

Using Deep Reinforcement Learning for the Adaptation of Semantic Workflows

Florian Brand^{1,2}, Katharina Lott¹, Lukas Malburg^{1,2,*}, Maximilian Hoffmann^{1,2} and Ralph Bergmann^{1,2}

¹Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany

²German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, 54296 Trier, Germany

Abstract

Case-Based Reasoning (CBR) solves new problems by using experience represented by solved cases. The acquisition of adaptation knowledge and its subsequent application remains a classic challenge for CBR applications. In this paper, we present a novel approach for *adapting semantic workflows* during the reuse phase of the CBR cycle. A *reinforcement learning* agent is utilized, which applies different actions to change nodes of the workflow. Thereby, changes to the workflow are made by replacing, deleting or adding nodes. The agent is evaluated in a case study outlining its ability to adapt a semantic graph in a smart manufacturing domain. While the approach is detailed for the application in the particular domain, it can be adopted for the usage in other process-oriented domains.

Keywords

Case-Based Reasoning, Semantic Workflows, Deep Learning, Reinforcement Learning

1. Introduction

Case-Based Reasoning (CBR) [1] is a technique that uses experience of problems and their respective solutions to solve new problems. As a retrieved solution usually cannot be applied to a new problem out-of-the-box, it often needs to be adapted to fit a new upcoming problem. However, the acquisition and expression of adaptation knowledge is hard and expensive, which often leads to the adaptation step being left out entirely or leaving modifications to domain experts [2]. This is also known as the *adaptation knowledge bottleneck* [3]. To diminish this bottleneck, Machine Learning (ML) methods are popular in CBR research [4, 5, 6]. For instance, based on the difference between problem and solution (*case difference heuristic*) adaptation rules can be learned by a neural network for classification and regression domains, and subsequently applied [7, 8]. However, these methods have so far only been used for simple cases represented as attribute-value pairs [8] and their applicability to more complex case representations remains to be examined.

ICCBR BEAR'23: Workshop on Beyond Attribute-Value Case Representation at ICCBR2023, July 17 – 20, 2023, Aberdeen, Scotland


*Corresponding authors.

✉ s4fnbran@uni-trier.de (F. Brand); s4kalott@uni-trier.de (K. Lott); malburgl@uni-trier.de (L. Malburg); hoffmannm@uni-trier.de (M. Hoffmann); bergmann@uni-trier.de (R. Bergmann)

🆔 0000-0002-9253-9307 (F. Brand); 0000-0002-2664-036X (K. Lott); 0000-0002-6866-0799 (L. Malburg); 0000-0002-2458-9943 (M. Hoffmann); 0000-0002-5515-7158 (R. Bergmann)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

This work addresses the domain of Process-Oriented Case-Based Reasoning (POCBR) [9] where CBR methods are extended and applied to processes and workflows [9]. Such workflows that are usually represented as semantically annotated graphs [10] are utilized in various domains, for example as cooking recipes [11], scientific workflows [12] or smart manufacturing workflows [13]. Various adaptation methods have been proposed to learn adaptation knowledge for these workflows without relying on manual knowledge acquisition from domain experts [14, 12, 13].

However, due to the inherently complex graph structure in combination with domain-specific semantic information, these methods are not generic enough to be shared between different domains. To tackle this and to provide a more streamlined approach to graph adaptation, we propose a novel framework which combines ML methods with CBR, needing only minor modifications to work in different domains. We adapt semantic workflows with a Deep Reinforcement Learning (RL) agent performing structural modifications to the workflows. The proposed work is the first concept to combine ML approaches to adapt workflows in the context of POCBR. This paper proceeds as follows: In the following section, the semantic graph representation, RL, and related work regarding the adaptation of semantic graphs are presented. The proposed approach to adapt semantic workflows is introduced in Sect. 3 in the context of smart manufacturing workflows, which is further outlined in Sect. 4. Finally, Sect. 5 concludes the paper and gives an outlook for future work.

2. Foundations and Related Work

The following sections present an overview of the foundations for this work. Section 2.1 introduces the concept of semantic graphs as a representation of workflows. Section 2.3 gives an introduction to RL. We also cover related work regarding existing adaptation methods for graph-based data and applications of GNNs in CBR in Sect. 2.4.

2.1. Semantic Workflow Graphs

In this work, we workflows are represented as semantic graphs, also named *NEST* graphs [10], that allow modeling complex semantic information encoded and various types of nodes and edges. A *NEST* graph is a quadruple $G = (N, E, S, T)$ where N is a set of nodes and $E \subseteq N \times N$ represents the edges between nodes. Semantic descriptions S can be used for enriching individual nodes or edges with semantic information. T specifies the type of nodes or edges. An exemplary semantic workflow is depicted as *NEST* graph in Fig. 1. The graph illustrates a simplified sheet metal production process and consists of task nodes that describe the activities during manufacturing, e. g., drilling holes, data nodes that are used for representing the current state of the product, i. e., the properties of the sheet metal, and semantic descriptions enriching nodes with domain-dependent properties, e. g., parameters that specify a manufacturing operation such as quantity of drilling holes. To reduce the expressiveness of workflows and, thus, to enhance the use of AI methods, we restrict our contribution to block-oriented workflows [14, pp. 80]. A workflow is block-oriented, 1) if it features only a single start and end task node, 2) if all edges are properly connected, and 3) if every task node is connected with control-flow edges and data nodes with data-flow edges (see [14, pp. 80]

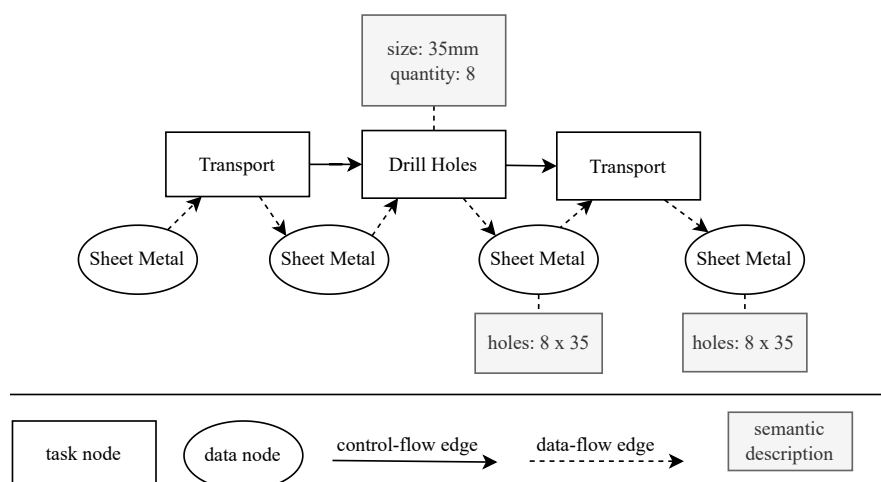


Figure 1: An Exemplary NEST Graph Showing a Manufacturing Process

for more details and the rules that must be satisfied for block-oriented workflows). The given workflow depicted in Fig. 1 is a block-oriented workflow, as 1) there is only one single start and end task node in the control-flow order, 2) all edges are properly connected, and 3) all task nodes are at least connected with one control-flow edge and the data nodes are also connected with at least one data-flow edge.

Based on the representation of workflows as semantic graphs, it is possible to assess the similarity during the retrieve phase in POCBR. For this purpose, a graph matching procedure is required, in which the nodes and edges of the query workflow are mapped to corresponding nodes and edges of the case workflow. If a mapping is found, the similarity is calculated based on the *local-global principle* [15, 10]: For this purpose, local similarity functions are defined with which the similarity between two nodes or edges is determined based on the semantic descriptions. These similarities are then aggregated into the global similarity by utilizing an aggregation function, e. g., a weighted average [10].

2.2. Methods for Workflow Adaptation

Adaptive workflow management [16] is an important topic for enabling flexible adaptations of workflows in changing environments. However, workflow adaptation is often performed manually by users, which makes it a complex and error-prone task. Recently, several methods based on CBR (e. g., [14, 17]) and other AI methods (e. g., [13]) have been proposed to support users in this process by performing adaptations in a semi-automatic or fully automatic way. In the following, we present some approaches used for adaptive workflow management.

One adaptation method is based on *generalization and specialization* [14]. In this approach, generalized cases are first learned by comparing similar workflows in the case base. Concrete nodes are generalized based on ontological knowledge, and a stored generalized case covers several concrete cases. Given the generalized cases, specialization is used to create new concrete cases that fit the requirements of the query. One drawback of this *substitutional adaptation*

method is that the workflow is not changed structurally. For this reason, *structural adaptation* methods exist with which the structure of the workflow is modified. One such method is the adaptation by *workflow streams* [14]. A workflow stream is a workflow fragment that is extracted to be reused in other workflows. To learn suitable adaptations automatically, workflow streams are created from the cases stored in the case base. Based on this learned knowledge, adaptations can be performed in a compositional way by replacing streams with other suitable streams. A similar adaptation method is the adaptation with *adaptation operators* [14]. In this context, adaptation knowledge is learned based on the cases stored in the case base. Each adaptation operator consists of small sub-workflows that describe insertions, deletions, or replacements. During adaptation, valid operators are retrieved and applied to the workflow. Thus, structural adaptations are possible by inserting new fragments into a workflow, by deleting fragments from a workflow, or by replacing fragments with other suitable ones. In general, these structural adaptations can be described at a higher level of abstraction by using *change patterns* [18]. Change patterns can be divided into two groups: *patterns for changes in predefined regions* and *adaptation patterns*. While the former allows to add information to workflows, the latter allows for structural changes in the workflow. These range from adding or replacing singular nodes or edges to moving, swapping, or replacing process fragments, which feature a part of the process consisting of multiple edges and nodes. Additionally, adaptation patterns support the adaptation of the control-flow, e. g., by making fragments parallel or looping a fragment.

2.3. Reinforcement Learning

Reinforcement Learning is a strategy for an agent to learn decision-making behavior based on trial-and-error experiences [19]. To train a machine learning agent to solve problems, a set of training data is first given. Here, the agent has a state in an environment and can choose actions according to its policy, from which a reward is generated to adjust the policy [20]. The key parts of a reinforcement learning model are the agent with its action space as part of the environment, the environment itself and its states in a state space, the policy with which the agent acts, the reward design of immediate feedback, and the value function to calculate long-term feedback [21].

The agent starts in an environment unknown to it. First, it observes its state in the environment. With its policy, the agent then chooses an action from the predefined action space to which it has access to and executes that action. Afterward, a reward or penalty is calculated as a direct result of the observation of the impact that the action had on the environment. This reward influences a value function that is the base for the agent's implicit policy for choosing actions. Next, the action observes its new state in the environment and the cycle starts again [19]. This is only broken if an end-state is reached or the training metric converges. This is defined as the end of an episode, with an episode being one training cycle. Reinforcement learning is episodic and therefore spans multiple training cycles, and the number of episodes has to be defined according to the specific use case. Many new RL approaches use neural networks as their policy due to their strong performance across many learning tasks (e. g., [22]).

2.4. Related Work

Related work to this work is based on existing solutions for adaptation in the context of POCBR [14, 12, 13], as well as an approach which combines machine learning and CBR [8]. Additionally, the usage of GNNs in CBR by Hoffmann and Bergmann [6] is of importance.

Ye et al. [7] propose a solution to combine machine learning techniques with CBR during the adaptation process by using a neural network that learns the case difference heuristic for problems and their respective solutions. The case difference heuristic generates adaptation rules which describe the needed changes in attributes to transform one solution into another [3]. The used neural network predicts the solution difference and passes it onto the CBR system, which then applies the difference [7]. This approach has been used for classification and regression tasks [7, 8], but in contrast to the prior methods it is unsuitable for graph adaptation as there is no suitable way of computing case differences for semantic graphs.

The combination of machine learning methods, graphs and CBR is shown by Hoffmann and Bergmann [6], who utilize different Graph Neural Networks (GNNs) to approximate the similarity of two semantic graphs during the retrieval step. This is done by transforming the nodes and edges of a graph into embeddings and training a GNN to calculate the similarity of the graphs. Their approach shows a greatly reduced effort to adapt to changes in the similarity definition or the domain models [6]. Additionally, it shows the successful application of GNNs in the CBR cycle, albeit in a different phase. We build upon the ability of these GNNs to compute embeddings as part of the policy of the RL agent.

3. Adaptation of Semantic Workflows with Reinforcement Learning

This section introduces our proposed approach that utilizes an RL agent to adapt semantic workflows. Section 3.1 gives an overview of the agent in the context of the CBR cycle and explains the different parts of the RL agent. The training setup is described in Sect. 3.2.

3.1. Architectural Overview and Approach

Figure 2 gives an overview of our approach. The starting point to the reuse step of the CBR cycle [1], where the proposed approach operates in, is the retrieved case in form of a workflow from the retrieve step. The retrieved case as well as the query case are then given to the RL agent to perform adaptation. As described in Sect. 2.3, an RL agent consists of three parts: An action space, a reward function, and an unknown environment, which the agent observes. The action space consists of different actions denoted in the form of change patterns, which are outlined in Sect. 2.4. The reward function consists of both an intermediate and a final reward after each episode. The environment consists of two parts: The retrieved case and the query. The goal of the adaptation is to modify the retrieved case in such a way that it satisfies the current problem expressed by the query (similar to [13]). Therefore, the agent adapts the case graph based on its reward function by applying the actions denoted in the form of change patterns [18] as introduced in Sect. 2.2.

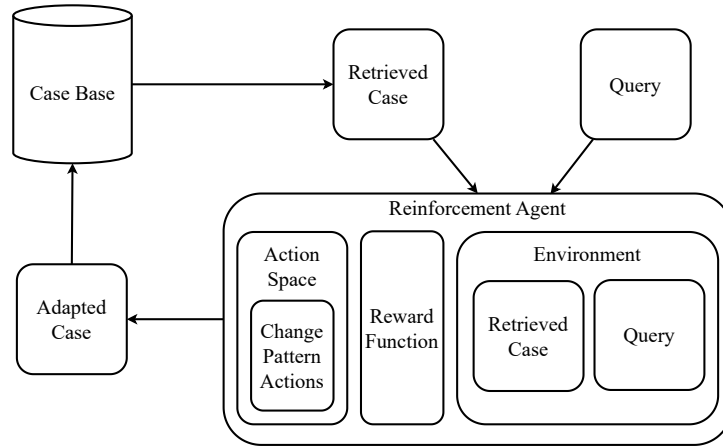


Figure 2: The Architecture of the Proposed Approach

Environment

The state of the agent is a single node on the graph to adapt, i. e., the *Retrieved Case* in the architecture shown in Fig. 2, starting on the first node of this graph (cf. [23]). After applying one of the possible actions described previously, the agent continues the traversal of the graph based on its control-flow. Additionally, the agent can also observe the *Query*. This is needed to let the agent adapt the case and, thus, to decide which action to apply based on the reward design.

Action Space

In this work, we use a subset of the change patterns as the notation of the actions. A change pattern has one of three operations: *add*, *delete*, or *replace* with one or two operands. The first operand is always a single node $n \in N_{WF}$ of the nodes of the workflow to adapt. The second operand is a single node $n' \in N$ with $n' \neq n$ or a set of nodes $N' \subseteq N \setminus \{n\}$ connected by edges E , with N being the set of all learned nodes. The *add* action adds one or more nodes after the given node in the control-flow of the workflow, while the *delete* action allows the deletion of a single node. The *replace* action allows the replacement of a single node with one or multiple nodes. Note that the sets of nodes (N_{WF} and N) contain both task nodes and data

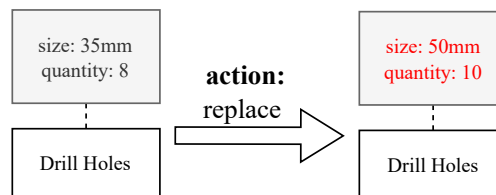


Figure 3: An Action Denoted in the Form of a Change Pattern

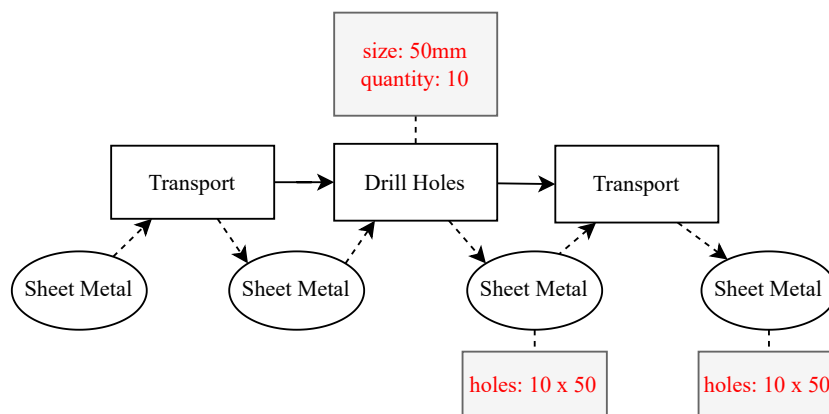


Figure 4: The Workflow From Fig. 1 After the Action From Fig. 3 Is Applied

nodes, so that every data or task node can be added, deleted or replaced by any corresponding node. Furthermore, we propose the addition of a *null action* that does not change a node so that the agent can decide to leave a node untouched. This is needed, as not every node has to be changed to accomplish the successful adaptation of a workflow.

An action in the form of a *replace* change pattern is shown in Fig. 3. Figure 4 shows the workflow from Fig. 1, which is introduced in Sect. 2.1, after the application of the action. As there exist numerous possible combinations of nodes with their semantic descriptions, the action space consisting of the nodes themselves would have to be classified as continuous [21]. We propose to learn the actions independently of the operands of the actions, i. e., learning the application of the change patterns independently of the node(s). This means, the agent has to select both from the discrete actions and the continuous operands as parameters, as seen in other approaches [24, 25, 22].

Reward Design

Both intermediate rewards after each step and final rewards after the traversal of the graph are used to steer the RL agent, similar to the approach proposed by You et al. [23].

The immediate rewards are the sum of domain-specific rewards and rewards based on the structure of the graph, i. e., whether the graph is a valid, block-oriented workflow [14]. We differentiate between domain-specific and structure-based rewards to make it possible to adapt this design to other domains. If the application of the action violates the structure of the graph, a large penalty is applied; otherwise, a small positive reward is assigned, similar to [23]. If, for example, a change pattern would add a second end node to the workflow, it would violate the block-orientation of the workflow and, thus, a large penalty is applied. Domain-specific rewards include rewards or penalties if domain-specific constraints are fulfilled or violated.

The final rewards also include a combination of domain-specific and domain-independent rewards. The domain-independent rewards feature the semantic similarity (see Sect. 2.1) between the query workflow and the case workflow [10]. If the similarity is the same or increases after adaptation, a positive reward is assigned; otherwise, a negative reward is assigned. As this

reward is computationally expensive and relies on an external system to calculate the similarity value, it is not feasible to use this as an immediate reward. Therefore, the reward is calculated at the end of an episode, i. e., after all actions are applied, and discounted to the prior steps afterward, similar to Darvariu et al. [26].

3.2. Training Setup

To train the model, reinforcement learning is used in the setup according to the previously defined guidelines to apply it to semantic workflow graphs and use it in the context of adaptation in a CBR system (see Fig. 5). As the reinforcement learning training can be sensitive regarding changes, the agent will start with a supervised pretraining similar to Jang and Kim [27]. For this, the agent gets pairs of semantic graphs (W, W'), where one is adapted from the other, as well as specific change patterns that were used to adapt them. Afterward, the RL training period can start. The training data is given in the form of pairs of semantic graphs (W, W') from the case base. Subsequently, these pairs of graphs are converted into embeddings and transferred to the agent [6]. Here, they can now be used as input for the training of the agent.

As shown in Fig. 5, the agent requires two graphs as input, one as the query graph and one as the case graph. The generated sets of graphs are utilized in these roles. Because reinforcement learning is more efficient with the use of more training data [28], each set of generated graphs can be utilized for training twice. This is achieved by swapping their assigned roles to act as both the query graph and the case graph for one training episode. The case workflow acts as the environment ϵ for the agent, with every node n being a state s . During training, the agent observes the current state of the environment, and with its policy, chooses an action from the predefined action list. The action space is modeled according to the earlier definition in Sect. 3.1.

After choosing and applying an action, the agent observes the changes in the environment and is given a reward or penalty based on the effects that the chosen action has (see Sect. 3.1). Therefore, every step, a large penalty or a small reward is calculated based on the structure of the graph. Additionally, domain-specific rewards or penalties are also calculated at every step. We propose to use a learning strategy based on a Deep Q Network (DQN) [29, 30]. This is a model-free approach that uses the Bellman equation [31] to calculate an estimated discounted

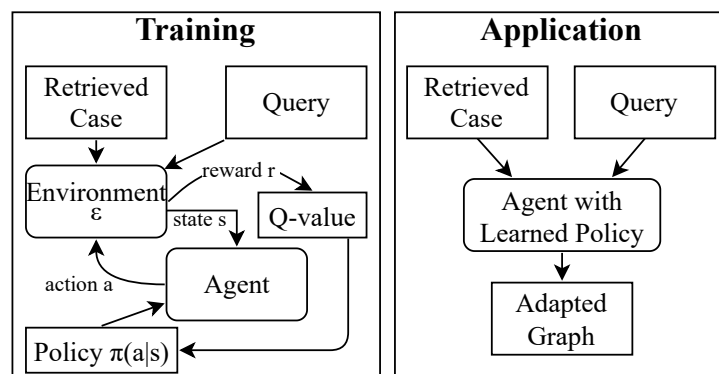


Figure 5: Training the Reinforcement Learning Agent

value (Q-value) for taking an action a at state s and following the policy π afterward. The goal of Q-learning is to maximize the Q-value by adding a learning factor α after getting the reward or penalty from the action taken [32]. The updates of the Q-function are therefore taken at each step with the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

With it, the old Q-value is updated by calculating the difference between itself and the discounted new value. To achieve that, a learning rate α is implemented to control the impact that the new Q-value has. Additionally, $\max_{a'}$ adds the maximum future reward from following the action to the new Q-value. Lastly, the discount rate γ is used as a way to balance the future reward [28].

Contrary to policy-based methods that greedily estimate the value of a policy based on the direct policy improvement, Q-learning cumulatively learns from an unknown environment over multiple episodes and saves the different values independently [29]. This also makes it an off-policy method where the policy is never directly altered, but instead implicitly derived from the Q-function [21], as can be seen in Fig. 5. It is preferred for this application because it is less sensitive to changes in the environment. Additionally, the approach is also used in other similar research approaches (e. g., [33]). After the Q-value of the short-term rewards is calculated, the agent observes the next state, which is always the next node in the graph, due to modifications only affecting the state at which the agent chose it.

In our context, an episode is defined as the agent having been in every state of the environment, so every node of the case workflow has been observed. After every episode, a final reward is calculated as described in Sect. 3.1 and then retrospectively applied to the Q-values of the previously taken actions. The training ends after a set number of episodes that need to be determined based on the quantity of the training data. If the number of episodes is too low, the risk of underfitting occurs where the agent is not trained with enough knowledge of the domain. Otherwise, if there is a small quantity of training data, but the number of episodes is very high, overfitting on those training examples is highly possible. It is therefore important that several training episodes are set in accordance with the training data available. After the initial training is completed, the agent can be used in the domain. In general, the agent should be routinely retrained, in case of changes in the domain or the training data.

4. Application Scenario: Adaptation of Smart Manufacturing Workflows

As a case study, we use a scenario based in the smart manufacturing workflow domain [13] (see Sect. 4.1 for an introduction) in a CBR system. In Sect. 4.2, we further describe the training of the RL agent in this domain. In Sect. 4.3, we show an exemplary application of the approach. Finally, we discuss the applicability to other domains in Sect. 4.4.

4.1. Smart Manufacturing Domain

One domain of semantic workflows is *smart manufacturing* [13]. Malburg et al. [13, 34] use semantic workflows to model manufacturing processes that are executed in a smart factory from Fischertechnik, which is shown in Fig. 6. The factory consists of two shop floors with six identical machines that are capable of performing various tasks, e. g., two *milling machines* (one on each shop floor) which can mill or drill workpieces. The machines are abbreviated based on the machine name and the shop floor they are on, i. e., the milling machine on the first shop floor is named *mm_1*, while the second milling machine is named *mm_2*. Additionally, the factory features several light barriers, switches, and sensors (see [34] for a detailed overview).

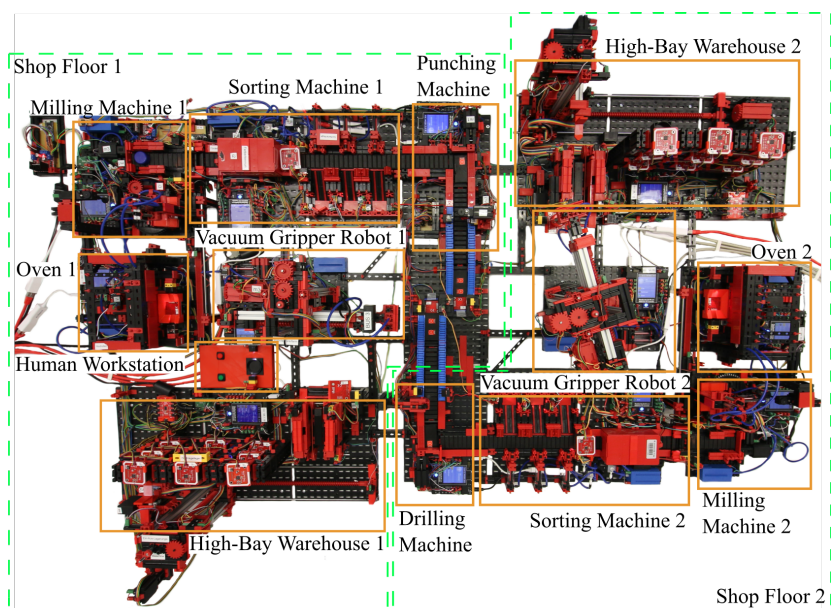


Figure 6: The Physical Fischertechnik Smart Factory Model [34]

Figure 7 shows an exemplary smart manufacturing workflow in the NEST graph format (see Sect. 2.1). The workflow describes the transport of a workpiece from the human workstation (*hw_1*) to the first milling machine (*mm_1*), which drills holes into the workpiece, followed by a transport to the third sink of the sorting machine (*sm_1*) on the first shop floor. Task nodes denote the production steps executed by actuators such as production machines in the physical factory. The semantic descriptors of the task nodes further specify the properties of each activity, e. g., the concrete parameters of the activity. In the depicted workflow, the *Drill Holes* task node is configured by the machine to execute the task, as well as the size and quantity of the holes to be drilled. Additionally, the state of the task is captured in the semantic description (*COMPLETED*, *ACTIVE*, *EXECUTABLE*, *FAILED* or *BLOCKED*). If a machine in the physical factory is broken, the task is blocked or failed. Data nodes represent the state of the workpiece throughout the production process, e. g., the position of the workpiece, or the amount of drilled holes. Hence, state changes of a workpiece are represented in the context of the execution of the workflow [13].

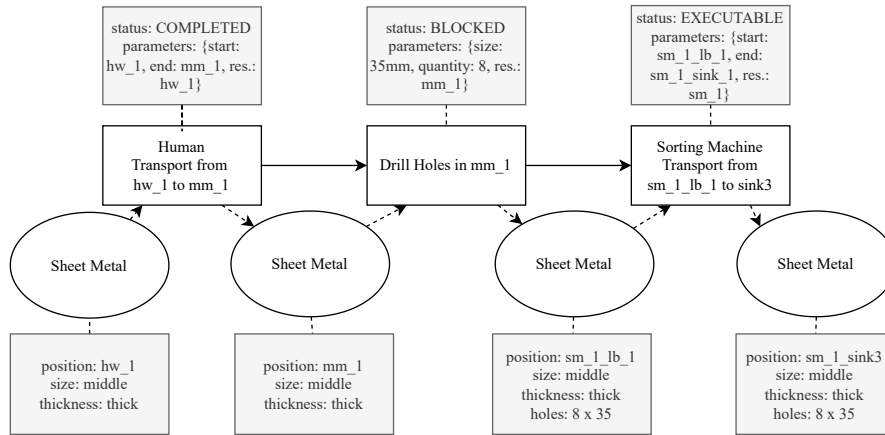


Figure 7: Graph Representation of a Smart Manufacturing Workflow

4.2. Training in the Smart Manufacturing Domain

To use the agent in this domain, it needs to be trained with the relevant information. The setup for the training follows the described methods in Sect. 3.2. First, the agent has a period of pretraining with pairs of semantic graphs and the change patterns that have been used to adapt them. These pairs can be obtained by applying different change patterns to a workflow and using the resulting outputs. Another possible approach is the usage of existing knowledge, e. g., by utilizing adaptations done by domain experts. All the parameters for the reinforcement learning agent are set as described in the training setup. For instance, a pair of graphs (W, W') is used as training input. W is a semantic graph with machine states containing a broken milling machine mm_1 , and W' is a second semantic graph that can be adapted from W by replacing task nodes that utilize the broken milling machine (see Fig. 8). The agent starts at the

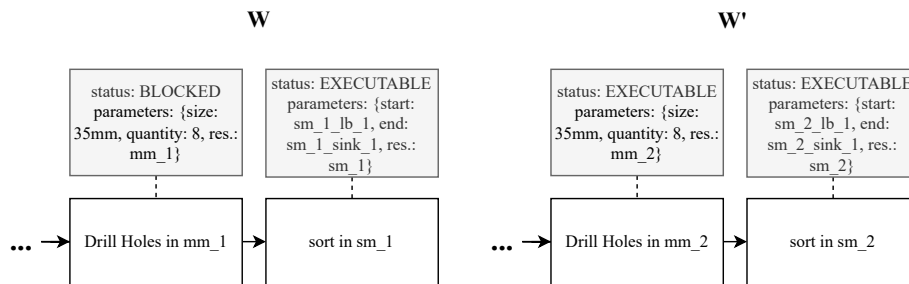


Figure 8: Abstracted Pair of Training Graphs

first node, going through the graph W one node at a time. Here, the agent chooses to take an action and is assigned rewards or penalties accordingly, as can be seen in Fig. 9. The change patterns will be stated in the action space exactly as they are presented in Sect. 3.1. This means that, at every node, the agent can choose one of four actions. To determine an operand for those actions that need it, the agent also stores possible existing nodes in the action space to use for

replacement or additions. Therefore, the action space is defined by the actions $replace(n_t, n'_t)$, $delete(n_t)$, $add(n_t, n'_t)$ and $null$ as operators, as well as all possible nodes N as operands, as outlined in Sect. 3.1. If, for example, the agent is at the second node of W it has the possibility to choose between $replace(Drill\ Holes\ in\ mm_1, n'_t)$, $delete(Drill\ Holes\ in\ mm_1)$, $add(Drill\ Holes\ in\ mm_1, n'_t)$ and $null$, with n'_t having to be chosen from the continuous action space as well.

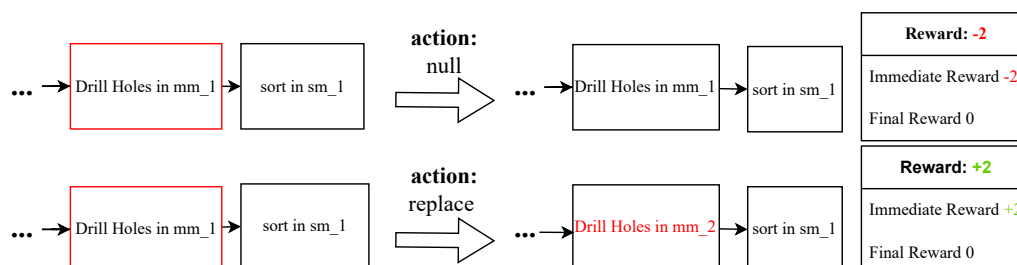


Figure 9: Abstracted Training Example

As stated in Sect. 3.1, for the reward design, domain-specific constraints need to be defined so that the reward is appropriate for the application. Here, for the smart manufacturing domain, a constraint is defined that only functional machines have to be used. If the agent uses an action which adds or keeps a broken machine, it is assigned a penalty. To continue the example, at the node *Drill Holes in mm_1*, if the agent decides on the *null* action, it is assigned a penalty because the domain-specific constraints specify that no broken machines have to be used and the action is still executed by the broken machine *mm_1*. Meanwhile, the decision for the action $replace(Drill\ Holes\ in\ mm_1, Drill\ Holes\ in\ mm_2)$ results in a positive reward because it complies with the domain-specific constraints and the graph is a block-oriented workflow [14]. The discounted final reward for each step will be retroactively applied after the whole training episode. In accordance with this, the agent adapts his policy to choose optimal actions. The training ends after a set number of training episodes. After the training episodes are finished, the agent is ready to be used in the domain for inference.

4.3. Application in the Smart Manufacturing Domain

To use the agent in an application context, the architecture from Fig. 10 can be used, which extends the architecture from Fig. 2. The cases consist not only of the workflows themselves, but additionally feature the states of the machines of the factory described in Sect. 4.1. The machine states contained in the query, i. e., the machine states at the point of process execution, are also part of the environment of the agent. This is needed to let the agent decide which action to apply based on the domain-specific reward design described in the previous section.

According to the training described in Sect. 4.2, the policy of the agent has been trained with a penalty for choosing any actions that result in the broken machine *mm_1* being utilized in the workflow. Due to this policy training, the agent switches or deletes the node that this machine is used in. Since the agent's policy has been trained with a reward on functionality, it does not delete the node. Instead, it decides on a node that is considered as similar as possible to the problem expressed in the query graph. Consequently, the agent finds a node with the

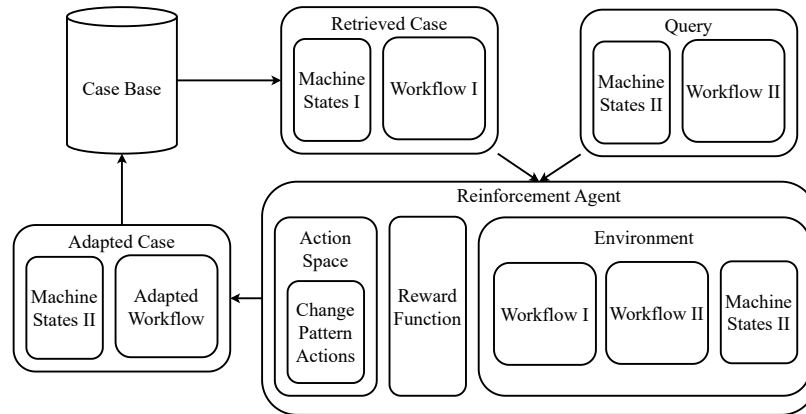


Figure 10: The Architecture for the Smart Manufacturing Domain

most similar functionality. Therefore, as seen in Fig. 11, at the first two states, it chooses the actions *replace(Human Transport from hw_1 to mm_1, Human Transport from hw_1 to mm_2)* and *replace(Drill Holes in mm_1, Drill Holes in mm_2)* to get rid of the task node that uses the broken resource. At state 3, which is the last node of this example, the agent chooses the action *replace(sort in sm_1, sort in sm_2)* in accordance with its policy, to keep the semantic logic of the workflow and adapt the case graph accordingly to the problem given in the query. With this, the adapted graph is built and introduced back into the CBR framework, where it can then be reused and stored in the case base.

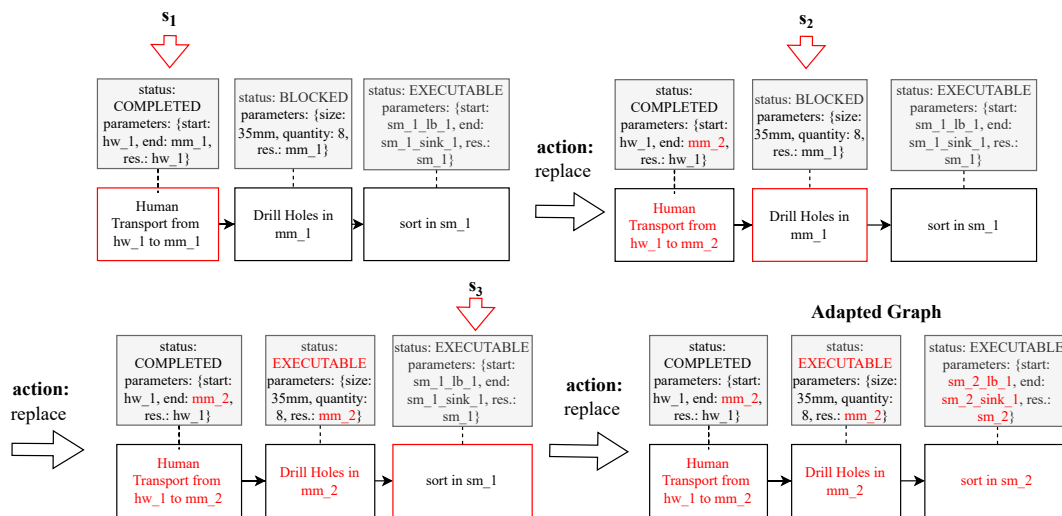


Figure 11: Abstracted Application Example

4.4. Applicability to Other Domains

To use this agent in other domains, the training data needs to be generated beforehand (see Sect. 4.2). The data needs to be properly defined as a set of workflow graphs. Additionally, some parameters of the setup need to be adapted for domain-specific circumstances. The agent can be used in a wide variety of domains, but it should be assured that the system can provide an output consisting of a query and a case graph. If this is not the case, it is important to change this for the agent before proceeding with the training. The change patterns need to be reviewed and changed if desired. If, for example, it is preferable to not add any new nodes to a graph, the according change pattern $add(n_t, n'_t)$ needs to be deleted from the action space. Additionally, for the reward design, specific domain constraints need to be defined so that the reward is appropriate for the application. For example, for cooking recipes, the constraints are not the usage of broken machines but the usage of undesired ingredients [11]. Consequently, a domain-specific reward can be applied if a desired ingredient is used during the adaptation. If there are no constraints in the domain, the domain-specific reward would be omitted in the reward design. It is crucial to design the reward according to the logic in the domain. A second part is the generation of training data and training of the agent. Once all the parameters are set up, it is important to get the needed data for training the model. If it is possible to get workflow pairs and the domain has both query and case graphs, it is possible to proceed as in Sect. 3.2. If there is no possibility to get pairs of workflows, the pre-training needs to be omitted. After setting this up, the agent is ready to be trained and used. Although it is still important to retrain the agent routinely, retraining is also dependent on the domain. Domains that have many changes will need retraining more frequently than others.

As the proposed approach is domain-agnostic, it can be applied to other domains, e. g., cooking recipes [11] or scientific workflows [12]. Section 4 demonstrates how to apply the concepts to the smart manufacturing domain.

5. Conclusion and Future Work

We present a novel approach which uses a *Deep Reinforcement Learning (RL)* agent for adapting semantic workflows. We propose to train the RL agent based on a Deep Q Network, which chooses actions denoted in the form of change patterns [18] to adapt the workflow by a GNN. These actions allow the addition, deletion, or replacement of nodes in the given workflow. The agent can then be used in the context of the CBR cycle to enable the adaptation of the workflows. Using a case study, we show the usability of this approach in the context of smart manufacturing workflows and outline needed changes to adapt the proposal to other graph-based domains.

In future work, we intend to implement the RL agent in the ProCAKE framework [35] and provide technical details of the implementation. Additionally, we aim at experimentally evaluating this implementation and comparing it to existing approaches in the context of smart manufacturing workflows (e. g., [13]). During this evaluation, we also want to further examine the transfer to other domains, such as the cooking recipes [11, 14] or scientific workflows [12].

References

- [1] A. Aamodt, E. Plaza, Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *AI Commun.* 7 (1994) 39–59.
- [2] B. Fuchs, J. Lieber, A. Mille, A. Napoli, Differential adaptation: An operational approach to adaptation for solving numerical problems with CBR, *Knowl. Based Syst.* 68 (2014) 103–114.
- [3] K. Hanney, M. T. Keane, Learning Adaptation Rules from a Case-Base, in: 3rd EWCBR, volume 1168 of *LNCS*, Springer, 1996, pp. 179–192.
- [4] K. Amin, S. Kapetanakis, N. Polatidis, K.-D. Althoff, A. Dengel, DeepKAF: A Heterogeneous CBR & Deep Learning Approach for NLP Prototyping, in: *INISTA*, IEEE, 2020, pp. 1–7.
- [5] X. Ye, D. Leake, D. Crandall, Case Adaptation with Neural Networks: Capabilities and Limitations, in: 30th ICCBR, volume 13405 of *LNCS*, Springer, 2022, pp. 143–158.
- [6] M. Hoffmann, R. Bergmann, Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning, *Algorithms* 15 (2022) 27.
- [7] X. Ye, D. Leake, V. Jalali, D. J. Crandall, Learning Adaptations for Case-Based Classification: A Neural Network Approach, in: 29th ICCBR, *LNCS*, Springer, 2021, pp. 279–293.
- [8] X. Ye, Robust Adaptation, Machine Learning Driven Adaptation, and Their Implications in Case-Based Reasoning, PhD Thesis (2022).
- [9] M. Minor, S. Montani, J. A. Recio-García, Process-oriented case-based reasoning, *Inf. Syst.* 40 (2014) 103–105.
- [10] R. Bergmann, Y. Gil, Similarity assessment and efficient retrieval of semantic workflows, *Inf. Syst.* 40 (2014) 115–127.
- [11] G. Müller, R. Bergmann, CookingCAKE: A Framework for the adaptation of cooking recipes represented as workflows, in: 23rd ICCBR Workshop Proc., volume 1520 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 221–232.
- [12] C. Zeyen, L. Malburg, R. Bergmann, Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning, in: 27th ICCBR, *LNCS*, Springer, 2019, pp. 388–403.
- [13] L. Malburg, F. Brand, R. Bergmann, Adaptive Management of Cyber-Physical Workflows by Means of Case-Based Reasoning and Automated Planning, in: 26th EDOC Workshops, volume 466 of *LNBIP*, Springer, 2023, pp. 79–95.
- [14] G. Müller, *Workflow Modeling Assistance by Case-based Reasoning*, Springer, 2018.
- [15] R. Bergmann, *Experience Management: Foundations, Development Methodology, and Internet-Based Applications*, Springer, 2002.
- [16] M. Reichert, B. Weber, *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*, Springer, 2012.
- [17] R. Bergmann, G. Müller, Similarity-Based Retrieval and Automatic Adaptation of Semantic Workflows, in: *Synergies Between Knowledge Engineering and Software Engineering*, *Adv. in Intell. Syst. and Comput.*, Springer, 2018, pp. 31–54.
- [18] B. Weber, M. Reichert, S. Rinderle-Ma, Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems, *Data Knowl. Eng.* 66 (2008) 438–466.
- [19] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, *J. Artif.*

- Intell. Res. 4 (1996) 237–285.
- [20] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, *IEEE Trans. Neural Netw.* 16 (2005) 285–286.
 - [21] R. S. Sutton, F. Bach, A. G. Barto, Reinforcement Learning: An Introduction, Adaptive Computation and Machine Learning series, MIT Press, 2018.
 - [22] M. J. Hausknecht, P. Stone, Deep Reinforcement Learning in Parameterized Action Space, in: 4th ICLR, 2016.
 - [23] J. You, B. Liu, R. Ying, V. S. Pande, J. Leskovec, Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation, *CoRR abs/1806.02473* (2018).
 - [24] W. Masson, G. D. Konidaris, Reinforcement learning with parameterized actions, *CoRR abs/1509.01644* (2015).
 - [25] A. Delarue, R. Anderson, C. Tjandraatmadja, Reinforcement Learning with Combinatorial Actions: An Application to Vehicle Routing, in: *Adv. Neural Inf. Process. Syst.*, volume 33, Curran Associates, 2020, pp. 609–620.
 - [26] V. Darvariu, S. Hailes, M. Musolesi, Improving the Robustness of Graphs through Reinforcement Learning and Graph Neural Networks, *CoRR abs/2001.11279* (2020).
 - [27] S. Jang, H.-I. Kim, Supervised pre-training for improved stability in deep reinforcement learning, *ICT Express* 9 (2023) 51–56.
 - [28] Y. Li, Deep Reinforcement Learning: An Overview, *CoRR abs/1701.07274* (2017).
 - [29] C. Watkins, Learning from Delayed Rewards, PhD Thesis (1989).
 - [30] B. Matzliach, I. Ben-Gal, E. Kagan, Detection of Static and Mobile Targets by an Autonomous Agent with Deep Q-Learning Abilities, *Entropy* 24 (2022) 1168.
 - [31] R. Bellman, The theory of dynamic programming, *Bull. New. Ser. Am. Math. Soc.* 60 (1954) 503–515.
 - [32] C. Watkins, P. Dayan, Q-Learning: Technical Note, *Mach. Learn.* (1992) 279–292.
 - [33] V. Mnih, et al., Human-level control through deep reinforcement learning, *Nature* 518 (2015) 529–533.
 - [34] L. Malburg, R. Seiger, R. Bergmann, B. Weber, Using Physical Factory Simulation Models for Business Process Management Research, in: 18th BPM Workshops, volume 397 of *LNBIP*, Springer, 2020, pp. 95–107.
 - [35] R. Bergmann, L. Grumbach, L. Malburg, C. Zeyen, ProCAKE: A Process-Oriented Case-Based Reasoning Framework, in: 27th ICCBR Workshops, volume 2567 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 156–161.