Search-Based Model Optimization using Model Transformations

Joachim Denil, Māris Jukšs
MSDL-Lab, School of Computer Science
McGill University, Canada
{denil,mjukss}@cs.mcgill.ca

Clark Verbrugge
Sable, School of Computer Science
McGill University, Canada
clump@cs.mcgill.ca

Hans Vangheluwe
Ansymo and MSDL
University of Antwerp, Belgium
McGill University, Canada
hv@cs.mcgill.ca

ABSTRACT

With the advent of new and more complex software engineering problems and applications, synergies between Search-Based Software Engineering (SBSE) and Model-Driven Engineering (MDE) have been proposed. SBSE formulates the software engineering problem as a search-based optimization (SBO) problem. In Model-Driven Engineering, model transformation is the preferred technique to manipulate models. The challenge thus lies in adapting model transformations to tackle SBO tasks. In this paper we explore the inclusion of search directly in model transformations, without the need for an intermediate representation. We also investigate the feasibility and scalability of the approach on an industrial-scale problem of resource allocation. We demonstrate that our solution is feasible and applicable to problems where representing the problem in a search amenable representation is time consuming, hard or even impossible.

Keywords

SBSE, MDE, Model Transformation, CMSBSE, Search Methods

1. INTRODUCTION

Model-Driven Engineering (MDE) [1] uses abstraction to bridge the cognitive gap between the problem space and the solution space in complex software and system engineering problems. To bridge this gap, MDE uses models to describe complex systems at multiple levels of abstraction. A model is said to conform to a meta-model [2]. A meta-model defines a possible, infinite set of syntactically conforming instance models by defining a set of constructs, relations and a set of constraints. The MDE approach supports systematic transformations of problem-level abstractions into their implementations. Model transformation is even regarded as the "heart and soul of model-driven software and system development [3]".

One technique for solving complex software engineering problems is Search-Based Software Engineering (SBSE). SBSE is a discipline where computational search techniques are used to solve complex problems in the software engineering field. SBSE formulates the software engineering problem as a search-based optimization (SBO) problem. A wide range of optimization techniques are used to search the solution space for a local or global optimum to the problem. SBSE has been applied to a multitude of software engineering processes and products such as test case generation, requirements engineering, resource allocation, etc. [4]. Applying Search-Based Software Engineering (SBSE) to a software engineering problem requires four components: (a) a representation of the problem, (b) a method to create a candidate solution to the problem (c) a goal-function metric to evaluate if a candidate solution is "good", and (d) an optimization method. The theory of SBSE currently offers little guidance as to the choice of representation, fitness metric, and search method, therefore such choices are often made empirically on a problem-by-problem basis [5].

With the advent of new and more complex problems and applications, synergies between the SBSE and MDE are considered [6]. Burton and Poulding [5] propose models as a suitable problem and solution representation. Models indeed enable the representation of the problem in a highly structured and consistent way. This eliminates the need of finding a suitable problem-specific representation amenable for search. Model-Driven Engineering also has a tool-set available for manipulating these models in a structured way by the use of model transformation. Finally, MDE also allows to visualize the obtained solutions without an additional translation cost from the problem-specific search representation to a representation in the problem domain.

Although model transformation is proposed as the tool for the manipulation of models, little work has been done in integrating search in model transformation models. In this paper we show how to integrate a set of well-known single-state search techniques in model transformation models. We identify the different components needed in a transformation language to allow this. Furthermore, we will look at the performance of the different search techniques using model transformation as the technique to generate candidate solutions to the problem.

Because a multitude of model transformation techniques exist in the literature, we focus on the use of rule-based model transformations where models are represented as typed, attributed directed graphs.

The rest of this paper is organized as follows: In the following Section, we look at other motivations to include search techniques in model transformation models. Section 3 introduces the running example used to show the contributions in this paper. Section 4 introduces the components of a rule-based model transformation language. In Section 5 model transformation models with search are created. Our experimental evaluation of the approach is studied in Section 6. In Section 7 we discuss the approach and look at other advantages of having a MDE approach to search prob-

lems as well as the disadvantages. Section 8 discusses related work with respect to the use of model transformation for search. Finally, in Section 9 we conclude and look at future work.

2. MOTIVATION

Including Search-Based Optimization techniques in model transformation models has some other advantages over creating a search-amenable representation of the same problem.

Transformations used to create candidate solutions for the search method make domain knowledge explicit. Indeed, they show where the variation points in the model are and how we can create candidate solutions to the problem. In the proposed approach, the model remains at the centre of the problem. Complex problems for searching are described in the natural language of the engineers since both the model and the transformation rules share a common visual representation. This removes the difficult need to create a problem-specific search representation of the problem. No transformations need to be created to transform the model to this search representation and vice versa.

There is however another advantage to the use of model transformation rules to explicitly model the variation points. Domain knowledge already known by the domain experts can be easily integrated in the search problem by either adding another rule or augmenting the existing rules with extra constraints.

Other domain knowledge can be discovered by mining the traces of the transformations. The mining of the trances can uncover the sensitivity of parameters, where the changes of certain parameters have more effect than the effects of other parameters. These are the choices that should be focussed on during search. The mining of the traces can also be used to uncover new domain knowledge when certain choices always lead to good or bad solutions.

Finally, using a transformation-based approach to search problems allows for the full integration of the optimization in the MDE-cycle. The FTG+PM proposed in [7] allows for the creation of complex model transformation chains with non-linear control- and data-flow. The optimization can be completely shown as an FTG+PM, as shown in [8]. The FTG+PM as well allows for the creation of optimization chains, where the search problem is divided into different parts to create complex, hybrid optimization chains [9]. Manual optimization steps are also possible in this approach, where a selection of steps can be done using human interaction. The overall approach allows for the full integration of search in the MDE cycle resulting in documented, reusable optimization models.

3. RUNNING EXAMPLE

The contribution of this paper is demonstrated using a running example. The example is a resource allocation problem in the automotive domain based on [10]. In the example, a set of communicating software functions needs to be assigned to a set of electronic control units (ECUs) connected by a communication bus. Afterwards, these functions need to be packed into tasks with an assigned priority and a set of messages (with a priority) on the bus. The goal is to minimize the end-to-end latency of the application.

Zheng et al. approach the problem by searching for a mapping where the total loads of the different ECUs are below a threshold of 69% (the schedulability test for rate-monotonic systems [11]). The goal-function or fitness-function for finding an optimal solution is to minimize communication between the different ECUs since the communication on the bus introduces delays that impact the timing behaviour of the final solution. Afterwards, a similar search problem is defined for finding an optimal mapping of software functions to tasks with a priority and messages on the bus. For reasons of clarity we only focus on the first part of the example, the mapping of software functions to ECUs. Feasible solutions for this problem have all software functions mapped to an ECU, while the total load on the ECU is smaller than 69%. The fitness-function minimizes the communication on the bus. The number of (feasible and non-feasible) solutions to this problem is defined by E^S, where E is the number of ECUs and S the number of functions.

The meta-model used in the motivating example is shown in Figure 1. It contains a SWC (software function) with a *period* (in ms) and a worst-case execution time ($WCET$; in ms) attribute. An SWC can be mapped to a single ECU. When a SWC is mapped to an ECU the $Load$ attribute of the ECU is increased by ($WCET/Period$) of the mapped software function. A similar calculation is used for the $Load$ on the Bus, based on the $Size$ of the communication message ($Comm$) and the $Period$ of the sending software function. The $Load$ of the Bus only increases when the sending and receiving SWC are mapped to a different ECU.

Figure 2 shows an example of a model in this meta-model. An industrial size model of this mapping problem consists of 40 software functions, communicating over 80 messages between them. These functions can be mapped to eight ECUs connected to a single bus. The design space of the industrial size model consists of 8^{40} different solutions.

We chose this example for different reasons. The problem is a resource allocation problem that is well known across different software engineering fields [12, 13, 14]. Because the problem is well studied, a more optimal search representation exists for this problem in the literature based on lists. This allows us to compare our optimization results as well as the performance with a known implementation. Finally, the search space is very large, so a lot of different choices can be made by the optimization algorithms.

4. TRANSFORMATION LANGUAGES AND T-CORE

The developed search augmented transformation models are based on the T-core transformation framework. T-Core is a minimal collection of model transformation primitives, defined at the optimal level of granularity, presented in [15]. T-Core is not restricted to any form of specification of transformation units, be it rule-based, constraint-based, or function-based. It can also represent bidirectional and functional transformations as well as queries. T-Core modularly encapsulates the combination of these primitives through composition, re-use, and a common interface. It is an executable module that is easily integrable with a programming or modelling language. Figure 3 shows some of the components of a transformation language. We briefly discuss the components we use in creating the different search transformations. More information can be found in [15].

Rule-based model transformation languages work on typed, attributed and directed graphs that represent the model.

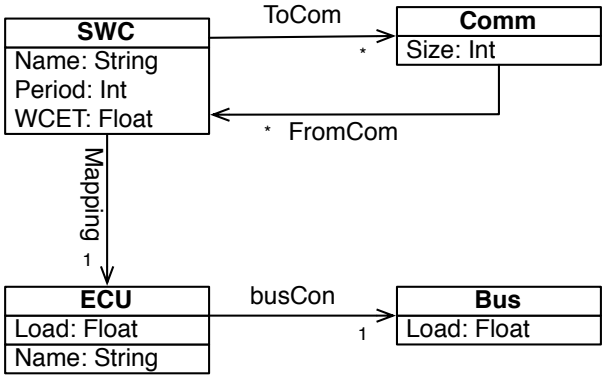

Figure 1: Metamodel used in the motivating example

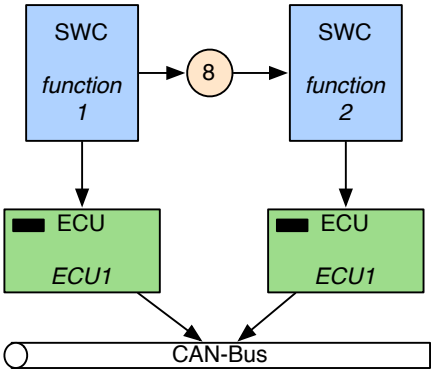

Figure 2: Example model showing the visual representation used in the motivating example

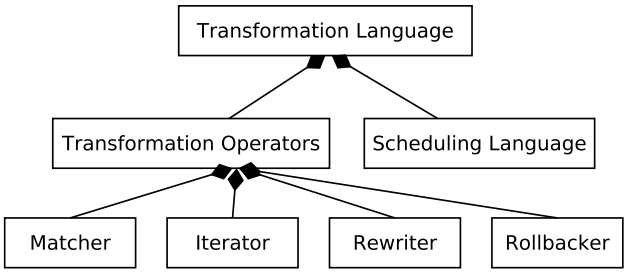

Figure 3: Composition of a Transformation Language

A transformation rule represents a manipulation operations on the represented model. A rule consists of a left-hand side (LHS) pattern representing the pre-condition for the applicability of the rule. The right-hand side (RHS) pattern defines the outcome of the operation. A set of negative application condition (NAC) patterns can be defined to block the application of the rule. Pattern elements in the LHS, RHS and NAC are uniquely labeled as in figure 5 to refer to matched instances. The transformation rule outcome is decided based on these unique labels.

- Matcher: The matcher binds model elements that satisfy the pre-condition pattern. The different "matches" are stored in a set. The matcher can be parametrised to find a certain number of matches or all of the available matches in the model. Using graph-based models, the matching problem leads to the subgraph isomorphism problem that is known to be NP-complete [16]. Different algorithms can be used to find the bindings:
  - VF2: VF2 solves the problem as a constraint-satisfaction problem. The algorithm constructs a search tree traversing the host graph depth-first and backtracks when the current search-state fails a compatibility test [17]. The default matcher in T-Core uses an augmented version of the VF2 algorithm [15].
  - Search Plans: Search Plans solve the problem by performing a local search. The algorithm focuses on ordering elementary operations by using tool-specific heuristics to guide the pattern matching process. The order of the operations are defined in advance. Examples of elementary operations are checking the existence of an edge and decisions of navigating the pre-condition pattern. The heuristics used in our T-Core search plan are based on [18].
  - Rete: Rete is an incremental pattern matching algorithm first developed for production systems [19]. It uses memoisation to store partial matches in a network. The network consists out of join nodes to join a partial match with a connected edge or node of the model. Other constructs can filter different matches based on a constraint. The Rete network used in this paper is constructed manually and is based on techniques shown in [20].
- Iterator: The iterator gives explicit control to select a match from the matchtset to rewrite in the model. The iterator can be setup to always select the first match in the set or to randomly select a match in the set.
- Rewriter: The rewriter rewrites the model using the RHS pattern.
- Rollbacker: The rollbacker enables backtracking in the transformation language. The backtracking mechanism for the VF2-like matcher and search plans matcher is based on copying the graph structure and the match-sets (including a selected match). For the Rete matcher a transaction based system is used where the operations are stored and rolled back.

The scheduling language is used to compose different rules and transformation primitives after each other. To execute a single transformation rule, a matcher first creates the matchset containing the matches that comply to the LHS pattern of the rule. One of these matches is chosen by the *iterator*. The *rewriter* adapts the model based on the chosen match and the RHS pattern. Different kind of scheduling languages can be used, like activity diagrams, DEVS or a common sequence, branch, loop language [15].

An in-depth study of the different features of a model transformation languages can be found in [21].

5. INCLUDING SEARCH IN TRANSFORMATION MODELS

To include search in model transformation models, the different components of a search-based optimization techniques need to be present in the model transformation. The components of a search-technique consist of:

- Problem representation: The representation is the graph-based model without any augmentations for search.
- Creation of candidate solution: Model transformation rules are used for creating a (or a set of) candidate solutions.
- Optimization technique: The optimization technique is implemented using the scheduling language of the model transformation language.
- A metric to evaluate if a solution is "feasible" and "good": Metrics can be calculated using (a) a model transformation, when the metric is based on structural properties of the model or, (b) transforming the model to another representation (e.g. a simulation model, algebraic equation, etc.) if the metric is based on the behaviour of the model. In the running example, the metrics are calculated using a model transformation since the metric is based on structural properties of the model.

We explicitly make the distinction between a "feasible" and a "good" candidate solution. A feasible candidate solution is a model that is within all the constraints of the search problem. In our running example this means that the load of each processor is below 69% and all software components have been mapped. A "good" solution or "better" solution is a comparison of two feasible solutions with respect to the goal-function. In the running example this means that the bus load of the one model is less than that of another model. Depending on the problem, the model transformation rules can make only feasible solutions or, because of the complexity of the problem, feasible and non-feasible solutions . In the latter case, these non-feasible candidate solutions should be pruned on evaluation.

In the following subsections we show that it is feasible to include different search techniques in model transformation models. Four well known search techniques that are used in optimization are constructed. The first two: *exhaustive and randomized* search create a number of solution points in the search space. The latter two: *Hill Climbing and Simulated Annealing* start optimizing a single solution. We define for each of the proposed search techniques what the requirements of the model transformation language are.

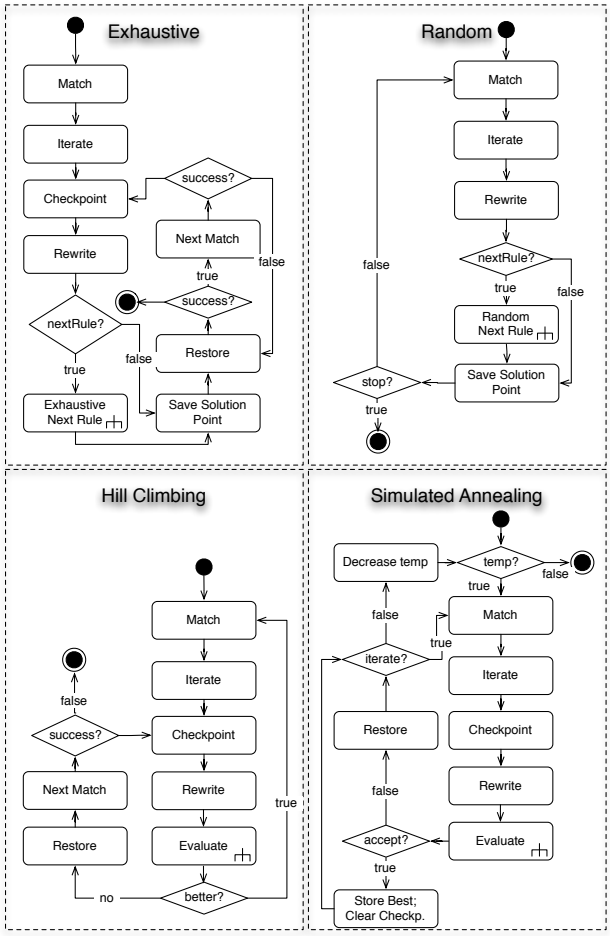

Figure 4: Activity diagram of Exhaustive Search, Random Optimization, Hill climbing and Simulated Annealing using T-Core primitives.

Transformation rules are created for the running example to create the candidate solutions.

5.1 Exhaustive Search

While the exhaustive search is infeasible for most problems, because a potentially huge search space needs to be explored, it can be used for the optimization of small problems. Exhaustive search will generate all solutions of the design space. Figure 4 shows an activity diagram of the implementation of the exhaustive search method.

The transformation starts by matching all the occurrences in the start model. The iterator chooses the first match in the matchset, followed by a creation of a checkpoint. This checkpoint contains (a) the model, (b)the selected match and (c) the matchset, without the chosen match. The selected match is rewritten in the model. When more rules are available or the same rule has to be executed multiple times, the sequence (match, select match, checkpoint and rewrite) is repeated. Once a single solution is created, the backtracking can start. Since the backtracker can contain multiple checkpoints (from multiple rule applications), the last one is selected. This restores (a) the model, (b) the match that

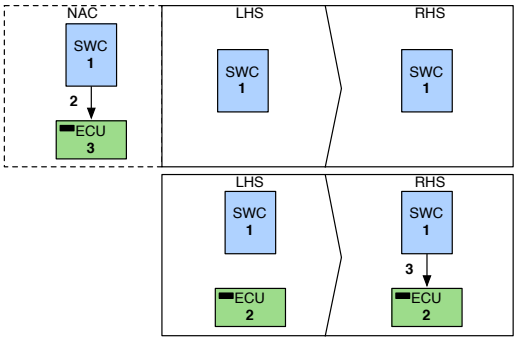

Figure 5: Amalgamated rule for the creation of mappings in the exhaustive and random search transformation model

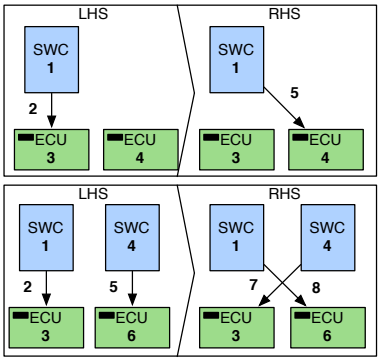

Figure 6: Rules for creating a neighbouring solution in hill climbing and simulated annealing

was selected at the time instant the checkpoint was made and (c) the rest of the matchset. The iterator is used to replace the previously chosen match with another one from the matchset, again this is check-pointed and rewritten. The process continues as described above until all matches in the matchsets of all checkpoints have been applied. It results in a DFS-like traversal of the design space.

The transformation rule for the exhaustive search for the motivating example uses an amalgamated rule [22], shown in Figure 5, to prevent the creation of duplicate results. The first rule, selects a single unmapped software function and marks it for the nested rule. The nested rule maps the software function to an ECU. The rule prevents the creation of infeasible solutions, by only selecting an ECU with enough temporal slack.

5.2 Randomized Search

In randomized search a set of solutions are created in a random way. The technique is used to get an overview of the search-space. It can also be used to create a starting point for other search techniques that require a candidate solution to start optimizing.

Random search uses only the matcher, iterator and rewriter. After matching all occurrences of the pattern in the model, a random match is selected for rewrite. This requires a different iterator than in exhaustive case. The rewriter applies the randomly chosen match on the model. Afterwards, another rule or the same rule can be executed until a solution point is obtained. A loop is used to create multiple solution points.

The same transformation rule of Figure 5 is used for creating a set of random solutions in the motivating example.

5.3 Hill Climbing

Hill climbing is a local search technique that uses an incremental method to optimize a single solution. The algorithm starts with an arbitrary solution to the problem and iteratively optimizes this solution by changing a single element. If the change is a better solution to the optimization problem, the change is accepted. This procedure is repeated until no better result is found.

Figure 4 shows the building blocks of the hill climbing transformation. After matching all occurrences in a (set of) rule(s), the iterator picks one match at random and rewrites this in the model. The solution is evaluated and compared with the original solution. In case the solution is not better, the original solution (with the matches) is restored and another match is randomly selected and evaluated. If the solution is a better one, it is accepted.

The evaluator contains a set of transformation rules to calculate the metrics of the solution or to generate an analysis or simulation model that can be executed. The metrics obtained are used by the scheduling language to decide if the solution is more optimal than the previous solution. When a better solution has been found, the process is restarted until no more improvements can be found.

Figure 3 shows the rules involved in the hill climbing process for optimizing the motivating example. The first rule moves a software function to another ECU. The second rule switches the mapping of two software components. Both rules only create feasible solutions by checking the temporal slack on the ECUs.

5.4 Simulated Annealing

Simulated annealing is a generic probabilistic optimisation algorithm [23]. The algorithm is inspired from metallurgy where controlled cooling is used to reduce defects in the crystal structure of the metal. The controlled cooling is used in simulated annealing to decrease the probability in accepting not only a more optimal solutions but also a less optimal solution. By not only accepting better solutions, the search algorithm is able to escape a local optimal result.

Again all occurrences are matched where only a single one is picked for rewriting. Based on the difference between the previous solution and the candidate solution, and the temperature, the candidate solution is accepted or rejected (resulting in a backtracking step). At low temperatures only better and equal solutions are accepted. Backtracking is thus more intensive at lower temperatures. This process is iterated for a pre-defined number of times. Afterwards, the temperature is decreased and the optimization algorithm resumes with a new temperature. The best overall solution is stored during the optimization cycle.

The transformation rules involved in generating a candidate solution in the running example are equal to the ones used in the hill-climbing algorithm.

6. EXPERIMENTAL EVALUATION

The experimental section is divided into five parts. In

the first part, we explain our experimental setup. The second part looks at the actual results obtained from optimizing the running example. We compare the three matching techniques together with a coded implementation of the same problem. The third part compares the computational expense of the three matching techniques. In the fourth part we compare the computational expense of a rule-based model transformation technique with that of a coded implementation. Finally, we define the treats to the validity of the experiments.

6.1 Experimental Setup

We compare for each of the proposed search techniques (with the exception of exhaustive search) the results of the optimization and the run-time performance. Different models are used in the study. We create a start point for each of the techniques by varying the number of ECUs and software functions in the industrial model by a step size of six software functions. The number of ECUs is automatically increased based on the total load of all ECUs if the functions would have been mapped. The total amount of used models in the experiments is six, starting from 10 software functions mappable to 3 ECUs to 40 software functions mappable to 8 ECUs (the industrial size model).

Because a random starting point is used for both hill climbing and simulated annealing and due to the high randomness of all algorithms, the results of the optimizations are not equal. Therefore multiple execution runs are done per starting model. For the randomized search, 100 solutions are created. For hill climbing we limit the study to nine execution runs, and because of the run-time cost of creating a single simulated annealing result, only three runs are executed.

The parameters of the simulated annealing are slightly adapted from the parameters used in [24]. The starting temperature is 100 with a temperature decrease of 0.89%, resulting in 40 temperature drops. A temperature decrease is done after the algorithm has explored 45 times the number of software functions in the model. The algorithm stops after finding the optimal solution (no communication) or a temperature below one. For the industrial size model, simulated annealing explores 72000 individual solutions to the problem.

The transformation models and schedules have been instrumented to record for each transformation step the metric used in this study (the load of the bus), as well as the time needed in all steps of the transformation process (matching, rewriting, storing and backtracking). For example, in a single simulated annealing optimization of the industrial model, 40 intermediate bus load metrics and 72000 performance results are obtained.

The coded implementation of the running example uses a more optimal list implementation instead of a graph. It is based on the implementation described in [24].

All experiments are run on a cluster consisting of 32 individual but equal hardware nodes. Each node has an Intel® Core 2 CPU X6800 running at 2.93GHz with 8 GB of memory.

6.2 Optimization Results

For each of the techniques and models we show the results using a bar graph. The X-axis shows the different models used in the search process, the label is based on the number of software functions x the number of available ECUs. The Y-axis shows the total load of the communication bus in the model. The goal function is to minimize this communication. We compare the results of the three common matching techniques in rule-based model transformation with each other. To make the result study complete, we also compare it with the results of the coded implementation. The height of the bar corresponds with the average of the load on the bus. A standard deviation is shown at the top of each bar. In all the graphs, the cyan bar shows the result of the coded implementation (CI), the green bar, shows the results when using the Search Plan matcher (SP), VF2 is shown with a red bar and Rete is shown in blue. The results between the different matching techniques and the coded implementation with a model of equal size should be similar.

6.2.1 Exhaustive Search

The exhaustive search was only executed for a very small model (namely two software functions mappable to two ECUs) with VF2 and Search Plans. Both techniques reported the same amount of solutions, with the same resulted loads on the bus. The results have been checked manually. No further experimentation is done due to the high computational and memory cost of finding all solutions of the mapping problem. The technique however, works as expected.

6.2.2 Randomized Search

Figure 7 shows the average bus load as well as the standard deviation of the randomized search on the set of 100 start models. The figure shows a very high similarity between the different matching techniques and the coded implementation.

For the industrial size model, the average load on the bus is 10.2 byte/ms. This result can be used as a base-line for the optimization of the models used in simulated annealing and hill climbing.

6.2.3 Hill Climbing

Figure 8 shows the optimization of nine random models. As expected, the results between the coded version and the model transformation based optimization is very similar. Compared to the randomized search (the start point for the hill climbing), the hill climbing finds a much better result. The average load on the bus is down to 4 byte/ms.

In Figure 9 we plot the execution run of three industrial size models, each with another matching technique. The X-axis shows the number of examined candidate solutions. The bus load is shown on the Y-axis. Since the number of steps required to reach a final solution is not equal between different models, some models reach their final solution faster than others. Hill climbing improves the solution very fast. In the late phases of the algorithm, it is hard to find better solutions and more plateaus are introduced in the graph.

6.2.4 Simulated Annealing

Figure 10 shows the average of optimizing three randomized models using simulated annealing. The average load on the bus with the industrial size model is 3.25 byte/ms, which is a better result than the solutions found by hill climbing.

As with hill climbing, we depict the execution of three industrial size models, each with another matching technique in Figure 11. The intermediate results between the different temperature drops is plotted. Compared to hill climbing, the optimization goes much slower. A single temperature

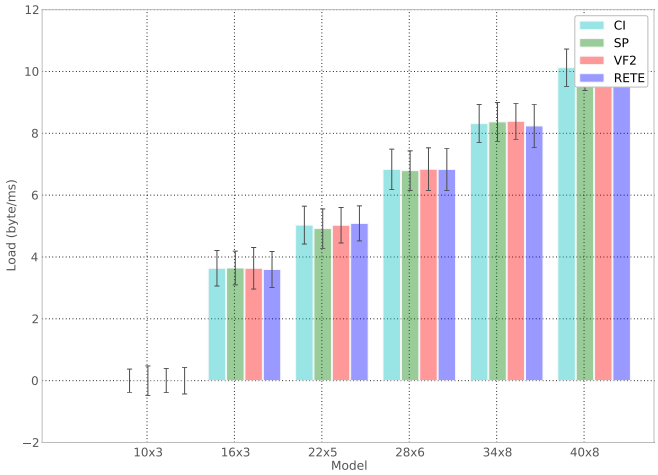

Figure 7: Randomized Search: Comparison of average load times between different techniques

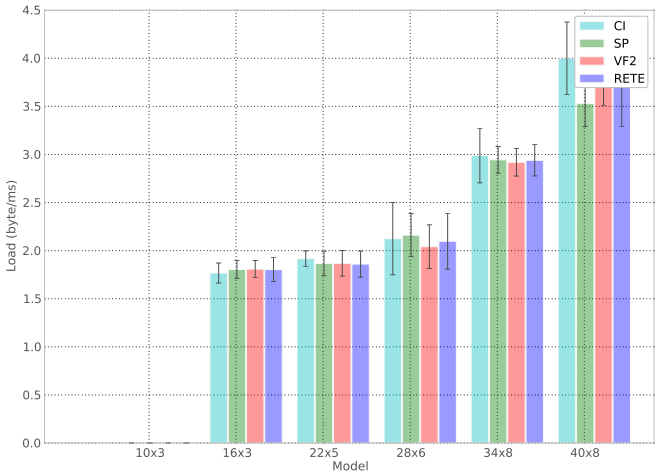

Figure 8: Hill Climbing: Average load times between different techniques

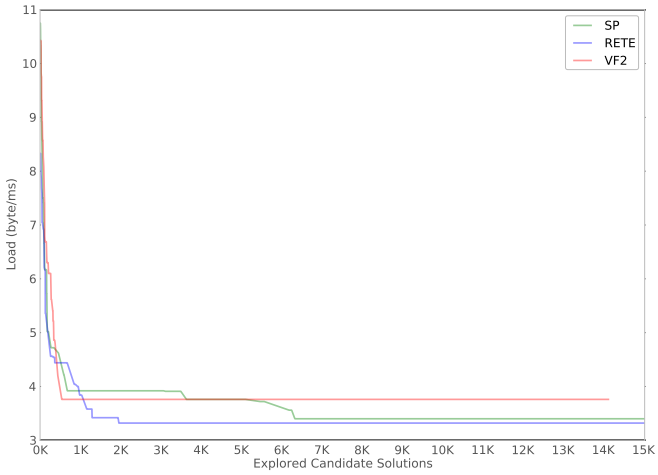

Figure 9: Hill Climbing: Example run of three different models (each using another matching technique)

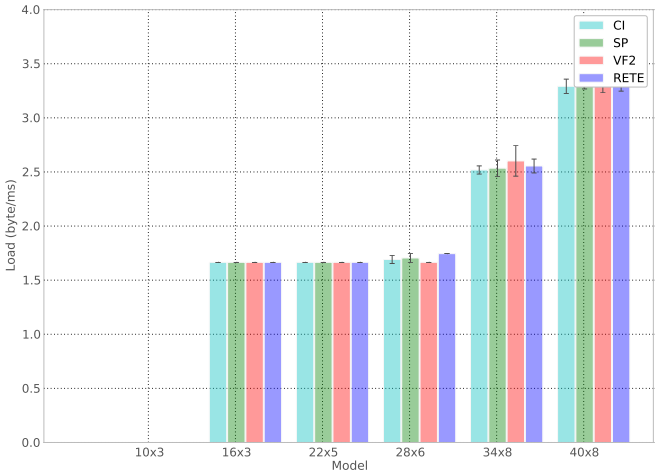

Figure 10: Simulated Annealing: Average load times between different techniques

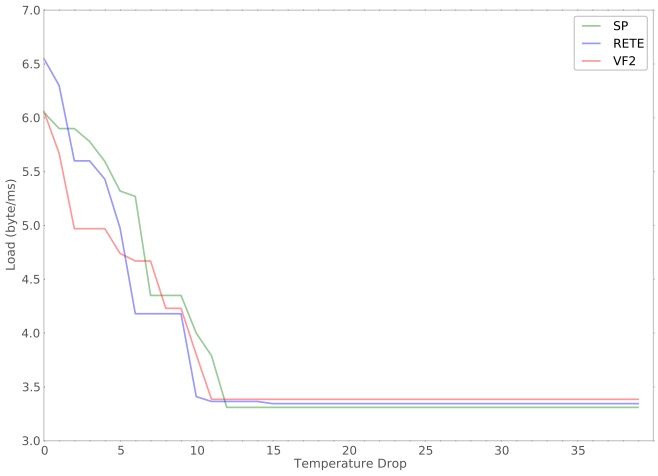

Figure 11: Simulated Annealing: Example of the result during cooldown

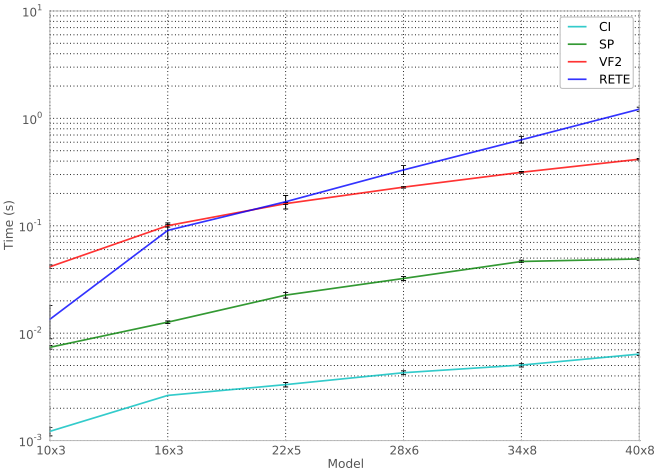

Figure 12: Randomized Search: Comparison of average computation times between different techniques

drop in this graph requires the algorithm to look at 1800 different candidate solutions.

6.3 Performance Analysis

In the following paragraphs we compare the performance and scalability of the three rule-based matching techniques on randomized search, hill climbing and simulated annealing. For all graphs, we only compare the time needed to create a single candidate solution. The time spent in calculating the metric is not included in the graphs. Because the Rete network by nature contains all of the rules in a single network, some time is spent in calculating part of the metric.

6.3.1 Randomized Search

Figure 12 shows the total cost of creating a single randomized solution. Note that the scale of the Y-axis (in seconds) uses a logarithmic scale. The search plan matcher however is around nine times faster for creating a single solution than the other two techniques.

The approach scales exponential based on the number of components in the model. This is normal since the complexity of finding graph isomorphisms in a graph is at least exponential to the number of edges and vertices in the model. Note that the X-axis is not fully linear to the number of components in the model.

6.3.2 Hill Climbing

A different representation is used for showing the performance results of the hill climbing algorithm. Figure 13 shows a bar graph where the top of the bar shows the time needed to create a single solution for the hill climbing algorithm. Each bar shows the time needed for each of the components in the algorithm: Green is the time needed for **M**atching, Red is used for **R**ewriting. Backtracking operation consists of the **S**et operation to create a checkpoint shown in blue and the **G**et operation to restore a checkpoint shown in magenta. For the rewriter (Rstd) and Matcher (Mstd) we also show the standard deviation. The first bar of each model, shows the run-time performance of the Search Plan technique (P), the second bar shows the results for VF2 (V) and the third bar depicts the results obtained form using a Rete network (E).

The results are calculated based on the total time spent during matching, rewriting, getting and setting the checkpoint divided by the number of created candidate solutions. We compare the techniques on a single candidate solution and not on the entire solution cost because the total number of created solutions is not fixed

For both Search Plans and VF2 matching is a very expensive operation while using a Rete net, the match operation is instantaneous. Rewriting on the other hand is a very expensive operation using Rete because new facts about the model are added, removed or updated in the net.

Hill climbing relies heavily on the backtracking operation. Restoring a checkpoint is almost instantaneous in both VF2 and Search Plans. Rete however uses a transaction based operation. This results in an opposite rewrite of the previously rewritten match. Rete with a transactional-based system for backtracking is thus not a good choice with hill climbing.

In total, the Search Plan implementation is the most computationally appropriate technique to use while Rete is the worst.

6.3.3 Simulated Annealing

Finally we show the results of the run-time performance of the simulated annealing algorithm. Figure 14 shows a similar graph as shown with hill climbing. In this case, VF2 is clearly the worst because of high matching times. The difference between Search Plans and Rete is less prominent. However Rete's high backtracking and rewriting times are evident. Figure 15 shows the run-time performance progress as the temperature drops. Note, at the beginning of the algorithm, less backtracking is required for simulated annealing. When time progresses and temperature drops, more backtracking is performed.

Search Plans therefore is the most appropriate technique to use for simulated annealing.

6.4 Comparison with a coded implementation

Figures 12, 13 and 14 also show the cost of creating a single solution in the coded implementation of the running example (C). The coded version is much faster for randomized search as well as for simulated annealing. The difference is significant because the coded implementation creates only a single candidate solution compared to the matching of all matches in the model transformation approach. For hill climbing, the situation is different. Search plans has an equal computation cost of finding all the neighbours in the graph compared to finding all the neighbours in the coded implementation.

6.5 Threats to Validity

The first threat to the internal validity of the study is the comparison of run-time performance data from VF2, Search Plans and Rete. We excluded the run-time cost of calculating the bus load in all algorithms. As a consequence of the architecture of Rete, only a single net is used for the transformations in creating a single solution and evaluating it. The rewriting cost of Rete thus includes some of the run-time overhead of calculating the bus load. We analyzed the results of the study with the data included as well, though it has no effect on the global results shown in the graphs.

The starting point of the optimization runs of hill climbing and simulated annealing is randomly chosen using the randomized search algorithm. To cope with this and the high randomness of the different techniques, multiple execution runs are performed for each of the models. We use the creation of a single candidate solution as the metric for the run-time cost of our approach. This is done because the number of candidate solutions needed to reach a local optimum in hill climbing differs from run to run. To be able to compare this with simulated annealing, the same comparison metric is chosen.

All models, model transformations and data is downloadable from *msdl.cs.mcgill.ca* so results can be reproduced.

The biggest threat to the external validity of the run-time performance techniques is that only a single case study is used in the evaluation of the performance. However, since the size of the models is incremented in the study some preliminary conclusions about our approach can be drawn from this. Another threat to the external validity of the run-time cost of this approach is the use of our own prototyping tool. Our Rete prototype is manually constructed though optimizations still need to be implemented. A simi-

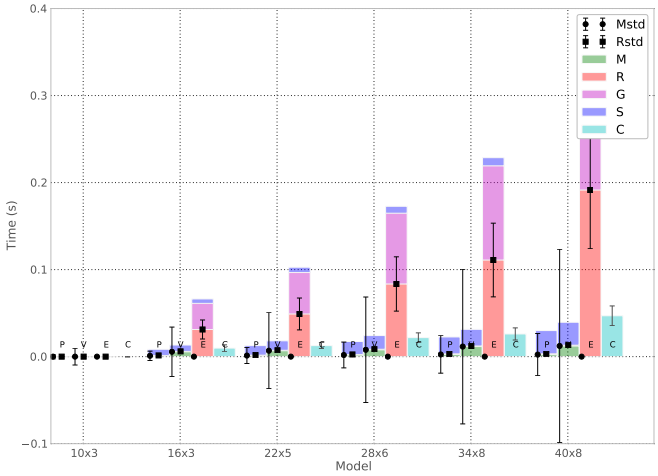

Figure 13: Hill Climbing: Average computation times for a single candidate solution

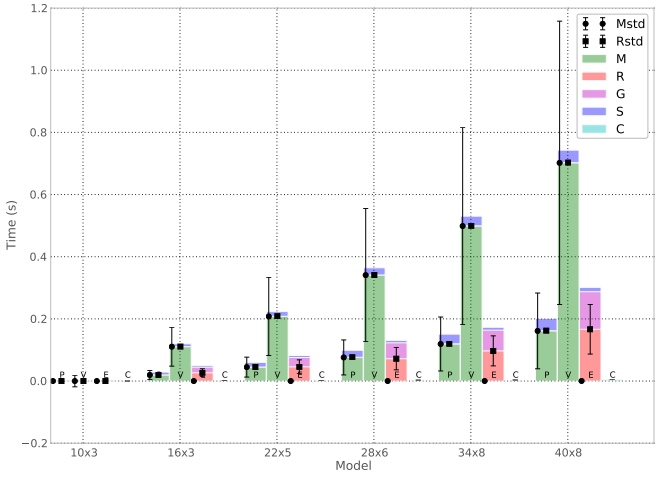

Figure 14: Simulated Annealing: Average computation times for a single candidate solution

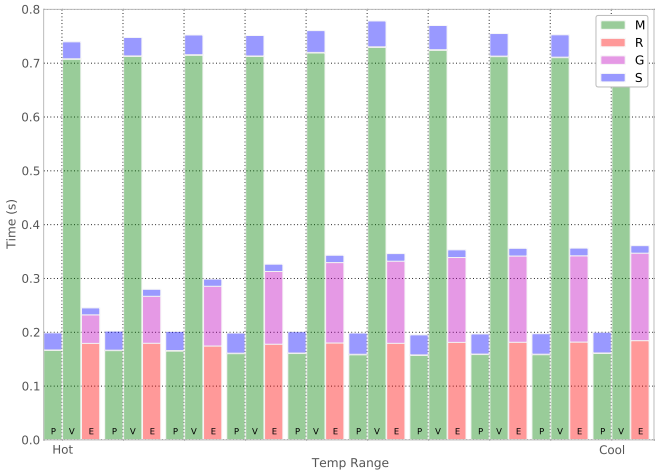

Figure 15: Simulated Annealing: Influence of cooldown on backtracking

lar threat stems from other tools used in academy and industry. The different matching techniques have different implementations and could potentially have a different run-time cost.

7. DISCUSSION

In this section we discuss some of the issues and opportunities of using a rule-based model transformation approach to search-based software engineering.

The proposed search algorithms can be used as a starting point for more advanced optimization techniques. Multiple variants of the presented algorithms are proposed in the literature. The exhaustive search, for example, can be converted to a branch-and-bound algorithm [25]. By adding an extra evaluation on partial candidate, solutions branches can be pruned very early during search to find the optimal solution when the branch is already less optimal than the currently found best solution. In hill-climbing, extra features can be added in the scheduling language to allow for random restarts, selecting the steepest descent, etc.

Adding extra domain knowledge to the transformation models is relatively easy. For example, we can augment the created rules of the example to guide the search. We know that communicating software functions should, where possible be put together on a single ECU. The transformations can encode this knowledge by adding an extra rule that is used in the randomized case. The rule can also be used as a preferred rule during hill climbing. When no matches are found, the other two rules can be used as a fall-back.

However, the main disadvantage of the approach is the run-time performance when compared to an optimal representation of the search problem using for example lists, as shown is Section 6. This is attributed to the complexity of finding of sub-graphs in the model. Using the correct type of matching technique on a model-to-model basis can help to boost the performance of the approach. For this problem, the Search Plan matcher outperforms the other techniques for all the performed experiments. However, since our Rete matcher is the least mature matching technique in our tool and the constructed net is not optimal, the technique shows promise.

Combinations of matchers can improve the computational expense of the model transformation approach. For example, backtracking in simulated annealing happens more often in the later phases of the algorithm. Since this is a very expensive operation with the Rete technique, a switch can be made to the Search Plan technique after a certain amount of cycles. Heuristics need to be developed based on sound experimentation with lots of different search problems.

Parallelism could also be used to increase the performance of this approach. In exhaustive searches, the different branches can be explored in parallel. Randomized search can create multiple solutions in parallel. Hill climbing and simulated annealing benefit from parallelism by optimizing multiple random solutions in parallel so more of the search-space is covered in a single execution of the algorithm.

Finally, all matches are always matched in the underlying model even when this is not always necessary. For example, in simulated annealing, only a single random match is required for the algorithm. The matching of all instances of the pattern is done because our implementations of both VF2 and Search Plans always find the same match in the model when only a single match is requested. It is deter-

ministic in the sense that the search for matches always start at the same point in the graph. Other implementations of the algorithms do not necessarily have this feature and are non-deterministic in selecting the start point in the model. This could work to the advantage of the randomized search and simulated annealing and would increase the performance drastically. However, when a true uniform random neighbour is required for the optimization algorithm to work, a true random matcher needs to be constructed.

Another approach to improving the performance of the matching is to use a divide and rule strategy. Scoping [26] can be used to select subparts of the model to optimize. The scoping can dynamically change (and broaden over time) to reduce the cost of matching during hill climbing where all matches are required for the algorithm to work.

Introducing SBO in model transformation is best used when the construction of an optimal search representation is hard, very time-consuming or even impossible. The rules to create a candidate solution are designed in the domain language of the experts. This allows non optimization experts to design optimization problems without the need

8. RELATED WORK

In the SBSE community, models have already been used as a representation for complex problems. But to the best of our knowledge, the use of SBO techniques in model transformations has not been explored in a structural way. Combining modelling, model transformation and search techniques has been proposed before and related work can be found in two areas: (a) CMSBSE and (b) Design-Space Exploration.

8.1 CMSBSE

In [5] authors advocate the use of models, domain-specific modelling and model transformation in SBSE. This implies however that SBO techniques should be present in the model transformations. Solutions for combining search techniques in model transformations are not discussed.

An example of the use of models and search can be found in [27]. The authors search for a model transformation to translate a sequence diagram into a colored Petri net. Simulated annealing as well as Particle Swarm Optimizations are used to search the large design-space of such a problem. The authors use this experience in [28] to create a framework for using genetic algorithms with models. A generic encoding metamodel is proposed as well as the use of model transformations for encoding and decoding the domain specific models.

8.2 Design-Space Exploration

Transformation based approaches to Design-Space Exploration is a relatively new topics in the field. Two approaches are used: In the first approach, models are transformed to another representation more suitable for exploration. For example, the DESERT tool-suite [29] provides a framework for design-space exploration. It allows an automated search for designs that meet structural requirements. Possible solutions are represented in a binary encoding that can generate all possibilities. A pruning tool is used to allow the user to select the designs that meet the requirements. These can then be reconstructed by decoding the selected design. In [30], Saxena and Karsai present an MDE framework for general design-space exploration. It comprises of an abstract design-space exploration language and constraint specification language. Model transformation is used to transform the models and constraints to an intermediate language. This intermediate language can be transformed to a representation that is used by a solver. Finally, the OCTOPUS toolchain [31] is a domain specific tool for the design-space exploration of embedded systems. The tool is organized around an intermediate language used for connecting different tools together.

A second approach uses model transformation to search the design-space using the model itself. Schätz et al. developed a declarative, rule-based transformation technique [32] to generate the constrained solutions of an embedded system. The rules are modified interactively to guide the exploration activity. In [33] a transformation-based approach is proposed to generate the full design-space of a cyber-physical system. The transformation language is based on Answer-Set Programming. Different approximation levels are introduced where non-feasible solutions can be pruned. In [34], a framework for guided design-space exploration using graph transformations is proposed. The approach uses hints, provided by analysis, to reduce the traversal of states.

9. CONCLUSIONS AND FUTURE WORK

In this paper we showed that it is feasible to include four well known SBO techniques in rule-based model transformations. Candidate solution are intuitively created in the language of the domain experts using transformation rules. The search techniques are incorporated in the scheduling language of the model transformation. We identified the different transformation language features that make this approach possible. Using a resource allocation example, we show that the proposed techniques work as well as a hand made coded implementation of the same problem. Furthermore, the run-time performance is compared for the running example between three different types of model transformation matching techniques and separately with the more optimal coded implementation. In our example, Search Plans is the most promising technique for including search in model transformations. Compared to a more optimal coded implementation, the model transformations are more computationally expensive but can be used when it is hard, time-intensive or impossible to create an optimal search amenable representation of the software engineering problem at hand.

Our next steps in including search in transformation models include improving the current performance of the approach. We will focus on creating a non-deterministic and random matcher, so a single non-deterministic or random match can be found in the model without the need of matching all the possible neighbours of a solution. Another optimization uses the notion of locality. Partitioning models into scopes can improve run-time performance.

10. REFERENCES

[1] Douglas Schmidt. Model-Driven Engineering. *Computer*, (February):25–31, 2006.

[2] T. Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5(4):369–385, 2006.

[3] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.

[4] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering. *ACM*

Computing Surveys, 45(1):1–61, November 2012.

[5] F. Burton and S. Poulding. Complementing Metaheuristic Search with Higher Abstraction Techniques. In *1st Workshop on Combining Modelling and Search-Based Software Engineering*, 2013.

[6] M. Harman, R. Paige, and J. Williams. 1st international workshop on combining modelling and search-based software engineering (CMSBSE 2013). In *Proceedings of the 2013 International Conference on Software*, number Cmsbse, pages 1513–1514, 2013.

[7] Levi Lucio, Sadaf Mustafiz, Joachim Denil, Maris Jukss, and Hans Vangheluwe. FTG + PM : An Integrated Framework for Investigating Model Transformation Chains. In *SDL 2013: Model-Driven Dependability Engineering*, pages 182—-202. Springer, 2013.

[8] Sadaf Mustafiz, Joachim Denil, Levi Lúcio, and Hans Vangheluwe. The FTG+ PM framework for multi-paradigm modelling: An automotive case study. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, pages 13–18, 2012.

[9] Joachim Denil, Gang Han, Magnus Persson, Paul De Meulenaere, Haibo Zeng, Xue Liu, and Hans Vangheluwe. Model-Driven Engineering Approaches to Design Space Exploration. Technical report, McGill University, SOCS-TR-2013.1, 2013.

[10] Wei Zheng, Qi Zhu, Marco Di Natale, and Alberto Sangiovanni Vincentelli. Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems. *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 161–170, December 2007.

[11] Chung Laung Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[12] S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *International Conference on Software Engineering*, pages 512–521, 2013.

[13] Paul Emberson and Iain Bate. Minimising Task Migration and Priority Changes in Mode Transitions. *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 158–167, April 2007.

[14] G. Antoniol, M. Di Penta, and M. Harman. Search-based techniques for optimizing software project resource allocation. In *Genetic and Evolutionary ComputationâĂŞGECCO 2004*, pages 1425–1426. Springer Berlin Heidelberg, 2004.

[15] Eugene Syriani, Hans Vangheluwe, and Brian LaShomb. T-Core: a framework for custom-built model transformation engines. *Software & Systems Modeling*, August 2013.

[16] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[17] Luigi Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–72, October 2004.

[18] Akos Horvath, Gergely Varró, and Dániel Varró. Generic Search Plans for Matching Advanced Graph Patterns. *Electronic Communications of the EASST*, 6(International Workshop on Graph Transformation and Visual Modeling Techniques), 2007.

[19] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(3597):17–37, 1982.

[20] Gábor Bergmann, András Ökrös, István Ráth, Dániel Varró, and Gergely Varró. Incremental pattern matching in the viatra model transformation system. *Proceedings of the third international workshop on Graph and model transformations - GRaMoT '08*, page 25, 2008.

[21] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[22] Arend Rensink and Jan-hendrik Kuperus. Repotting the Geraniums : On Nested Graph Transformation Rules Repotting the Geraniums : On Nested Graph Transformation Rules. *Electronic Communications of the EASST*, 18, 2009.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[24] Haibo Zeng and Marco Di Natale. Improving Real-Time Feasibility Analysis for Use in Linear Optimization Methods. *2010 22nd Euromicro Conference on Real-Time Systems*, pages 279–290, July 2010.

[25] A.H. Land and A.G. Doig. An Automated Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, 1960.

[26] Maged Elaasar Maris Jukss, Clark Verbrugge and Hans Vangheluwe. Scope in model transformations. Technical report, School of Computer Science, McGill University.

[27] Marouane Kessentini, Manuel Wimmer, Houari Sahraoui, and Mounir Boukadoum. Generating transformation rules from examples for behavioral models. In *Proceedings of the Second International Workshop on Behaviour Modelling Foundation and Applications - BM-FA '10*, pages 1–7, New York, New York, USA, 2010. ACM Press.

[28] M. Kessentini, P. Langer, and M. Wimmer. Searching Models, Modeling Search. In *Proceedings of the 1st Workshop on Combining Modelling with Search-Based Software Engineering*, pages 51–54, 2013.

[29] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis. In *Embedded Software*, pages 290–305. Springer, 2003.

[30] Tripti Saxena and Gabor Karsai. MDE-based approach for generalizing design space exploration. In *Model Driven Engineering Languages and Systems*, pages 46–60. Springer, 2010.

[31] Twan Basten and Emiel Van Benthum. Model-driven design-space exploration for embedded systems: the octopus toolset. In *LEVERAGING APPLICATIONS OF FORMAL METHODS, VERIFICATION, AND VALIDATION, LNCS*, pages 90–105. Springer, 2010.

[32] B. Schätz, F. Hölzl, and T. Lundkvist. Design-Space

Exploration through Constraint-Based Model-Transformation. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 173–182. IEEE, 2010.

[33] Joachim Denil, Antonio Cicchetti, Matthias Biehl, Paul De Meulenaere, Romina Eramo, Serge Demeyer, and Hans Vangheluwe. Automatic Deployment Space Exploration Using Refinement Transformations. *Electronic Communications of the EASST Recent Advances in Multi-paradigm Modeling*, 50, 2011.

[34] A. Hegedus and A. Horváth. A model-driven framework for guided design space exploration. In *Automated Software Engineering (ASE), 2011*, number i, 2011.