Towards Reinforcement Learning for In-Place Model Transformations

Martin Eisenberg, Hans-Peter Pichler, Antonio Garmendia, Manuel Wimmer
CDL-MINT, Institute of Business Informatics - Software Engineering
Johannes Kepler University Linz
Linz, Austria
{firstname.lastname}@jku.at

***Abstract*—Model-driven optimization has gained much interest in the last years which resulted in several dedicated extensions for in-place model transformation engines. The main idea is to exploit domain-specific languages to define models which are optimized by applying a set of model transformation rules. Objectives are guiding the optimization processes which are currently mostly realized by meta-heuristic searchers such as different kinds of Genetic Algorithms. However, meta-heuristic search approaches are currently challenged by reinforcement learning approaches for solving optimization problems.**

In this new ideas paper, we apply for the first time reinforcement learning for in-place model transformations. In particular, we extend an existing model-driven optimization approach with reinforcement learning techniques. We experiment with value-based and policy-based techniques. We investigate several case studies for validating the feasibility of using reinforcement learning for model-driven optimization and compare the performance against existing approaches. The initial evaluation shows promising results but also helped in identifying future research lines for the whole model transformation community.

***Index Terms*—Model Transformations, Reinforcement-Learning, Model-based Optimization**

I. Introduction

In recent years, Artificial Intelligence (AI) methods have been applied for model transformations (MTs) [1]–[8]. Specifically, model-driven optimization has recently gained much interest [1]–[6]. The core idea is to optimize models by applying a set of MT rules. This process is usually guided by meta-heuristic searchers such as Genetic Algorithms (GA) [1]–[6]. However, meta-heuristic search approaches are currently challenged by Reinforcement Learning (RL) approaches for solving particular optimization problems. RL introduces the concept of an agent. This agent must learn a corresponding behaviour through trial-and-error interactions [9].

There is already some work that explore RL for specific MT problems. For instance, the work of Iovino et al. [8] applies RL to the specific problem of model repair. However, to the best of our knowledge, there is a lack of research on how in-place MTs in general may benefit from RL.

In this new idea paper, we apply for the first time RL to general in-place MTs. In particular, we extend an existing model-driven optimization approach for including RL approaches, considering both, value-based and policy-based approaches. We investigate several case studies for validating the feasibility of using RL for in-place MTs and compare the performance against an existing search-based approach. The initial results show potential of applying RL for in-place MTs, especially when combined with search-based techniques. We provide our solution as an open source project [10]. Finally, based on our experiences and initial results, we outline several future research lines for the model transformation community.

The remainder of this paper is structured as follows. In Section II, we introduce the background for our work, i.e., MTs and RL. Then, we present our approach on how to combine RL and MTs in Section III. Section IV presents some initial experiments that show the feasibility and performance of our approach. Finally, we conclude this paper with a roadmap for future research of the application of RL for MTs.

II. Background

We now present the background and related work on MTs as well as a short introduction to RL.

A. Model Transformation Basics

MT is a key technique in MDE [11], [12]. Generally, MTs are used to create new models from existing ones (out-place MTs) or modify existing ones directly (in-place MTs) [13]. In this paper, we consider in-place MTs.

As a running example, we consider the use of a simplified Pacman game [14], [15]. Fig. 1 shows the meta-model excerpt for this example. Based on [15], we reduce the game features by just modeling grid nodes where the Pacman moves searching for food, and at the same time, avoids running into a ghost. In addition, each time the Pacman eats food, the score value will be incremented. To illustrate in-place MTs, we choose Henshin [16] as it will be also used for our prototype. However, the present approach is conceptually not specific to Henshin. Henshin provides a rule-based MT language. Fig. 2 shows the application of an example Henshin rule to a Pacman model. At the top, the input model is shown (Pacman is on grid 1, food on grid 3). For this model, the Henshin rule *moveRight* may be applied. The result is shown at the bottom of the figure. It can be observed that the referenced element of reference on changed from 1:GridNode to 2:GridNode. Playing the Pacman game in this setting is about finding the best sequence of rule applications to maximise the score value.

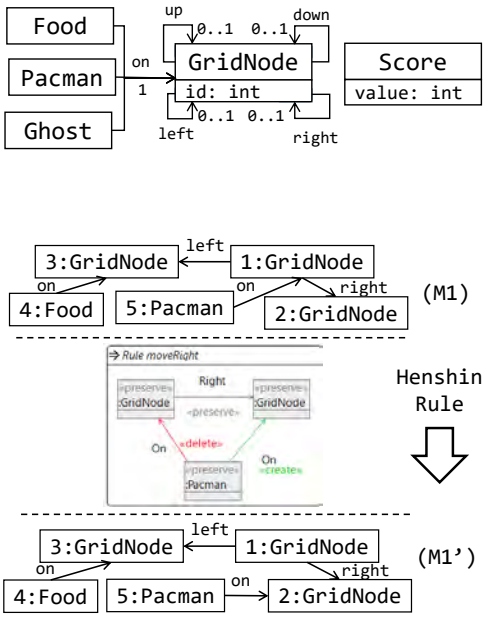

Fig. 2. Example Henshin rule.

B. Related Work

In recent years, search-based orchestration of transformation rules has been explored. For instance, the Transformation Tool Contest (TTC) 2016 proposed the Class Responsibility Assignment (CRA) problem in which a total of eight submissions were accepted[1]. Most of these proposals rely on the use of meta-heuristic search, e.g., GA. General frameworks such as MDEOptimiser (MDEO) [2], VIATRA Optimiser [6], and MOMoT [1] have been proposed which use a set of transformation rules and a set of objectives taken into consideration when searching for good rule application sequences.

In addition to search-based approaches, there exist work that applies RL to MDE tasks. Specifically, Iovino et al. [8] use RL for model repair, i.e., to find a quality repair action for each error in the model. Barriga et al. [17] perform a comparative study of RL techniques for this problem. Regarding MTs, Burgueño et al. [7] employ an AI method for automatically building out-place MTs. They train a neural network using a data set of input/output pairs.

C. Reinforcement Learning

As a major branch of machine learning (ML), RL techniques consider problems where one ought to learn a behavior that maximizes a reward signal [9]. In sequential decision problems, an agent interacts with an environment and receives feedback on the interaction choices in terms of a reward function. The agent operates and intends to learn a behavioral pattern—a so-called policy—that defines which step to take in a particular situation to end up with the most beneficial outcome. This behavioral pattern can be derived after observing the advantage of being in each state. The benefit of taking a certain action in a particular state is given by the reward function.

[1]http://www.transformation-tool-contest.eu/2016/solutions_cra.html

Initially, there is no knowledge base for the agent to estimate the consequences of decisions [9]. Only after performing actions and receiving rewards, the agent learns how fruitful particular actions are in a certain state and combines this information with the benefit of ending up in a respective successor state. Relating this to the Pacman game, ideally, the Pacman follows the path that not only increases the score in the short term, but turns out most promising for scoring points in upcoming steps as well. For this, the fusion of immediate reward and potential future rewards needs to be considered. In the long run, the agent converges towards a policy with the sequence of actions that yields the highest achievable reward in the explored environment.

ML approaches such as RL may introduce novel aspects to optimization that are not in the scope of traditionally used Evolutionary Algorithms (EAs) such as GAs. For instance, solutions produced by the latter algorithms may become obsolete after changing the initial problem instance. Instead, RL methods may reuse their gained experience for future decision processes in the same problem domain without having to start the exploration of the search space from scratch. Another point is the fixed length for the individual solutions in a population [18] when using GAs. This is not necessary for RL agents which may use more or less steps before ending a learning episode. Additionally, crossover in GAs may be complicated if there are dependencies between consecutive steps as is the case for MT steps [1]. In fact, promising results with deep RL were presented for particular optimization problems [19]. Considering preference-based multi-objective optimization, algorithms as the one presented by Yang et al. [20] could omit the need for prior setting of preferences to guide the search as the trained model provides optimal solutions for all possible preferences. Overall, EAs and RL techniques appear to shine under different aspects and their proper selection as well as their synergy constitute a promising research endeavour.

D. Synopsis

To the best of our knowledge, the search-based exploration driven by RL methods in MTs has not been considered so far. However, other works formulate specific optimization tasks in a form solvable with RL methods already, but they are missing a generic formalism that maps the necessary concepts of a RL problem to the components of a rule-based MT paradigm. Therefore, we derive such a formalism and employ two RL types, namely value-based and policy-based, to demonstrate the feasibility of this work.

III. RL for In-place MTs

Based on the previous discussions, we now present our approach for applying RL for in-place MTs.

A. Approach at a Glance

In order to employ RL methods, the task of finding valuable MTs can be formulated by means of a Markov Decision Process (MDP) [21]. Generally, the MDP is defined by a set

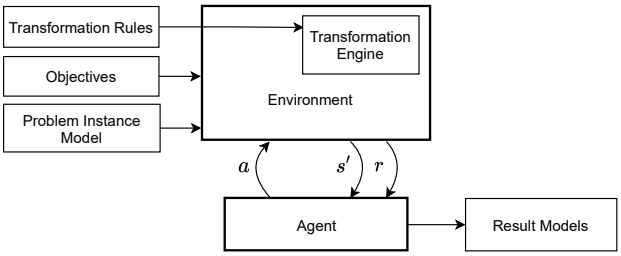

Fig. 3. Integration of RL into MT Frameworks.

of states S in an environment, a set of actions $A(s)$ executable in a given state s, a probability distribution $P(s'|s,a)$ over all possible successor states when performing action a in state s, and a reward function $R(s,a,s')$ to asses the benefit of performing action a in state s and ending up in state s'.

Fig. 3 depicts the MDP adopted to rule-based in-place MTs. The agent interacts with the environment as it selects a rule a to which the environment responds with the transformed model s' as a result of applying a on the previous model state. Furthermore, a reward r hints at the value of selecting rule a. In order to find an optimized version of a model, the agent needs to choose rules successively that, when executed on the initial model, produce the result models that optimize the target objectives. In this sense, the states are models in their current composition, the actions are rule applications that modify a models composition. Such modifications may be feature changes or adding/removing model elements based on a predefined set of rules. The reward can be assessed as the change in the objective value after the application of a rule on a model. Rewards are to be maximized, hence, in case of a minimization target, the additive inverse of the fitness value needs to be maximized. Since there is no uncertainty involved in the MT process, i.e., the successor model s' after executing a rule a on a current model s is always identical, the MDP is deterministic. Hence, there is no probability distribution over a set of possible successor models.

Further specifications need to be considered in the search process as follows. First of all, MTs themselves need to be valid by applying feasible changes to the model. Second, the transformed model should represent a valid solution to the given problem. The implementation of T to control the number of MT steps is arbitrary. However, for the comparison in Section IV, the termination criterion T corresponds to the specified transformation lengths TL_{max} per case that denotes the solution length of the GA.

The primary endeavour is to find a rule application sequence that optimizes the objectives for the resulting model. Additionally, minimizing the number of transformation steps required to obtain the optimized target model is desirable, e.g., to consider resource-intensive operations. In essence, this poses a multi-objective optimization problem. However, RL-based agents do not treat both objectives as such, but rather optimize the primary objective while considering less steps to achieve the same primary objective fitness as superior. The extent to which future steps in the transformation chain are considered is determined with the discount factor γ.

B. Value-based Learning

Value-based methods are model-free RL approaches that try to derive a value function from the obtained rewards of actions within certain states. In the variant of Q-Learning, the goal is to approximate the action-value function $Q(s,a)$ using the Bellman equation [9], [22]. In case deterministic policies are applicable, value-based methods are often advantageous because of their simplicity during implementation and efficiency.

In the context of MT and Q-Learning, the agent has to remember the encountered model states in a Q-table that consists of applied rules, parameter values, and the corresponding Q-values. Consequently, we store a map $Q(s,a)$ of applied rules and parameter values. By remembering the ordered transformation sequence that lead to a particular model, it can be reconstructed using the initial model when needed.

Algorithm 1 Q-Learning for In-place MTs

1: Initialize parameters γ, ϵ
2: Initialize environment *ENV* with initial model s_0, reward
3: function $R(s,a)$, and transformation engine
4: Initialize $Q(s,a) \leftarrow \{\}$
5: **for** (max. number of evaluations)
6: Initialize s $\leftarrow$ *ENV-RESET()* ▷ reset to initial state s_0
7: **while** (not T) ▷ termination criterion
8: **if** (rand(0,1) $\leq$ ϵ), Select next rule a = *SEARCH(s)*
9: **else**, Select next rule $a = \text{argmax}_a\ Q(s,a)$
10: r, s', T $\leftarrow$ *ENV-STEP*(a) ▷ apply rule a, observe reward r from $R(s,a)$, result model s', and if T satisfied
11: $Q(s,a) \leftarrow Q(s,a) + (r + \gamma * \max_a Q(s',a) - Q(s,a))$
12: $s \leftarrow s'$
13: **end while**
14: **end for**

Algorithm 1 adopts Q-Learning [22] for in-place MTs. Parameters for Q-Learning are: discount factor γ and exploration probability ϵ. The RL environment is initialized with the initial state s_0, the reward function $R(s,a)$, and an interface to consult the transformation engine. The reward function $R(s,a)$ provides the objective fitness value evaluated on the result model s' after applying rule a on a current model s.

For the conducted evaluation, we implemented two general Q-Learning agents, Q_{Basic} and $Q_{Explore}$, both following the epsilon-greedy action-selection strategy in Algorithm 1 lines 8-9. They differ in their capability to explore neighbor solutions with *SEARCH(s)*. In the case of Q_{Basic}, the next rule a is chosen randomly by the transformation engine. For $Q_{Explore}$, a local search is performed to evaluate a set of possible transformation rules and select the one that maximizes the reward r. The Q-table update using the Bellman equation happens on line 11.

C. Policy-based Learning

Policy-based methods are also model-free RL approaches that aim to directly learn a policy through function estimation

without consulting a value function. This function estimation can be done by using an artificial neural network (ANN) that aims to maximize the expected reward [9], [23]. In contrast to the afore presented general value-based approach, we perform a specific application of the policy-based approach for one MT and leave its generalization as subject to future work.

In particular, we apply the Policy Gradient (PG) Theorem with the REINFORCE [23] algorithm to the MT-based Pacman game. The PG agent learns a parametrized policy function to select one of the four possible step directions. Due to the use of an ANN and a gradient ascent learning strategy for finding the policy function, an appropriate encoding of the model state is necessary. The transformation of models into a format applicable to ANNs demands a proper encoding as the one presented by Burgueño et al. [7]. Therefore, we stick to a simple one-hot encoding of the board state and additional information such as the distance to the next food in each direction. Using a model containing a 8x8 play grid, the ANN receives already 201 inputs and provides a probability distribution over four outputs with softmax activation in the output layer. For the reward function, the agent receives a reward of 30 and -150 whenever the eat or kill rule is applied, respectively. Each step yields a negative reward of -5 and eating the last food on the grid issues an additional reward of 150. There are ten nodes with food and three with a ghost.

D. Prototypical Implementation

We use the open-source library DeepLearning4J [24] together with ND4J [25] to implement the PG approach. The RL algorithms are embedded in the MOMoT [1] framework (based on EMF and Henshin) which is used to conduct the case studies in the following section. Our tool is available at [10].

IV. Evaluation

We now apply the presented approach for several cases by following the case study research methodology [26].

A. Research Questions

RQ1: Applicability: Are RL methods suitable for solving in-place MTs in general? In particular, we are interested in the limitations in applicability of value-based and policy-based approaches as well as in the required preparation steps needed to enable their usage.

RQ2: Performance: What are the advancements for MTs brought forth by RL methods in terms of objective optimization? In particular, we are interested in comparing RL with search-based approaches to understand their (dis-)advantages.

B. Setup

To answer the RQs, we evaluate the value-based RL approach described in Section III-B on three cases and compare their performance to the Non-dominated Sorting Genetic Algorithm II (NSGA-II) as it is encoded in the MOMoT framework [1]. In addition, to answer RQ1 concerning value-based vs. policy-based RL, we use our running example (Pacman) to compare the PG agent with Q_{Basic}. In the following, we introduce the cases, discuss the metrics used for evaluation, the algorithm-specific settings, and the hardware/software used to conduct the experiments.

1) Cases: **Stack Load Balancing [1]**: The *StackModel* consists of multiple stacks that are connected in a circular way with each stack having a number of loads assigned. With the rules *shiftLeft* and *shiftRight*, a stack can send a load amount to one of it's neighbours. In our experiments, we use models with varying number of stacks and loads and intend to balance the loads among all stacks, i.e., minimize the standard deviation of stack loads.

Class Responsibility Assignment (CRA) [1]: In this problem, the aim is to group features into classes to produce high-quality object-oriented models. With a single rule *reassignFeature*, the features are distributed among classes to maximize intra-class (cohesion) and minimize inter-class (coupling) dependencies whereas we combine both metrics in the objective *CRA-Index* ought to be maximized. The problem instances assume each feature to be assigned to one class in the beginning and vary in complexity with the number of features.

OO-Refactoring [27]: Here we aim to optimize an object-oriented model by removing duplicate attributes from classes which may be collected in superclasses instead. Consequently, *createRootClass* introduces a new class C with an attribute a that was previously declared in two subclasses that henceforth inherit this property from C. The rule *extractSuperClass* adds a level to the inheritance hierarchy: A new class C is introduced and becomes the superclass of classes that share an attribute a and previously derived from a class S. Class C then declares a instead of its subclasses and inherits from S. The goal is to minimize the sum of entities and properties.

Pacman: A simplified version of the Pacman game adopted from Heckel [14]. The PacMan moves on the grid from node to node whereby each node is either empty, holds food, or is inhabited by a ghost. Upon encountering a node with a food, the *eat* rule is applied and the *score* increased. Entering a node with a ghost ends the game. The goal for the Pacman is to maximize the score by finding all food pieces.

2) Evaluation metrics: To assess the performance, we calculate the hypervolume (HV) on the objectives of the result models produced by the algorithms. HV is a set measure for the extent to which the objective value space is covered by a Pareto front approximation in relation to a reference set [28]. Commonly used in search-based software engineering, HV as the only indicator among many reflects the four qualities convergence, spread, cardinality, and Pareto compliance [29]. Furthermore, HV has a bias towards knee points in non-dominated solutions, i.e., solutions where the increase in one objective severely decreases fitness of at least another one [30]. The authors of [29] recommend the use of HV for comprehensive evaluation of Pareto sets with less than 10 objectives. As there is no consensus on how to set the reference points when there is no preference, e.g., for extreme solutions, we take the combined non-dominated front of all sets in the respective experiment as reference set. Additionally, we are interested in the total optimization quality delivered by each

optimizer and compare the best obtained objective fitness value (BOV) found among the result models.

3) Parameter settings: For NSGA-II, which we consider as a reasonable baseline for comparison, we set the population size to 100 individuals in general and, to deal with the high computational overhead, to 15 for the Refactoring case. For CRA problem cases with $TL_{max} = 160$ and $TL_{max} = 320$, we increase the population to match TL_{max} as to not limit the size of the Pareto solution set. We perform crossover for all descendants in a generation and set the mutation probability for rules, concerning either changing parameter values of a rule or removing a rule, to 0.1 and 0.2, respectively.

We use a constant discount factor γ = 0.9 for the RL agents to prioritize optimization over longer sequences and start the search with ϵ = 0.9 to focus on exploration in the beginning. A constant ϵ_{decay} decreases ϵ each time the agent explores with ϵ_{decay} = 0.001 until ϵ reaches a minimum of 0.1, except for the Refactoring case where only 1000 models are evaluated per run to deal with the high computation overhead, we use ϵ_{decay} = 0.1.

4) Used hardware: We use: a virtual machine of type E2 on the Google Cloud Platform[2] running a Intel(R) Xeon(R) CPU @ 2.20GHz with up to 16 cores on a Debian GNU/Linux 10 OS, and 128GB physical memory; Java 14.0.2 with an initial heap size of 1GB and allow the use of up to a maximum of 100GB in the Eclipse version 4.19; Henshin SDK 1.6.0, Eclipse Modeling Framework 2.25, and MOMoT 2.0.0.

C. Results

We now present the results of applying the RL implementations against NSGA-II (MOMoT encoding) on the first three cases. Experiments are executed 30 times whereby one execution comprises of 100,000 solution evaluations for Stack and CRA, and 1000 evaluations for the Refactoring case. The average performance and best overall solutions are reported in Tab. I. Subsequently, we discuss the comparison of the value-based vs. policy-based approach for the Pacman case.

1) Stacks: In the simplest case with five stacks, all algorithms find the optimal distribution of loads after three moves although only $Q_{Explore}$ manages to do so consistently over all runs. By doubling up the number of stacks, $Q_{Explore}$ finds the lowest objective value of 0.4 with 14 moves and shows overall higher average HV than NSGA-II. Q_{Basic} cannot compete with increasing complexity. In the cases of 25 and 50 stacks, NSGA-II and $Q_{Explore}$ perform about equally well in terms of avg HV. However, $Q_{Explore}$ provides higher quality given that the focus is on the best objective values.

2) CRA: Only in the case with 9 features, Q_{Basic} shows higher average performance than NSGA-II. In all other cases, Q_{Basic} performs much worse than NSGA-II and $Q_{Explore}$. With increasing complexity, the latter shows the best performance on avg HV and concering BOV as well as appears with less variation between independent runs.

[2]https://cloud.google.com/compute/docs/machine-types

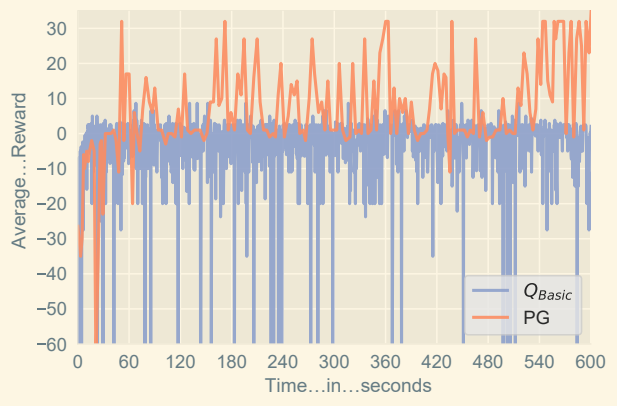

Fig. 4. Average reward returned from the RL algorithms.

3) Refactoring: The BOVs are equal for the $Several$ case, but not on par for $Pullup$ as Q_{Basic} fails to find the optimum when attributes are to be repeatedly moved up the inheritance hierarchy. Also, HV results indicate that the exploration capability of Q_{Basic} suffers from the reduced number of evaluations. Considering the standard deviation, $Q_{Explore}$ and NSGA-II perform on the same level.

4) PacMan: Fig. 4 depicts the avg reward over time that is returned every 10 episodes. It shows the superiority of a domain-specific encoding for the model states. The PG receives much higher average rewards after a minute of learning time as it is able to learn a reasonable policy to navigate the Pacman on the grid. Q_{Basic} struggles to adapt its behavior towards finding nodes with food. It fails to avoid terminal, and thus, highly undesirable states as shown with the large negative rewards it receives over the whole runtime.

D. Discussion

Answering RQ1: The presented results demonstrate the feasibility of RL for in-place MTs. The value-based RL methods fit the needs of single objective, in-place MTs where the number of transformation steps matters. Our Q-Learning implementations offer general applicability but face limits when the search space is huge as demonstrated with the Pacman case. Furthermore, value-based methods store all seen transformations, thus imposing a memory condition. The policy-based approach learns to choose the proper rule applications that maximize the score as the Pacman is guided towards rewarding game states. Moreover, storing transformations is not required since a parametrized policy function is learned to select desirable transformations. However, such a neural network based approach currently requires a specific encoding of the model states and a dedicated reward function.

Answering RQ2: In our experiments, with RL, the resulting models provide a similar or better approximation of the Pareto front provided that an extensive exploration phase is enabled. This is observable with our $Q_{Explore}$ variant which finds the highest quality models in terms of BOV. Expanding rule by rule seems to benefit the optimization at different stages and the convergence towards a global optimum. Nevertheless, the

TABLE I

MEAN HYPERVOLUME (AVG), STANDARD DEVIATION (STD) AND BEST OBJECTIVE VALUE (BOV) OBSERVED OVER 30 RUNS FOR Q_{Basic}, $Q_{Explore}$, AND $NSGA-II$ (WITH MOMOT ENCODING). THE ARROW NEXT TO THE CASE NAME INDICATES THE OPTIMIZATION GOAL. BOLD NUMBERS MARK THE BEST MEAN AND BOV FOR EACH SUB-CASE.

Case	TL_{max}	Complexity	Q_{Basic}				$Q_{Explore}$				NSGA-II			
			Avg	Std	BOV	TL	Avg	Std	BOV	TL	Avg	Std	BOV	TL
Stack (↓)	8	5 Stacks	.266	.024	0	3	**.287**	0	0	3	.278	.027	0	3
	16	10 Stacks	.523	.032	0.98	11	**.644**	.01	**0.4**	14	.607	.039	0.75	8
	40	25 Stacks	.229	.014	10.24	39	.559	.005	**5.052**	40	**.575**	.015	6.23	29
	80	50 Stacks	.129	.007	26.87	80	**.538**	.007	**18.88**	80	.513	.029	21.08	51
CRA (↑)	18	9 Features	.366	.018	3.0	5	**.382**	.005	3.0	5	.337	.052	3.0	5
	36	18 Features	.444	.027	-1.0	20	**.675**	.014	**4.083**	12	.541	.055	2.0	11
	70	35 Features	.429	.015	-17.58	66	**.761**	.009	**-0.2**	66	.609	.044	-7.56	23
	160	80 Features	.325	.013	-83.77	155	**.778**	.009	**-17.17**	160	.586	.033	-38.19	52
	320	160 Features	.251	.012	-195.68	276	**.778**	.008	**-44.58**	318	.522	.025	-87.66	99
Refactoring (↓)	10	Several	.287	.087	26.4	7	.491	.0	26.4	8	**.493**	.015	26.4	7
	20	Pullup	.377	.033	42.1	14	**.458**	.0	41.1	13	0.454	.006	41.1	13

incremental approach requires continuous interchange with the transformation engine for the respective next transformation step. GAs, in contrast, consult the engine mostly once to generate the initial population (besides computing mutations and repairs), thus are faster in comparison.

E. Threats to Validity

We now discuss threats that can affect the validity of the presented case study.

1) Internal Validity: We demonstrated our approach by using RL techniques to tackle different problems which are inapplicable for exhaustive methods. For comparison with other algorithms, we primarily used HV to assess the quality of found solutions. As the choice of metrics is important, we adhere to recent recommendations [29] suggesting HV as a comprehensive quality measure. Reproducibilty is threatened as the Henshin Engine suggests applicable rules in a non-deterministic way. This means, the engine returns one rule of the pool of all possible rules randomly. However, due to the low variance observable in our experiments with 30 repetitions, we deem this a negligible factor for our results.

2) External Validity: The RL algorithms are integrated in the MOMoT framework which uses Henshin as MT language to express in-place MTs. The agents in RL obtain and store a new state through exchange with the environment that receives applicable rules from the engine as response to a transformable model. Therefore, both environment and agents would need to be adapted to comply with the model representation in order to be used with other MT languages. Moreover, the solution representation of MOMoT builds on the *Solution* class of the MOEA framework[3] to save transformation variables and objectives. Although we used several cases of varying complexity levels, further experiments are required to identify strengths and weaknesses of the approach before adding RL to MDE toolboxes. Furthermore, optimizations considering multiple objectives to an equal degree are not in the scope of this work. In that regard, we point to the following future research lines.

[3]http://moeaframework.org

V. WHAT IS NEXT?

Based on our initial results, we outline several research lines (L1-4) for the model transformation community.

L1: Hybrid Search & Learning Approaches. Hybrid algorithms aim to combine the best of several worlds. We have already done a first step in this direction as we allow to apply a local search inside RL. However, further combinations may be tested in the future such as applying more sophisticated searchers for Q-learners before updating the Q-table.

L2: Multi-Objective Optimization. Currently, we use single-objective formalization for RL, also for problems which may be better formulated as a multi-objective optimization problem. Multi-objective RL may be applied for MTs as has been done with search-based approaches providing dedicated support for multiple objectives.

L3: Specific vs. generic RL. In our current solution, we provide a generic encoding for value-based RL. How to abstract and encode models for policy-based RL and further neural network based architectures is an important line to be explored in the future. In addition, RL for general in-place MTs should be compared with RL for specific problems.

L4: Further Empirical Studies. We studied an initial set of cases in our evaluation. However, further studies have to investigate how much computation power and memory are needed to apply RL for in-place MTs. Additional studies on comparing learning and search approaches or different learning approaches, architectures, and encodings are needed. Comparisons between model-based solutions as presented in this paper vs. non-model based solutions (such as pixel-based solutions) may lead to interesting scalability, comparison, and reproduction studies. Finally, reusing of learned policies for different models is an important topic for future studies.

ACKNOWLEDGMENT

Work partially funded by the Austrian Science Fund (P 30525-N31) and by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG).

References

[1] R. Bill, M. Fleck, J. Troya, T. Mayerhofer, and M. Wimmer, "A local and global tour on MOMoT," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1017–1046, 2019.

[2] A. Burdusel, S. Zschaler, and D. Strüber, "MDEoptimiser: a search based model engineering tool," in *Companion Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 2018, pp. 12–16.

[3] A. Burdusel and S. Zschaler, "Towards Scalable Search-Based Model Engineering with MDE Optimiser Scale," in *Companion Proceedings of the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 189–195.

[4] A. Burdusel, S. Zschaler, and S. John, "Automatic Generation of Atomic Consistency Preserving Search Operators for Search-Based Model Engineering," in *Proceedings of the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 106–116.

[5] S. John, A. Burdusel, R. Bill, D. Strüber, G. Taentzer, S. Zschaler, and M. Wimmer, "Searching for Optimal Models: Comparing Two Encoding Approaches," *J. Object Technol.*, vol. 18, no. 3, pp. 6:1–22, 2019.

[6] H. Abdeen, D. Varró, H. A. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth, "Multi-objective optimization in rule-based design space exploration," in *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 2014, pp. 289–300.

[7] L. Burgueño, J. Cabot, and S. Gérard, "An LSTM-Based Neural Network Architecture for Model Transformations," in *Proceedings of the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 294–299.

[8] L. Iovino, A. Barriga, A. Rutle, and R. Heldal, "Model Repair with Quality-Based Reinforcement Learning," *J. Object Technol.*, vol. 19, no. 2, pp. 17:1–21, 2020.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[10] RL4MT, "Reinforcement Learning for In-Place Transformations," https://github.com/RL4MT/RL4MT, 2021.

[11] D. C. Schmidt, "Model-driven engineering," *Computer*, vol. 39, no. 2, p. 25, 2006.

[12] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, Second Edition*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.

[13] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, no. 3, pp. 621–646, 2006.

[14] R. Heckel, "Graph Transformation in a Nutshell," *Electron. Notes Theor. Comput. Sci.*, vol. 148, no. 1, pp. 187–198, 2006.

[15] E. Syriani, R. Bill, and M. Wimmer, "Domain-specific model distance measures," *J. Object Technol.*, vol. 18, no. 3, pp. 3:1–19, 2019.

[16] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations," in *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Springer, 2010, pp. 121–135.

[17] A. Barriga, L. Mandow, J. Pérez-de-la-Cruz, A. Rutle, R. Heldal, and L. Iovino, "A comparative study of reinforcement learning techniques to repair models," in *Companion Proceedings of the ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 2020, pp. 47:1–47:9.

[18] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing, Second Edition*, ser. Natural Computing Series. Springer, 2015.

[19] K. Li, T. Zhang, and R. Wang, "Deep Reinforcement Learning for Multi-objective Optimization," *CoRR*, vol. abs/1906.02386, 2019.

[20] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019, pp. 14 610–14 621.

[21] M. van Otterlo and M. A. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement Learning*, ser. Adaptation, Learning, and Optimization. Springer, 2012, vol. 12, pp. 3–42.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.

[24] Eclipse Deeplearning4j Development Team, "DL4J: Deep Learning for Java," https://github.com/eclipse/deeplearning4j, 2016.

[25] ——, "ND4J: Fast, Scientific and Numerical Computing for the JVM," https://github.com/eclipse/deeplearning4j, 2016.

[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec 2008.

[27] K. Lano and S. K. Rahimi, "Case study: Class diagram restructuring," in *Proceedings of the Sixth Transformation Tool Contest (TTC)*, ser. EPTCS, vol. 135, 2013, pp. 8–15.

[28] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[29] T. Chen, M. Li, and X. Yao, "How to Evaluate Solutions in Pareto-based Search-Based Software Engineering? A Critical Review and Methodological Guidance," *CoRR*, vol. abs/2002.09040, 2020.

[30] X. Zhang, Y. Tian, and Y. Jin, "A Knee Point-Driven Evolutionary Algorithm for Many-Objective Optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 6, pp. 761–776, 2015.