of-the-art optimization methods [9]. This way, we can focus on the effects of HypOCBR without the need for factoring in differently parameterized HPO methods which might be required, e. g., if using Bayesian methods. Each optimization procedure is conducted twice, with the only difference being that HypOCBR is just used in one procedure. All other factors are kept identical, which includes the dataset, the training procedure, and the HPO procedure. Randomized parts, e. g., the initial model weights and biases or the sampled hyperparameter vectors, are made deterministic by using random seed values. This means that it is possible to perform an optimization run with the same results if the same random seed is used. The used meta-parameters of HypOCBR are set according to results of pretests. For S - I, that is $\theta_1 = 20$, $\theta_2 = 0.81$, $\theta_3 = 50$, $\theta_4 = 15$ and for S - II that is $\theta_1 = 15$, $\theta_2 = 0.7$, $\theta_3 = 45$, $\theta_4 = 10$.

The termination criterion for each optimization with and without HypOCBR is the maximum elapsed time. A value of one hour is set for all optimizations of S - I and four hours for all optimizations of S - II. These values reflect the different complexities of both setups, where training and validation takes longer for S - II. After the optimization process is terminated, the trained and validated hyperparameter vectors are analyzed. We assume a scenario where an expert reviews the best hyperparameter vectors to make the final decision. Therefore, the validation results, i. e., the accuracy or MAE, of all trials and the ten best trials are compared among optimization procedures with and without HypOCBR. The implementation is realized as an extension of the open-source process-oriented CBR framework ProCAKE[2] [5]. All experiments are conducted on a computer with an Intel Xeon Gold 6138 CPU and an NVIDIA Tesla V100 SXM2, running Ubuntu 18.04 LTS.

9.4.2 Experimental Results

The results of the conducted experiments are depicted in Tab. 9.2 for S - I and in Tab. 9.3 for S - II. The information in both tables is structured analogously: The lines show ten optimization procedures, once conducted with and without HypOCBR under the same circumstances, i. e., with the same random seed. We give information on the number of trials that were evaluated (two leftmost columns), which is further split up into the trials with a *proceed* and an *abort* decision for the optimizations with HypOCBR. The loss values of the individual optimizations are percentage differences between optimizations with and without HypOCBR. A negative value corresponds to a negative impact of HypOCBR and vice versa. The table shows best, mean, and median values aggregated from all trials (left) and the ten best trials (right) of the individual optimizations. All presented differences are statistically significant ($p < 0.01$).

Table 9.2: Evaluation Results for S - I

Trials		Loss (All)			Loss (Top)	
#	# (Proceed, Abort)	Best	Mean	Median	Mean	Median
161	620 (155, 465)	0.00%	0.50%	1.10%	0.70%	0.81%
152	609 (149, 460)	-0.44%	1.85%	1.60%	0.87%	0.99%
141	646 (143, 503)	0.00%	1.64%	1.50%	0.49%	0.66%
149	728 (141, 587)	1.37%	1.18%	1.29%	0.74%	0.55%
153	640 (149, 491)	2.33%	1.02%	0.72%	0.76%	0.57%
158	613 (158, 455)	-0.10%	0.90%	0.49%	0.29%	0.37%
161	456 (157, 299)	0.00%	1.49%	1.10%	1.00%	1.17%
164	413 (160, 253)	-0.43%	2.00%	1.94%	0.30%	0.41%
163	863 (159, 704)	-0.34%	1.96%	2.21%	0.52%	0.69%
159	690 (158, 532)	0.77%	1.13%	0.71%	0.37%	0.43%

The results for S - I (see Tab. 9.2) show that the HypOCBR approach aborts many trials and only proceeds with a small share. The number of trials that are not aborted is in a similar range as the number of trials when HypOCBR is not used. However, the optimizations that use HypOCBR are capable of pre-filtering a larger number of trials (increase between 251 % and 529 %) which provides a better coverage of the search space. The aggregated accuracies (higher values are better) show better results of the mean and median values for HypOCBR on average, while the approach without found a better vector in four optimizations. All in all, the differences between both variants are small, ranging from 0.29 % to 2 % for the mean and median accuracies and from -0.44 % to 2.33 % for the best accuracy.

The results for S - II (see Tab. 9.3) show that the number of proceeded trials for optimizations with HypOCBR is approximately equal to the number of trials for optimizations without HypOCBR while pre-selecting numerous trials (increase from 144 % to 854 %). The aggregated

[2] http://procake.uni-trier.de

Table 9.3: Evaluation Results for S - II

Trials		Loss (All)			Loss (Top)	
#	# (Proceed, Abort)	Best	Mean	Median	Mean	Median
105	685 (110, 575)	-0.19%	17.15%	16.21%	22.95%	22.00%
108	923 (103, 820)	2.97%	16.17%	34.29%	17.13%	27.17%
113	671 (111, 560)	0.38%	16.97%	17.47%	14.13%	11.70%
113	865 (103, 762)	0.00%	16.52%	12.31%	20.13%	17.71%
107	155 (116, 39)	0.00%	7.01%	21.39%	18.11%	19.73%
112	169 (118, 51)	2.75%	4.33%	21.23%	8.62%	14.06%
115	750 (112, 638)	22.60%	16.07%	14.66%	26.00%	49.14%
113	876 (105, 771)	0.00%	11.80%	9.76%	14.57%	4.52%
114	716 (114, 602)	-0.03%	15.58%	19.27%	-0.70%	3.48%
113	171 (110, 61)	0.00%	6.53%	22.57%	0.07%	0.00%

MAE values (lower values are better) generally show improvements when using HypOCBR: The best MAE values show decreases between -0.19 % and 22.6 %. The mean and median MAE values mostly show large decreases of up to 49.14 %.

9.4.3 Discussion

The overall results of the optimization procedures that use HypOCBR are promising. Regarding Hypothesis H1, HypOCBR optimizations manage to train and validate approx. the same number of trials as the corresponding optimizations without HypOCBR. The small deviations of the results follow no trend and are most likely caused by computational variations of the underlying hardware. Thus, the introduced performance overhead is negligible and H1 is accepted. When analyzing the MAE and accuracy values, HypOCBR improves the optimization results in most cases with a few exceptions that stem largely from the best trials. The overall small improvements for S - I might be due to the full training run that is conducted during trialing. Thereby, the optimizer has the complete dataset to train the model and, thus, might be able to find suitable model weights even for inferior hyperparameters. The results for S - II show much clearer improvements than for S - I. With median MAEs being decreased by up to 49 %, HypOCBR significantly improves the optimization procedures. This especially reveals potential in scenarios with a limited time budget for optimization where only a small amount of trials is possible, e. g., in prototyping. There, the CBR component allows sampling more hyperparameter vectors that can be assessed for trialing. Although some of these trials might be misclassified by HypOCBR, it still improves the overall results of the optimization. The consistently improved mean and median results of all trials also hints at a success of eliminating bad trials. Therefore, we conclude that Hypothesis H2 is only partly accepted, since the integration of HypOCBR does not improve the optimization results in every case. The results are consistent with Hypothesis H3 due to the noticeably improved optimization for the more complex setup S - II compared to S - I. However, additional tests are needed to solidify the results.

9.5 Conclusion and Future Work

This paper introduced our approach called HypOCBR for enabling the use of *Case-Based Reasoning (CBR)* to improve *Automated Hyperparameter Optimization (HPO)*. HypOCBR acts as a filter for sampled hyperparameter vectors, deciding whether a vector should be considered for training and validation or not. The decision is based on a case base of hyperparameter vectors that were already trained and validated during optimization and on expert-modeled domain and similarity knowledge. These components are utilized in the following decision-making process: retrieve similar hyperparameter vectors given a query vector, aggregate their loss values, compare the aggregated loss value to the distribution of loss values, proceed training and validation with the query if it is similar to well-performing hyperparameter vectors. When evaluated for two *Deep Learning (DL)* setups of varying complexity, optimization procedures with HypOCBR show great potential: Due to the filtering function, more hyperparameter vectors can be analyzed, which leads to better optimization results for both setups.

A focus of future research should be on further optimizing, extending, and evaluating the approach. For instance, the approach might benefit from self-learning capabilities for the decision-making process that enable automatic tuning of the meta-parameters during optimization and easier application of the approach to new data. Further, the role of the case base can be extended to reuse existing case bases during startup and export the case base after the optimization procedure. It could also be beneficial to investigate more advanced methods for the reuse and revise phases of the described lightweight CBR cycle, which might, for instance, include methods for case adaptation or managing the growth of the case base (e. g., [21]). Additionally, HypOCBR could be evaluated on a larger scale by covering other HPO methods (e. g., [8, 15]), other DL setups (see [10, 11] for more examples), and the influence of differently modeled domain and similarity knowledge.

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994). https://doi.org/10.3233/AIC-1994-7104
2. Amin, K., Lancaster, G., Kapetanakis, S., Althoff, K., Dengel, A., Petridis, M.: Advanced Similarity Measures Using Word Embeddings and Siamese Networks in CBR. In: Bi, Y., Bhatia, R., Kapoor, S. (eds.) Intelligent Systems and Applications - Proceedings of the 2019 Intelligent Systems Conference, IntelliSys 2019, London, UK, September 5-6, 2019, Volume 2. Advances in Intelligent Systems and Computing, pp. 449–462. Springer (2019). https://doi.org/10.1007/978-3-030-29513-4_32
3. Auslander, B., Apker, T., Aha, D.W.: Case-Based Parameter Selection for Plans: Coordinating Autonomous Vehicle Teams. In: Lamontagne, L., Plaza, E. (eds.) Case-Based Reasoning Research and Development - 22nd International Conference, ICCBR 2014, Cork, Ireland, September 29, 2014 - October 1, 2014. Proceedings. LNCS, vol. 8765, pp. 32–47. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-11209-1_4
4. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Springer (2002)
5. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Kapetanakis, S., Borck, H. (eds.) Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Reasoning co-located with the Twenty-seventh International Conference on Case-Based Reasoning (ICCBR 2019), Otzenhausen, Germany, September 8-12, 2019. CEUR Workshop Proceedings, pp. 156–161. CEUR-WS.org (2019)
6. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res. **13**, 281–305 (2012). https://doi.org/10.5555/2503308.2188395
7. Claesen, M., De Moor, B.L.R.: Hyperparameter Search in Machine Learning. CoRR **abs/1502.02127** (2015). arXiv: 1502.02127

8. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, pp. 1436–1445. PMLR (2018)

9. Feurer, M., Hutter, F.: Hyperparameter Optimization. In: Automated Machine Learning - Methods, Systems, Challenges. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren, pp. 3–33. Springer (2019). https://doi.org/10.1007/978-3-030-05318-5_1

10. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

11. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

12. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995, pp. 1942–1948. IEEE (1995). https://doi.org/10.1109/ICNN.1995.488968

13. Leake, D., Crandall, D.J.: On Bringing Case-Based Reasoning Methodology to Deep Learning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS, vol. 12311, pp. 343–348. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_22

14. Leake, D., Schack, B.: Exploration vs. Exploitation in Case-Base Maintenance: Leveraging Competence-Based Deletion with Ghost Cases. In: Cox, M.T., Funk, P., Begum, S. (eds.) Case-Based Reasoning Research and Development - 26th International Conference, ICCBR 2018, Stockholm, Sweden, July 9-12, 2018, Proceedings. LNCS, vol. 11156, pp. 202–218. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01081-2_14

15. Li, L., Jamieson, K.G., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. J. Mach. Learn. Res. **18**, 185:1–185:52 (2017)

16. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Netw. Model. Anal. Health Informatics Bioinform. **5**(1), 18 (2016). https://doi.org/10.1007/S13721-016-0125-6

17. Malburg, L., Hoffmann, M., Trumm, S., Bergmann, R.: Improving Similarity-Based Retrieval Efficiency by Using Graphic Processing Units in Case-Based Reasoning. In: Bell, E., Keshtkar, F. (eds.) Proceedings of the Thirty-Fourth International Florida Artificial Intelligence Research Society Conference, North Miami Beach, Florida, USA, May 17-19, 2021 (2021). https://doi.org/10.32473/FLAIRS.V34I1.128345

18. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. **51**(11), 8107–8118 (2021). https://doi.org/10.1007/S10489-021-02251-3

19. Molina, M.D.M., Romero, C., Ventura, S., Luna, J.M.: Meta-learning Approach for Automatic Parameter Tuning: A case of study with educational datasets. In: Yacef, K., Zaïane, O.R., Hershkovitz, A., Yudelson, M., Stamper, J.C. (eds.) Proceedings of the 5th International Conference on Educational Data Mining, Chania, Greece, June 19-21, 2012, pp. 180–183. www.educationaldatamining.org (2012)

20. Pavón, R., Díaz, F., Laza, R., Luzón, M.V.: Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study. Expert Syst. Appl. **36**(2), 3407–3420 (2009). https://doi.org/10.1016/J.ESWA.2008.02.044

21. Roth-Berghofer, T.: Knowledge maintenance of case-based reasoning systems - the SIAM methodology. PhD thesis, Kaiserslautern University of Technology, Germany (2003).

22. Sánchez, L.Q., Bridge, D.G., Díaz-Agudo, B., Recio-García, J.A.: A Case-Based Solution to the Cold-Start Problem in Group Recommenders. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, pp. 3042–3046. IJCAI/AAAI (2013)

23. Schafer, J.B., Frankowski, D., Herlocker, J.L., Sen, S.: Collaborative Filtering Recommender Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web, Methods and Strategies of Web Personalization. LNCS, vol. 4321, pp. 291–324. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_9

24. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian Optimization of Machine Learning Algorithms. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pp. 2960–2968 (2012)

25. Wettschereck, D., Aha, D.W.: Weighting Features. In: Veloso, M.M., Aamodt, A. (eds.) Case-Based Reasoning Research and Development, First International Conference, ICCBR-95, Sesimbra, Portugal, October 23-26, 1995, Proceedings. LNCS, vol. 1010, pp. 347–358. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60598-3_31

26. Yang, L., Shami, A.: On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. CoRR **abs/2007.15745** (2020). arXiv: 2007.15745

27. Yeguas, E., Luzón, M.V., Pavón, R., Laza, R., Arroyo, G., Díaz, F.: Automatic parameter tuning for Evolutionary Algorithms using a Bayesian Case-Based Reasoning system. Appl. Soft Comput. **18**, 185–195 (2014). https://doi.org/10.1016/J.ASOC.2014.01.032

Part III

Appendix

Appendix A

Curriculum Vitae (as of Jan. 2025)

Name	Maximilian Hoffmann
E-mail	hoffmannm@uni-trier.de
ORCID	0000-0002-7932-5822

Education

Ph.D. (Doctorate of Natural Sciences) 03/2020 – today
University of Trier
First reviewer and supervisor: Prof. Dr. Ralph Bergmann
Second reviewer: Prof. Dr. David B. Leake
Dissertation: Hybrid AI for Process Management - Improving Similarity Assessment in Process-Oriented Case-Based Reasoning via Deep Learning

M.Sc. Business Information Systems 10/2018 – 02/2020
University of Trier
Final grade: 1,5

B.Sc. Business Information Systems 10/2014 – 01/2019
University of Trier
Final grade: 1,6

Professional Career

Researcher 05/2021 – today
German Research Center of Artificial Intelligence (DFKI)
Branch University of Trier (Experience-Based Learning Systems)
Prof. Dr. Ralph Bergmann

Researcher 03/2020 –today
University of Trier
Department of Artificial Intelligence and Intelligent Information Systems
Prof. Dr. Ralph Bergmann

Student Researcher 02/2017 – 02/2020
University of Trier
Department of Artificial Intelligence and Intelligent Information Systems
Prof. Dr. Ralph Bergmann

Reviewing

International Conference on Case-Based Reasoning 2020-2024

	KI Conference	2020, 2021, 2023
	AAAI-MAKE Symposium	2021, 2022
	FLAIRS Conference	2023
	KnoSys Journal	2021
	LWDA Conference	2020
	IJCAI Workshop on DeepCBR	2021
Awards	Best Student Paper Award, FLAIRS Conference	2021

Appendix B

Complete List of Publications

Journals/Collections

Guldner, A., Hoffmann, M., Lohr, C., Machhamer, R., Malburg, L., Morgen, M., Rodermund, S.C., Schäfer, F., Schaupeter, L., Schneider, J., Theusch, F., Bergmann, R., Dartmann, G., Kuhn, N., Naumann, S., Timm, I.J., Vette-Steinkamp, M., Weyers, B.: A framework for AI-based self-adaptive cyber-physical process systems. it Inf. Technol. **65**(3), 113–128 (2023). https://doi.org/10.1515/ITIT-2023-0001

Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022). https://doi.org/10.3390/A15020027

Hoffmann, M., Malburg, L., Bergmann, R.: Augmentation of Semantic Processes for Deep Learning Applications. Applied Artificial Intelligence. Taylor and Francis. Submitted for Publication. (2024)

Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. **61**(1), 83–111 (2023). https://doi.org/10.1007/S10844-022-00766-W

Conference Proceedings

Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R.O. (eds.) Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings. LNCS,. Vol. 12311, pp. 229–244. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58342-2_15

Hoffmann, M., Bergmann, R.: Improving Automated Hyperparameter Optimization with Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings. LNCS,. Vol. 13405, pp. 273–288. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_18

Hoffmann, M., Malburg, L., Bach, N., Bergmann, R.: GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning. In: Keane, M.T., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development - 30th International

Conference, ICCBR, Nancy, France, September 12-15, 2022, Proceedings. LNCS,. Vol. 13405, pp. 240–255. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-14923-8_16

Hoffmann, M., Bergmann, R.: Ranking-Based Case Retrieval with Graph Neural Networks in Process-Oriented Case-Based Reasoning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133039

Kumar, R., Schultheis, A., Malburg, L., Hoffmann, M., Bergmann, R.: Considering Inter-Case Dependencies During Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Barták, R., Keshtkar, F., Franklin, M. (eds.) Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022 (2022). https://doi.org/10.32473/FLAIRS.V35I.130680

Malburg, L., Hoffmann, M., Trumm, S., Bergmann, R.: Improving Similarity-Based Retrieval Efficiency by Using Graphic Processing Units in Case-Based Reasoning. In: Bell, E., Keshtkar, F. (eds.) Proceedings of the Thirty-Fourth International Florida Artificial Intelligence Research Society Conference, North Miami Beach, Florida, USA, May 17-19, 2021 (2021). https://doi.org/10.32473/FLAIRS.V34I1.128345

Pauli, J., Hoffmann, M., Bergmann, R.: Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning Using Graph Neural Networks and Transfer Learning. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023). https://doi.org/10.32473/FLAIRS.36.133040

Schuler, N., Hoffmann, M., Beise, H., Bergmann, R.: Semi-supervised Similarity Learning in Process-Oriented Case-Based Reasoning. In: Bramer, M., Stahl, F.T. (eds.) Artificial Intelligence XL - 43rd SGAI International Conference on Artificial Intelligence, AI 2023, Cambridge, UK, December 12-14, 2023, Proceedings. LNCS,. Vol. 14381, pp. 159–173. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-47994-6_12

Schultheis, A., Hoffmann, M., Malburg, L., Bergmann, R.: Explanation of Similarities in Process-Oriented Case-Based Reasoning by Visualization. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development - 31st International Conference, ICCBR 2023, Aberdeen, UK, July 17-20, 2023, Proceedings. LNCS,. Vol. 14141, pp. 53–68. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-40177-0_4

Zeyen, C., Hoffmann, M., Müller, G., Bergmann, R.: Considering Nutrients During the Generation of Recipes by Process-Oriented Case-Based Reasoning. In: Cox, M.T., Funk, P., Begum, S. (eds.) Case-Based Reasoning Research and Development - 26th International Conference, ICCBR 2018, Stockholm, Sweden, July 9-12, 2018, Proceedings. LNCS,. Vol. 11156, pp. 464–479. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-01081-2_31

Workshop Proceedings

Brand, F., Lott, K., Malburg, L., Hoffmann, M., Bergmann, R.: Using Deep Reinforcement Learning for the Adaptation of Semantic Workflows. In: Malburg, L., Verma, D. (eds.) Proceedings of the Workshops at the 31st International Conference on Case-Based Reasoning

(ICCBR-WS 2023) co-located with the 31st International Conference on Case-Based Reasoning (ICCBR 2023), Aberdeen, Scotland, UK, July 17, 2023. CEUR Workshop Proceedings, pp. 55–70. CEUR-WS.org (2023)

Hoffmann, M., Bergmann, R.: Informed Machine Learning for Improved Similarity Assessment in Process-Oriented Case-Based Reasoning. CoRR **abs/2106.15931** (2021). arXiv: 2106.15931

Hoffmann, M., Malburg, L., Bergmann, R.: ProGAN: Toward a Framework for Process Monitoring and Flexibility by Change via Generative Adversarial Networks. In: Marrella, A., Weber, B. (eds.) Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6-10, 2021, Revised Selected Papers. LNBIP,. Vol. 436, pp. 43–55. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-94343-1_4