An Experimental Evaluation of Design Space Exploration of Hardware/Software Interfaces

Thomas Rathfux[1], Hermann Kaindl[1], Ralph Hoch[1] and Franz Lukasch[2]
[1]*Institute of Computer Technology, TU Wien, Vienna, Austria*
[2]*Robert Bosch AG, Göllnergasse 15-17, Vienna, Austria*

Keywords: Model-driven Engineering, Design Space Exploration, Reuse, Heuristic Search, Hardware/Software Interfaces.

Abstract: We observe ever increasing variability of hardware/software interfaces (HSIs), e.g., in automotive systems. Hence, there is a need for the reuse of already existing HSIs. In this regard, an important question is whether automated adaptation of an already existing HSI to one that fulfills the requirements on a new HSI is feasible in industrial practice. Ideally, the number of adaptation steps should be minimal, so that new hardware production can be avoided. In this paper, we address the problem of finding such an *optimal solution* for a given specific HSI and a set of formally specified requirements on a new HSI. We propose using *design space exploration* employing (heuristic) search with optimality guarantees. Hence, a meta-model of such HSIs has been created together with transformation rules. Based on all that, an experimental evaluation of this approach shows its feasibility for realistic HSIs.

1 INTRODUCTION

Software has become increasingly important in cyber-physical systems, which are actually software-intensive systems in many domains. For instance, in today's cars *electronic control units* (ECUs), i.e., embedded systems are ubiquitous in large numbers.

One important functionality of such ECUs is that they serve as hardware/software interfaces (HSIs). Both sensors and actuators are connected to ECUs, and the HSIs need to make sure that the respective signals are correctly represented in the software. For example, in an automotive system like a car, a pedal position sensor is connected to the corresponding ECU. Some pedal position sensors deliver analogue, others digital signals. Depending on what kind of sensor is being connected to the ECU, a differently configured HSI must be used.

Since there are many variations possible like that, a variability problem has become important. This variability can be addressed by *reuse*. One possibility is to take already existing HSI specifications as *reusable assets*, and to attempt reusing them by automated adaptation of an already existing HSI (more precisely, its specification) to one that fulfills the requirements on a new HSI.

For finding such a new HSI, *(heuristic) search* may be employed, see, e.g., (Pearl, 1984). It tries out different adaptation possibilities and, once it comes across an HSI that fulfils the requirements, it has found a *solution*. Of course, the requirements must be available in a formal representation, which can be used as goal conditions for the search.

This is reminiscent of *search-based software engineering*, a notion coined in (Harman and Jones, 2001). Usually, search approaches such as genetic algorithms are employed there for finding solutions to very complex problems, but they cannot normally find optimal solutions. Even if they find optimal solutions, they cannot prove that these solutions are indeed optimal.

Actually, the search space must be formally represented as well, in terms of its states and its possible transformations from one state to the other. Programming such a search space directly, as often done for puzzles and games, would take unreasonable effort, however. *Model-driven engineering* offers the possibility of representing such a search space by defining meta-models and transformation rules. More specifically, we use the approach to *design space exploration* as exemplified in VIATRA2 (Hegedus et al., 2011).

Since for cost reasons hardware production should be kept minimal, the number of adaptation steps should be kept minimal. Hence, we need to em-

Rathfux, T., Kaindl, H., Hoch, R. and Lukasch, F.
An Experimental Evaluation of Design Space Exploration of Hardware/Software Interfaces.
DOI: 10.5220/0007689002890296
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 289-296
ISBN: 978-989-758-375-9

ploy search for *optimal solutions*, i.e., sequences of adaptation steps with a guaranteed minimum number of steps. Unfortunately, such a search typically has exponential complexity, in our case with an average branching degree of about one-hundred.

This raises the question of whether this approach is feasible under realistic conditions such as those in industrial practice. For answering this question, we performed an experimental evaluation with a model realistic for automotive systems, more precisely a meta-model, for enabling design space exploration.

The remainder of this paper is organized in the following manner. First, we provide some background and discuss related work, in order to make the paper self-contained. Then we present our (meta-)modeling approach for design space exploration of HSIs. Based on that, we explain our search approach and define our heuristic function. After that, we present our experiment using (heuristic) search and its results. Finally, we conclude and indicate future work.

2 BACKGROUND AND RELATED WORK

First, we sketch the ECUs that our HSIs are implemented on, and the essence of the HSIs themselves as far as needed for this paper. Then we refer to the model-driven tool VIATRA2, which we use for design space exploration. Since we perform this exploration using heuristic search, we also explain it here briefly.

2.1 ECUs and HSIs

ECUs are commonly used to provide functionality of hardware and software components for external systems that they are embedded in, e.g., in the automotive domain. For this purpose, each ECU provides an HSI, which enables external hardware components to interact with internal software functions. This software typically runs on a microcontroller and uses various resources, which are made available through the pins of the microcontroller.

An ECU may contain different building blocks, but commonly includes a microcontroller and its internal resources alongside with its pins, hardware components for signal processing and ECU pins. Internally, these building blocks are (potentially) connected to others through the wiring of the ECU. For external connections to other hardware, the ECU pins are used (and not directly the microcontroller pins).

Figure 1 shows the building blocks of a typical ECU as used in the automotive domain, as well as their connections. On the right, the microcontroller is depicted, which contains the possible connections between its pins and its resources. Note, that this figure only illustrates the essential structure, while a real ECU has many more building blocks and connections between them, and many more resources and their connections within the microcontroller. Hence, a real ECU is both larger and more complex, but for the purposes of the explanations in this paper, this schematic illustration should be sufficient.

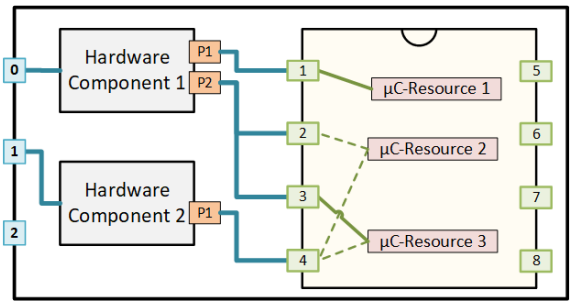

Figure 1: Schematic illustration of an ECU and its HSI.

Apart from the microcontroller, which runs the software, the hardware components represent any type of hardware that is used to process input or output signals (e.g., power output states, low- or high-pass filters). They are both connected to ECU pins and to one or more microcontroller pins (μC-pins). The latter connections are through ports (e.g., P1, P2), and define the microcontroller resources (e.g., μC-Resource 1, μC-Resource 2) that are potentially available at a port. The microcontroller makes its resources accessible through its μC-pins. One resource may be connected to several μC-pins, and one μC-pin to several resources.

It is very important for defining HSIs, that inside the microcontroller certain of these connections can be configured through activating some of the potential connections, or not. Figure 1 illustrates potentially available connections as dashed lines, and currently activated connections as bold solid lines. Such resources can be an analogue digital converter (ADC), timer input module (TIM), etc. For example, in Figure 1, μC-Resource 2 is connected to μC-pin 2 and μC-pin 4.

It is important that the right resources are connected to some hardware component so that it can provide a certain interface type, such as analogue measurement of input signals, pulse width modulation measurement, etc. All hardware components together with their connected resource configurations define an HSI, where the software communicates with the configured resources in terms of digital information.

2.2 VIATRA2

VIATRA2 is a model-driven framework for design space exploration. This framework supports defining search strategies for traversing the design space, starting from an initial model by applying rules. VIATRA2 allows defining rules based on the meta-model used. They are applied throughout the search to explore the design space. Goals are defined as conditions that must be satisfied for a solution.

VIATRA2 as used in our work presented here, is actually just one tool of a set of tools developed over time, where different tools provide different techniques for design space exploration (Bergmann et al., 2015).

2.3 Heuristic Search for Optimal Solutions

Many search algorithms have been presented in the literature, so it would be prohibitive to review all of them here. Rather, we focus on those we use in the experiment reported in this paper, a (unidirectional) search algorithm with certain optimality guarantees and a special case of it.

The traditional *best-first* search algorithm A* (Hart et al., 1968) maintains the set OPEN of so-called *open* nodes that have been generated but not yet expanded, i.e., the frontier nodes. Much as any best-first search algorithm, it always selects a node from OPEN with minimum estimated cost, one of those it considers "best". This node is expanded and moved from OPEN to CLOSED. A* specifically estimates the cost of some node n with an evaluation function of the form $f(n) = g(n) + h(n)$, where $g(n)$ is the (sum) cost of a path found from s to n, and $h(n)$ is a heuristic estimate of the cost of reaching a goal from n, i.e., the cost of an optimal path from s to some goal t. If $h(n)$ never overestimates this cost for all nodes n (it is said to be *admissible*) and if a solution exists, then A* is guaranteed to return an *optimal* (minimum-cost) solution (it is also said to be *admissible*). Under certain conditions, A* is optimal over admissible unidirectional heuristic search algorithms using the same information, in the sense that it never expands more nodes than any of these (Dechter and Pearl, 1985).

Since A* needs an admissible heuristic evaluation function, which is often not easy to find for problems in industrial practice, let us also mention the special case of A* without using heuristic knowledge, i.e., $h(n) \equiv 0$, or simply $f(n) = g(n)$. In the special case of unit costs, i.e., the cost of each step on each path is exactly 1, this implements *breadth-first* search. Note, that breadth-first search can also be implemented by the traditional shortest-path algorithm due to (Dijkstra, 1959).

3 MODELING APPROACH

First, we present (part of) our meta-model as used by VIATRA2 for design space exploration. In order to enable it, we next define transformation rules based on this meta-model, and goal conditions as given in formally specified requirements for a new HSI. Finally, we explain how the search approach is implemented in VIATRA2, and define our admissible heuristic for guaranteeing optimal solutions.

3.1 Meta-model

Based on the sketch of the application domain above, we specify here in the meta-model generically what is needed for design space exploration. This meta-model covers the structure of an ECU, the possible variations for activating connections for a configuration, and the currently selected configuration. Since the searches for design space exploration must know when a solution is found, they must be given goal conditions. These are defined in the formally represented *Requirements*, which are also generically included in the meta-model. Technically, we created this meta-model in Ecore and the Eclipse Modeling Framework (EMF) (Eclipse, 2017; Steinberg et al., 2009), as this meta-modeling approach is directly supported by the VIATRA2 tool.

Figure 2 shows selected parts of the meta-model that are essential for the purpose of explaining the design space exploration. The following classes and their associations define the application domain: ECU pins (EcuPin), hardware components (HardwareComponent, HardwareComponentPort) and the microcontroller (Microcontroller, McResource, McHwPin, McPin, McPinConnection). Each instance of a HardwareComponent is connected to a McPin instance through HardwareComponentPort instances. The same applies for the McResources of the Microntroller and the Ecu. Different interface types on hardware components are expressed via the interfaceType attribute of the HardwareComponent class.

Each instance of the meta-model represents a specific ECU model (and the Requirements on its HSI). Actually, it also has the HSI configuration defined, i.e., such an instance represents a specific state in the search for design space exploration.

In addition, the meta-model defines all the possibilities for variation and what is needed for defining the transformation rules, see below. The variations

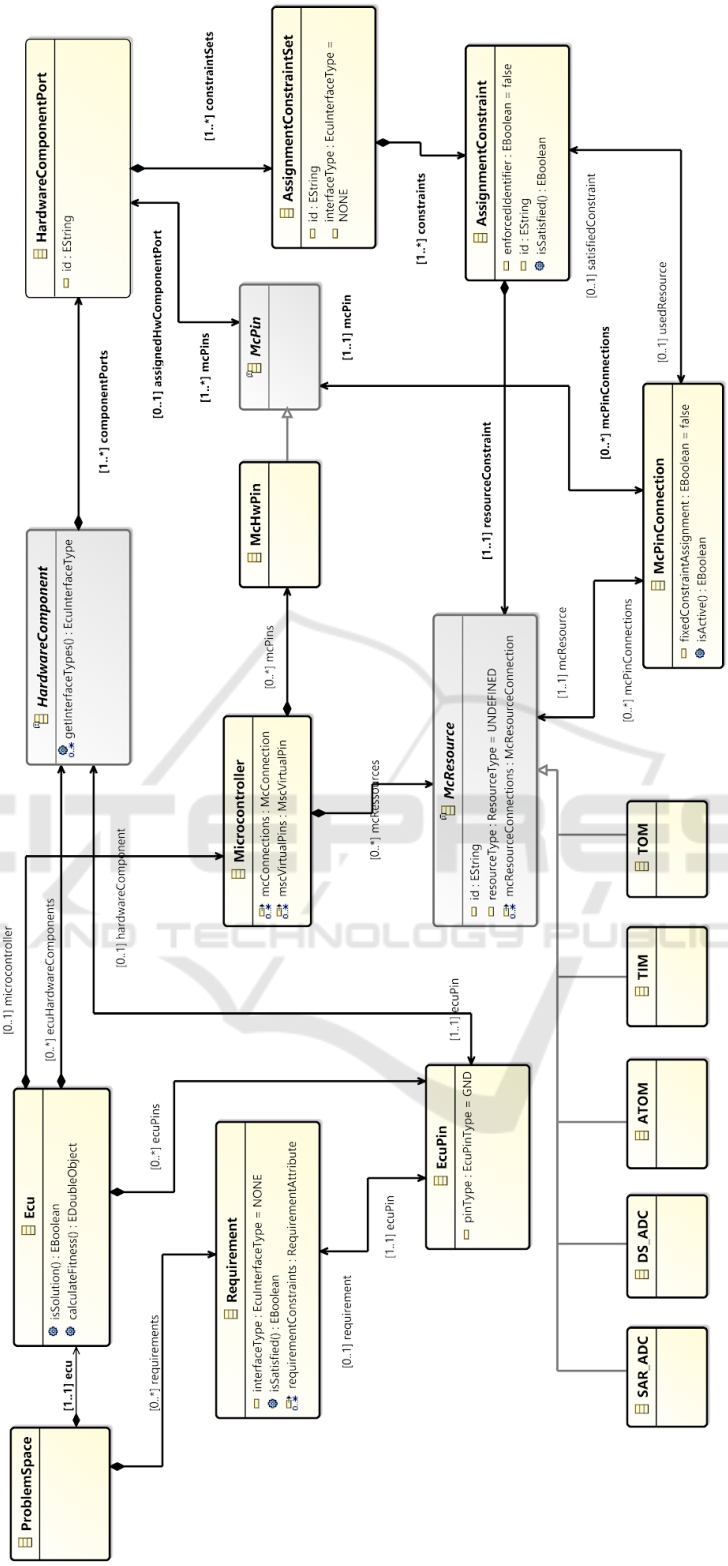

Figure 2: Selected parts of our meta-model.

are expressed through instances of the McPinConnection class, which connects instances of McResources to instances of McPins and, consequently, to HardwareComponentPorts. For defining transformation rules, it is necessary to specify what kind of resources are needed by hardware components. In the meta-model, this is represented by the classes AssignmentConstraintSet and AssignmentConstraint. Each HardwareComponentPort may contain several AssignmentConstraintSets with AssignmentConstraints and one specific AssignmentConstraint specifies which (type of) McResource is needed. This restricts the variability of HSIs.

The currently configured HSI is defined through the association *usedResource*, which specifies if a connection to a resource is activated or not. It links McResources on McPins through McPinConnections and AssignmentConstraint(Set)s to HardwareComponentPorts, and, consequently, to EcuPins. Each value change on usedResource transforms a state in the search for design space exploration to another state.

Formally represented Requirements specify the goal conditions. Each Requirement is associated with a specific EcuPin and contains an attribute of type EcuInterfaceType for specifying what kind of interface is required on this pin.

3.2 Transformation Rules

Based on the variation expressed in the meta-model, we defined transformation rules in VIATRA2 (using its query language). Essentially, they model two kinds of transformations, one for activating a connection in a configuration, and one for deactivating a connection.

Figure 3 illustrates both kinds of transformation rules. The transition from (a) to (b) shows the deactivation of the connection from μ-pin 4 to ADC 2, the transition from (b) to (c) the activation of the connection from μ-pin 3 to TIM 1.

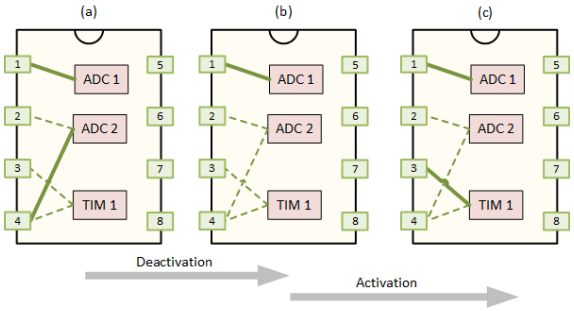

Figure 3: Illustration of transformation rules for HSIs.

However, as hardware components have specific assignment constraints, not all connections are activated or deactivated arbitrarily. Each transformation is only applied in the course of the search if it may lead to satisfying a goal condition, i.e., fulfilling a given requirement.

Technically, such a transformation rule as defined in VIATRA2 consists of three parts: pre-condition, transformation and post-condition. The pre-condition specifies the applicability of a transformation rule, in our application in terms of the constraints of HardwareComponentPorts. The transformation specifies the changes in the model, i.e., the instance of the meta-model that the transformation rule is applied to. The post-condition specifies the result of the rule application as a condition defined according to the meta-model.

3.3 Goal Condition

As indicated above, a goal condition for a search is defined by Requirements (as specified in our meta-model). In general, more than one Requirement is given in this way, and a solution is only found, if and when all corresponding conditions are satisfied. That is, a goal condition is a *conjunction* of conditions given by Requirements.

Depending on the EcuInterfaceType attribute of a given Requirement, the HardwareComponent at the specific EcuPin that this Requirement is associated with needs to fit. Consequently, all constraints of this HardwareComponent have to be fulfilled as well. This requires the fulfillment of AssignmentConstraintSets on HardwareComponentPorts as they also depend on the EcuInterfaceType. One EcuInterfaceType may be supported by more than one AssignmentConstraintSet. In addition, one HardwareComponent and its HardwareComponentPorts may support more than one EcuInterfaceType. Thus, specifying more than one AssignmentConstraintSet per HardwareComponentPort, each related to a different EcuInterfaceType, is also possible. If at least one of these AssignmentConstraintSets is fulfilled for a specific EcuInterfaceType on all Ports, then also the specific Requirement is fulfilled and, therefore, the corresponding part of the goal condition. That is, all ports of a HardwareComponent are connected via *conjunction* and all AssignmentConstraintSets of a port are connected via *disjunction*.

However, the AssignmentConstraints of a particular AssignmentConstraintSet have all to be fulfilled. Hence, they are connected via *conjunction*.

Summarizing, one goal condition of a specific EcuInterfaceType is logically expressed in *disjunctive normal form*. If one part of the disjunctive normal form is fulfilled, then the interface type may be used.

This corresponds to one Requirement only, however. Hence, such a disjunctive normal form has to be fulfilled for each and every Requirement.

4 SEARCH APPROACH

With this modeling approach, a search space is defined for design space exploration using the VIATRA2 tool. The transformation rules can be chained together in sequences. Figure 3 shows an example where first a deactivation of one connection is followed by the activation of another one. In this way, several adaptations of an HSI design can be achieved in a model. A sequence of transformations that connects a start configuration of an already existing HSI with another one that satisfies a goal condition defined through given Requirements is a *solution*. For example, it may be necessary to deactivate a connection and activate several others in succession. A solution with minimal cost, in our case a minimal number of transformations is an *optimal* solution. Since finding a solution, in particular an optimal one, is not straight-forward, in general, alternative transformations need to be investigated, and this leads to a search in this space. For this design space exploration, our search approach is breadth-first search (without heuristic) and A* (with heuristic) as reviewed above.

Such a search for an optimal solution typically has exponential complexity, in our case with an average branching degree of about one-hundred. Hence, this search space is very large for realistic HSIs. Fortunately, it has important properties that need to be utilized for making such searches feasible. Figure 4(a) illustrates a *cycle*. It may simply occur after activating a particular connection in the configuration of state S1 and subsequently deactivating the very same connection in the configuration of state S2. This results in state S5, which is the same as state S1, of course. Therefore, further search below S5 is not necessary and can be pruned. Figure 4(b) illustrates a *directed acyclic graph (DAG)*, where the same state S5/S6 is visited more than once, when reached from the root S1 via two (or, in general, several) different paths. This may simply occur after activating a particular connection c_1 in the configuration of state S1 and subsequently deactivating another connection c_2 in the configuration of state S2, where deactivating c_2 in the configuration of state S1 and subsequently activating c_1 in the configuration of state S3 leads to the same configuration, of course. Cycles and DAGs can be recognized by the VIATRA2 tool, so that the searches can be effectively pruned for achieving very strong reductions of the search costs. This makes the searches much more efficient both in terms of space and time.

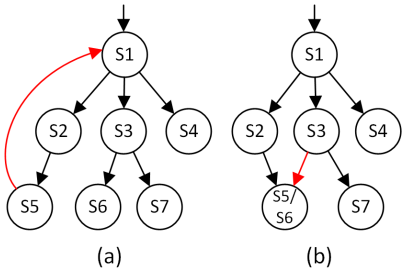

Figure 4: Cycles and DAGs in the search space.

Breadth-first search is implemented directly in VIATRA2, while for best-first search an evaluation function has to be defined. Since we implemented A*, this evaluation function is $f(n) = g(n) + h(n)$ as reviewed above. Note, that using $f(n) = g(n)$ leads to an alternative implementation of breadth-first search, since there are unit costs for all transformations. A* with a heuristic h usually searches more efficiently than breadth-first search.

For A*, *admissibility* of the heuristic function is important for guaranteeing the optimality of solutions found. For evaluating a configuration in such a function with respect to its goal achievement, the number of not (yet) fulfilled conjunctively related goal conditions is counted. In case of disjunctively related conditions, the minimum is taken. The resulting number can be used as the heuristic value, since each condition needs at least one application of a transformation rule. In fact, these can only be activation rules. Deactivation rules may additionally be necessary, in order to deactivate some connection so that another one needed can be activated at this particular pin. Consequently, this number is less than or equal to the number of minimal steps to achieve the goal condition, i.e., this is an admissible heuristic. This can also be explained more theoretically based on the *meta-heuristic* of problem relaxation, see (Pearl, 1984). A relaxed problem would only need activation rules for its solution, i.e., the number calculated by our heuristic function.

5 EXPERIMENT

First, we present our design of the experiment, and then its results.

5.1 Experiment Design

The purpose of this experimental evaluation is purely exploratory, answering the question of feasibility of this approach under realistic conditions such as those in industrial practice, more precisely for reusing HSI designs of automotive ECUs. Hence, we had to make sure in the experiment design that the given HSI designs and the requirements for new ones are realistic.

Since statistical fluctuation was to be expected for different instances, we defined several ones and included some randomness into their creation. In addition, we wanted to systematically get data on the average running times for different (optimal) solution lengths, in order to evaluate the effect of scaling with increasing problem difficulty.

First, we defined a specific ECU hardware based on typical hardware components available on the PCB of an ECU in our domain. For the microcontroller, we defined a resource set inspired by a typical 64-pin ARM microcontroller (STM, 2018; Infinion, 2018). For creating differently configured ECUs, ECU-pins of this first ECU were selected and requirements randomly assigned to them. For each of them, a solution was determined (through searches) and stored as a base model for the next step of model creation.

Then we created problem instances with different solution lengths by generating requirements randomly (again). This resulted in 75 problem instances for each solution length (up to 20 steps), with different starting configurations and different target requirements. From this set, ten problem instances each were selected randomly and used for the searches in the experiment. Since we experienced some variation in the runs of VIATRA2 also regarding the ordering during design space exploration, which influences the running time depending on when a solution is found, we ran each problem instance five times and calculated mean values.

We executed the experiment runs on a standard Windows laptop computer with an Intel Core i7-8750H Processor (9MB Cache, up to 4.1 GHz, 6 Cores). It has a DDR4-2666MHz memory of 32GB. The disk does not matter, since all the experimental data were gathered using the internal memory only.

5.2 Results

On the laptop computer used, we were able to collect the results of searches with optimal solution lengths (costs) C* of up to six for breadth-first search and of up to 17 for A*. All the searches found optimal solutions and guaranteed that they are optimal (in terms of solution length). Figure 5 plots the mean running times for both kinds of search in seconds (where no other processes were running on the laptop computer used). In fact, breadth-first search shows up in two plots, one for running on six cores in parallel and one on a single core. A* was running on a single core (only), since VIATRA2 does not support parallel execution of searches with cost functions.

In more detail, Table 1 shows the data gathered from the multi-core runs, including the mean numbers of states visited. Table 2 shows the data of the breadth-first searches on a single core, and Table 3 the detailed results of the A* searches (on a single core). The number of states visited per second shows minor fluctuations (presumably due to technical reasons inside the VIATRA2 engine), but there is no systematic deviation due to the running time or the search depth. The factors of running times (or the numbers of nodes visited) from level i to $i+1$ of C* are an order of magnitude less than the average branching degree for breadth-first search, and two orders less for A*. The reasons are the DAGs in the search space, which can be exploited even much more by best-first search, since it is directed to a goal state and can search much deeper.

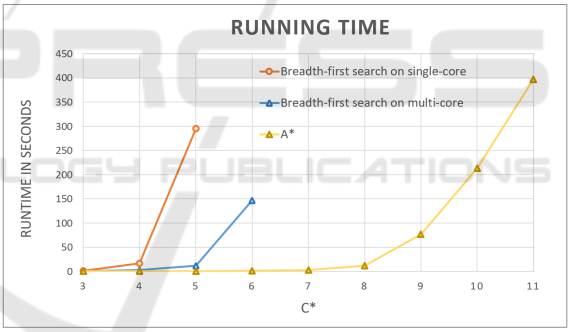

Figure 5: Comparison of running times of breadth-first search vs. A*.

Table 1: Summarized results of breadth-first search executed on six cores.

C*	Running time	Running time factor	No. of visited states	No. of states per second
3	0.30	–	1,160	3,890
4	2.73	9.17	12,680	4,640
5	11.54	4.22	32,287	2,800
6	146.89	12.73	301,830	2,050

It is easy to see in Figure 5 that A* is much more efficient than breadth-first search in terms of running time, even when running on a single core as compared to six cores. Breadth-first search running on six cores can just find optimal solutions with a maximum length of six, while A* (running on a single

Table 2: Summarized results of breadth-first search executed on a single core.

C*	Running time	Running time factor	No. of visited states	No. of states per second
3	0.95	–	1,327	1,399
4	16.98	17.91	14,739	868
5	295.29	17.39	214,624	727

Table 3: Summarized results for A*.

C*	Running time	Running time factor	No. of visited states	No. of states per second
3	0.11	–	133	1,262
4	0.13	1.28	289	2,144
5	0.53	3.96	1,075	2,016
6	1.20	2.26	2,677	2,225
7	3.04	2.52	6,690	2,204
8	12.15	4.00	21,848	1,799
9	76.91	6.33	106,518	1,385
10	213.65	2.78	328,704	1,539
11	396.91	1.86	604,839	1,524

core) can do so nearly twice as deep. Hence, the admissible heuristic serves well its purpose of making its searches much more directed and, hence, efficient than breadth-first search without any heuristic.

6 CONCLUSION AND FUTURE WORK

The question is now, whether these search results indicate the feasibility of this approach for reusing realistic HSI designs. We think that breadth-first search does not qualify due to the shallow searches it was able to perform. It will most likely be insufficient in practice to find optimal solutions of, say, six adaptation steps. However, A* can search much deeper, and this makes it feasible for such reuse. This shows the importance of the heuristic knowledge involved.

Of course, the actual applicability for real-world problems will yet have to be shown in a case study. Actually, we also plan to address the problem of automatically finding a previous HSI that is similar to the new one for which the requirements are given. This may lead to *case-based reasoning* where both finding a similar previous case and the adaptation of its stored (previous) solution to the new one is automated.

ACKNOWLEDGEMENTS

The InteReUse project (No. 855399) is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program “ICT of the Future” between September 2016 and August 2019. More information can be found at https://iktderzukunft.at/en/.

REFERENCES

Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., and Varró, D. (2015). Viatra 3: A reactive model transformation platform. In Kolovos, D. and Wimmer, M., editors, *Theory and Practice of Model Transformations*, pages 101–110, Cham. Springer International Publishing.

Dechter, R. and Pearl, J. (1985). Generalized best-first strategies and the optimality of A^*. *J. ACM*, 32(3):505–536.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. In *Numerische Mathematik 1*, pages 269–271.

Eclipse (2017). Package org.eclipse.emf.ecore (emf javadoc).

Harman, M. and Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics (SSC)*, SSC-4(2):100–107.

Hegedus, A., Horvath, A., Rath, I., and Varro, D. (2011). A model-driven framework for guided design space exploration. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE ’11, pages 173–182, Washington, DC, USA. IEEE Computer Society.

Infinion (2018). *STM32 32-bit Arm Cortex product family*.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

STM (2018). *Infineon AURIX^{TM} Family TC29xT*.