International Journal of Parallel Programming (2024) 52:59–92
https://doi.org/10.1007/s10766-024-00763-2

Investigating Methods for ASPmT-Based Design Space Exploration in Evolutionary Product Design

Luise Müller[1] · Philipp Wanko[2] · Christian Haubelt[1] · Torsten Schaub[2]

Received: 30 January 2023 / Accepted: 19 January 2024 / Published online: 24 February 2024

Abstract

Nowadays, product development is challenged by increasing system complexity and stringent time-to-market. To handle the demanding market requirements, knowledge from prior product generations is used to derive new, but partially similar product versions. The concept of product generation engineering, hence, allows manufacturers to release high-quality products within short development times. Therefore, in this paper, we propose a novel approach to evaluate the similarity of two product implementations based on the concept of the Hamming distance. This allows the usage of similarity information in various heuristics as well as in strategies and thus, to improve the product design process. In a wide set of cases, we investigate the quality and similarity of design points. In the experiments, the use of strategies leads to significantly short searching times, but also tends to be too restrictive in certain cases. Simultaneously, the quality of the solutions found in the heuristic design space exploration has been shown to be as good or better than for the search from scratch and considerably closer solutions as part of the non-dominated solution front have been found.

Keywords Design space exploration · Evolutionary product design · Strategies · Heuristics

✉ Luise Müller
luise.mueller@uni-rostock.de

Philipp Wanko
wanko@cs.uni-potsdam.de

Christian Haubelt
christian.haubelt@uni-rostock.de

Torsten Schaub
torsten@cs.uni-potsdam.de

[1] Applied Microelectronics and Computer Engineering, University of Rostock, Rostock, Germany

[2] Department of Computer Science, University of Potsdam, Potsdam, Germany

1 Introduction

Nowadays, electronic systems are used in a wide range of applications. Analogous to the number of application areas, the complexity of such systems, regarding the number of included components or their heterogeneity, is increasing significantly. When designing an electronic system, due to that, a vast amount of design options is available. Given a specification, which defines the functionality of the system, and the available resources for its execution, there is an enormous number of design decisions to be made. This includes the allocation of hardware resources, namely the processing units and the communication infrastructure, the binding of functionalities to the allocated architecture and the scheduling.

On the one hand, a design decision must lead to a valid system implementation, i.e., it must guarantee the desired functionality. On the other hand, each design decision affects the quality characteristics of the resulting system, like the energy consumption or the area costs. Obviously, the quality of the implementation shall be optimized.

Due to demanding market requirements on innovation, product quality and short product development time, an exhaustive exploration of all possible designs is not a feasible option. This is why, an efficient design space exploration (DSE) is indispensable. Besides, an electronic system often does not have to be developed from scratch because domain knowledge can be gained during the development process of an earlier product version. In reality, new products are generally based on a reference configuration [1] and numerous variants can form an entire product line. Major parts of the specification are in common in different versions of the product or are carried over from one generation to the next one. Like this, only small parts of the system need to be designed from scratch. Also, the verification and testing tasks are limited to the new parts and their integration. This form of product design is called *product generation engineering (PGE)* and aims at fulfilling the requirements of different target groups while, simultaneously, reducing the cost of the production [1]. Further, this product development process can be seen as an evolutionary process. Step by step, generation by generation, modifications, due to changing market demands or technological improvements, are made. By adapting to these while adopting good features, the product can be optimized from generation to generation.

Also, in the automotive sector, the concept of PGE and product lines is commonly used. Assume, for example, a car manufacturer. It offers various basic car models, classified by categories like compact cars, SUVs or vans. For each one further choices concerning e.g. the engine type or the supported drive mode are available. But the amount of possible designs is not limited to these categories, because a basic product type can be extended by several assistance systems or smart safety features. Starting with a cruise control feature, going further to an automatic emergency brake system and even up to a self-driving car, the number of possible extensions is increasing over the years. To show it in concrete numbers, for example, the amount of extra functions offered in the BMW 7 series raised from 14 extras in 1986 to 92 in 2006 [2]. In another example, taking all decision options and extension possibilities together, there are more than 10^{23} final design options for a Volkswagen Golf [2].

With a high number of possible configurations and a vast hardware environment the design process is a complex and expensive task. It is highly demanded to use knowledge from prior product generations through the concept of PGE. In addition, to mitigate the complexity in the exploration process, we consider the system at a high degree of abstraction, namely the *electronic system level (ESL)*. In reality, selected design solutions are gradually refined with an increasing level of detail in the subsequent process.

This work at hand is an extended version of our prior work [3]. Our prior work [3] has taken up the idea of evolutionary product design and tried out an first approach to exploit the similarity between two product versions. For different kind and percentages of changes, in general, a heuristic-based DSE helped to find good solutions earlier and was particularly useful applied to large configurations. The results have shown us that the use of knowledge from prior versions of a product can improve the product design process. Because of the promising results of the previous presented approach, this paper adapts the idea of reusing similarity information but we formalize the first intuitive definition of the similarity in [3]. Further, instead of the use of one specific heuristic in [3], we investigate a wide range of different Answer Set Programming (ASP) methods. Our contribution is the following:

1. We provide a modular and clear definition of the similarity of two product implementations based on the concept of the Hamming distance. Using a declarative encoding based on ASP, this results in a succinct, countable and easily adaptable formulation.
2. The novel similarity evaluation allows the general applicability in a variety of heuristics and further, enables the use of similarity information in strategies. By constraining the search space or by steering the search towards promising solutions, the product design process is improved.
3. In massive experiments, we investigate the potential of ASP methods in the context of evolutionary design. Therefore, a high number of adjustment points is examined and the results are evaluated concerning their quality and similarity. The results indicate that strategies, due to short searching times, can be quite helpful in certain cases, but often tend to be too restrictive. Further, the search is enhanced by the use of domain-specific heuristics, since non-dominated solutions of high similarity can be found.

2 Related Work

Grimm [4] sees the handling of the increasing complexity as one of the core challenges in the development of hardware/software systems. For example in the automotive industry, with each new product version an increasing amount of functionalities is implemented as electronic or software modules to fulfill quality criteria like reliability, availability and safety as well as user friendliness and ease of use [4]. At the same time, the demand for new innovations in short as possible time-to-market requires an efficient product development process. Besides the use

of domain knowledge, a designer can learn from design decisions from the past. In previous works, the necessity of knowledge reuse in connection with product design has been discovered [4, 5] and the question of reusability addressed [6, 7]. But still, there is lack of formal guidelines and methods to assist the integration of existing modules into newly build systems [4, 5].

Possible approaches to enable design reuse, such as documentation, standardization, parameterization and modularization [5], are carried out to allow processes like design exchange, design evolution or component-based design [5, 6]. Additional methods and models for software reuse have been surveyed in [8].

In order to enhance the product development process, the concept of platform-based design utilizes prior design decisions and implementation details. Instead of developing each new version of a technical device from scratch, it is seen as one out of multiple related product variants, all basing on the same system platform [9]. This process enables the organization of the product portfolio into an entire product line with multiple variants of the system. Inside, all members share a common base. By applying small modifications, several product derivatives can be created, varying e.g. in the processing capabilities or in the battery sizes. According to [9], platform-based design enables the creation of either module-based or scale-based product families. The concept of product line engineering can lead to short development cycles and a flexible adaption to the customers needs.

Additionally, two recent literature reviews on product family design and platform-based product development can be found in [10, 11].

Further, the previously presented idea can be applied during the development process over several generations of a product. Albers et al. [1] specify the concept of *product generation engineering* as a reuse mechanism which is combined with significant new developments during the generation of a new technical product. Within, the development process of new product generations is always oriented to a reference product, that specifies the core structure [1]. Design decisions for future generations are based on the knowledge gained from predecessor versions. That way, strong aspects can be kept and weaknesses be cleared up.

Several companies in the automotive sector enhance their product development process by employing the approach of product line management. Concrete use cases are illustrated at the example of Porsche [1] and Volvo [12]. While the former example addresses the concept of PGE on a high abstraction level, the later one is considering the product as a embedded system, consisting of hardware and software aspects and is investigating the influence of changes on the technical platform as well. In [13] the authors evaluate the evolution in software product lines by comparing the example of two Swedish organizations. Therefore, they analyze the development of the software assets release by release. In the end, they give guidelines for the creation as well as for the evolution of software product lines. Further, the author Lim [14] has shown at two examples from within Hewlett-Packard that effective reuse can improve the product quality and increase the process productivity. Since, the costs needed to develop a reusable component are higher than for the development of a non-reusable version, the *economic return*, i.e. the trade-off between costs and benefits, is an important metric to consider by companies [14].

In this work, we address the system synthesis problem of a hardware/software system at the ESL. This problem has been successfully encoded using SAT-based [15, 16] and ASP-based techniques [17, 18]. In contrast to SAT, ASP is based on a closed-world assumption which allows an efficient implementation in particular for densely connected networks and multi-hop communication [17, 18]. For that reason, we have implemented our approach with ASP. ASP is a declarative programming language tailored towards NP-hard search problems and it offers a compact and modular representation, which allows an efficient solving of knowledge-intensive combinatorial problems.

As ASP does not scale for numerical and non-linear constraints [19], further approaches to handle timing constraints regarding the scheduling have to be considered. Thus, Andres et al. [20] and Biewer et al. [21] split the system synthesis tasks. The binding of computational units on a hardware platform and the routing of communications is solved by an ASP-solver, whereas the scheduling is decided by a SMT-solver based on *quantifier-free integer difference logic* (QF-IDL) [20, 21]. In a next step, Neubauer et al. [18, 19] have integrated QF-IDL into the background theory of an ASP-solver by using custom theory propagators [22] and thus, could improve the entire system synthesis.

The underlying encoding of our work is a re-implementation of [18] based on the state-of-the-art API of the ASP-solver clingo [23].

The DSE is a systematic search for feasible design points. During the solving process, the solver clingo employs a *conflict-driven clause learning (CDCL)* strategy as well as the heuristic *Variable State Independent Decaying Sum (VSIDS)* [24] in its default configuration. Since the underlying heuristic steers the search by means of the order of decisions to be taken, the performance of a DSE depends heavily on it. Therefore, it can be of great utility to equip the solver with domain specific knowledge. The solver clingo allows the implementation of user-defined heuristics [25]. Thus, in the paper at hand, we will use domain specific heuristics and constraints to accelerate the evolutionary product design process by augmenting the search with knowledge from prior product developments. In prior work [3], we made a first attempt, which delivered us promising results. Subsequently, in this approach, we improve the methodology for gaining knowledge to systematically explore various ASP methods.

A further approach utilizes domain-specific heuristics in ASP to improve the system synthesis, by exploiting the knowledge gained when analyzing e.g. the task graph structure [20]. Domain-specific heuristics have also been successfully applied to two hard industrial problems, namely the Partner Units Problem (PUP) [26] and the Combined Configuration Problem (CCP) [26, 27].

Another approach, which aims at exploiting domain knowledge in the context of system-level DSE of multi processor systems, is presented in [28]. The authors propose two new techniques to improve a GA-based search process, including a *mapping distance metric*. The metric evaluates the distance between a pair of sets of mappings, in means of the number of reassignments required to turn the one set of mappings into an equivalent image of the other one [28].

Also, the work of Richthammer et al. [29] focuses on the improvement of the system-level DSE in the context of multi-objective evolutionary algorithms. Since,

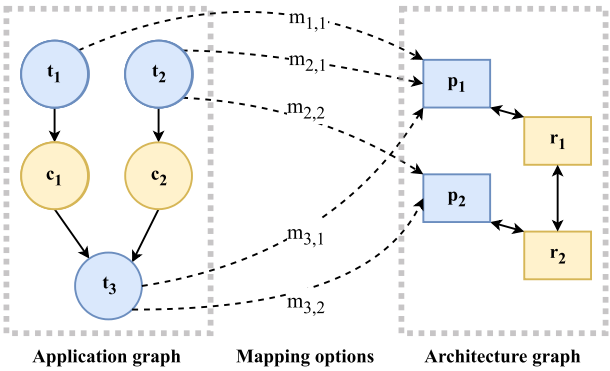

Fig. 1 An exemplary specification graph [3]

nature-inspired optimization approaches so far lack explainability [29], an automated data-mining technique is proposed to gain knowledge from the DSE results after each iteration of the evolutionary algorithm. The gained information, including the design decisions made and the quality of the respective solution, is used to incorporate, e.g. constraints into the next iteration of the search process to steer the search towards promising solutions [29].

When partially reusing systems or combining parts of different product versions, another important aspect to consider is the integration of an existing system into an other one. In this context, Neubauer et al. [16] address the composition of subsystems according to the idea of *systems-of-systems*. The authors introduce *hierarchical mapping edges* which represent the mapping options of a sub-application onto a set of resources. Subsequently, the system synthesis steps are redefined using the new concept so that the composition of subsystems and, especially, the integration of schedules is supported [16].

3 Fundamentals

In the following chapter, we formally describe our system model used throughout this paper. The graph-based problem model is inspired by work of [30] and it is used as input to the DSE. Subsequently, the underlying exploration problem is explained. We conclude this chapter with an introduction to two approaches for improving the exploration process, namely strategies and heuristics.

3.1 System Model

Our approach models a hardware/software system at a high degree of abstraction, specifically at the ESL. In Fig. 1 an example of the system specification $S = (A, H, M)$ is shown. It is a triple consisting of an application graph $A = (V_A, E_A)$, a hardware architecture graph $H = (V_H, E_H)$ and a set of mapping options M which interconnect both graphs. The application as well as the hardware architecture are modeled by

directed graphs. The vertex set of the application V_A is a bi-partition of two types ($T \cap C = \emptyset$), precisely computational tasks T and communication messages C. The set of communication edges $E_A \subseteq (T \times C) \cup (C \times T)$ defines the dependencies of task elements and thereby represents the data flow in the model. A communication always consists of a sending task, a communication message and a receiving task. In our system model, each message $c \in C$ is sent and received exactly once, i.e., $\nexists c : \{(c, t_i), (c, t_j)\} \subset E_A$ and $\nexists c : \{(t_i, c), (t_j, c)\} \subset E_A$. This establishes a point-to-point communication between individual tasks used to exchange data packages. All in all, the elements of the application model the desired behavior of the system.

The architecture is described as a 2-tuple $H = (V_H, E_H)$, consisting of a set of vertices $V_H = P \cup R$ with $P \cap R = \emptyset$, representing processing units P as well as routing units R, and a set of edges $E_H \subseteq V_H \times V_H$ linking the hardware elements. In Fig. 1 the two types of hardware devices are colored differently to draw attention to their differing functionality. While processing units are the targets for the execution of tasks from the application graph, the routing units are used to form a communication structure to enable the transfer of messages between two processing nodes. The communication between two processors is performed via intermediate routers as a multi-hop communication. Per message hop, a routing energy E_r and a routing delay δ_r is needed. The actual communication channels are established via the edges of the hardware architecture. Potentially, each device can be linked to any other device. Therefore, the implementation of various architecture topologies is enabled. In principle, even bus-based or mixed hardware architectures can be modeled. In this paper, our system model is based on networks on chip (NoC) with regular mesh topologies. Note that the bidirectional edges in Fig. 1 represent two individual links.

Further, each hardware device is parameterized by the functions $P_{stat} : V_H \to \mathbb{N}$ and $area : V_H \to \mathbb{N}$ representing static characteristics like power consumption and area costs.

Finally, a set of mapping options $M \subseteq T \times P$ defines the connection between the application and the architecture graph. For each task, it exists at least one mapping option $m \in M = (t_i, p_j)$, representing that the task t_i can be potentially executed on the processing unit p_j. The communication messages are not explicitly assigned to routing units, because they can be routed over the entire communication infrastructure. Their routing paths are only implicitly constrained by their sending and receiving tasks. For each mapping option $m = (t, p)$ the function $w : M \to \mathbb{N}$ defines the worst case execution time of the task t on the processing element p. Likewise, $E_{dyn} : M \to \mathbb{N}$ models the dynamic energy requirement of each mapping option. Finally, the specification is characterized by a periodicity $\mathcal{P}$ that specifies the time, after which the next iterative execution of the complete application is started. For simplicity, all properties in our approach are restricted to integer values. The proposed ASPmT-based [22] approach, however, in principle allows real-valued properties.

The behavior and properties of the system described in the specification are translated into a structural description, namely the implementation. For this, a valid allocation, binding, routing, and schedule have to be determined. The allocation $\alpha \subseteq V_H \cup E_H$ is a selection of devices and links from the architecture graph H which are used to realize the desired system behavior. Accordingly, α is separated into the

device and link allocation α_D and α_L. The static binding $\beta \subseteq M$ selects exactly one mapping option for each task, specifying on which component the task is actually executed. Similarly, the routing $\gamma \subset C \times 2^{E_H}$ chooses for each message a cyclic-free path in the communication structure, depending on the binding of the sending and receiving task. This cycle-free path is represented by a subset of allocated links (an element of the power set 2^{E_H}).

In this work, three approaches to decide on the routing path of a message are considered. The first idea is to define the route between a sending and a receiving task in a dimensional order (*encoding_xyz*). Using this restriction, only one routing path is allowed, but it is guaranteed to be the shortest path. In the second approach (*encoding_bound*), also the length of the route is restricted to the smallest number of hops possible, but in difference, the route is not fixed, so that alternative routes can be selected. Finally, the *encoding_arb* allows an acyclic routing path of arbitrary length and in arbitrary directions. This increases the complexity in terms of the number of decision variables. But it offers the possibility to distribute the communication traffic over the communication network.

The schedule τ assigns to each task and each communication message starting times for the execution on the allocated resources, i.e., $\tau : T \cup C \mapsto \mathbb{N}$. Note, that the start times are integer values as our entire model is based on integer parameters.

3.2 Exploration Model

Given a system synthesis problem, i.e., a specification S, the DSE searches for feasible implementations for that system. To reach a feasible solution, each design decision made needs to fulfill all requirements given by the specification and the previously defined system model. But in reality, not every feasible implementation is an useful solution. Due to that, each implementation x has to be evaluated regarding a set of desired objective functions. The focus in this paper lays on the overall latency $lat(x)$ of the system, its overall energy consumption $E(x)$ and area costs $area(x)$. Regarding these, the DSE is, without loss of generality, formulated as a multi-objective minimization problem [30]:

$$\begin{aligned} &\textbf{minimize} f(x) = (lat(x), E(x), area(x)), \\ &\text{subject to:} \\ &\quad x \text{ is a feasible system implementation.} \end{aligned}$$

The latency of an implementation (including its schedule), is defined as the difference between the maximum end time $(\tau(t) + w(m))$, with the actual binding $(m = (t, p)) \in \beta$, and the minimum start time $(\tau(t))$ of all tasks:

$$lat(x) = \max_{(t,p)\in\beta} (\tau(t) + w((t, p))) - \min_{t\in T}(\tau(t)).$$

The energy consumption is the sum of the systems static and dynamic energy requirements:

$$E(x) = \mathcal{P} \cdot \sum_{d \in \alpha_D} P_{stat}(d) + \sum_{m \in \beta} E_{dyn}(m) + \sum_{r \in \gamma} E_r \cdot hops(r).$$

Finally, the area costs of the system are calculated as the addition of the area costs of each allocated hardware device:

$$area(x) = \sum_{d \in \alpha_D} area(d)$$

For further details on the evaluation steps for the objective functions, please refer to [18].

Due to the conflicting objectives, the previous formulated multi-objective minimization problem commonly does not have one single optimal solution, but a set of Pareto-optimal solutions X_P. These are identified by the use of the dominance relation $\succ$. It is defined for the n-dimensional quality vectors of two distinct solutions. A solution x dominates another solution y ($x \succ y$), if x is at least as good in every objective as y and if it outperforms y in at least one objective. Without loss of generality, for a minimization problem with n objectives, the dominance relation is formally defined as follows:

$$\begin{aligned} x \succ y \leftrightarrow \forall i \in \{1, \ldots, n\} &: f_i(x) \leq f_i(y) \\ \wedge \exists j \in \{1, \ldots, n\} &: f_j(x) < f_j(y) \end{aligned}$$

Further, a solution x is said to be Pareto-optimal, if there is no solution y which dominates x. Hence, by definition, Pareto-optimal solutions in the Pareto set X_P for a given problem are mutually non-dominated to each other: $\nexists x, y \in X_P : x \succ y \vee y \succ x$.

3.3 Adapted Exploration

Even though the number of Pareto-optimal solutions resulting from a DSE is limited, the size of the design space remains too large to be exhaustively explored in a feasible time. Hence, there is a need for efficient search strategies. As shown in [3], the utilization of domain knowledge offers great potential to improve the exploration. In the following, we present two approaches which allow to take influence on the solving process, namely strategies and heuristics. In Chapter 5, we will show how these approaches could be exploited in the context of PGE.

3.3.1 Strategies

To reduce the number of potential solutions, constraining functions, i.e. strategies, can be defined. They exclude certain design points from the feasible solution space and thus, reduce the search space. By exploring fewer options, the complete search time is lowered, which is a strong advantage of strategies. However, strategies can be too restrictive. That is problematic in the case that actually desired solutions, e.g.

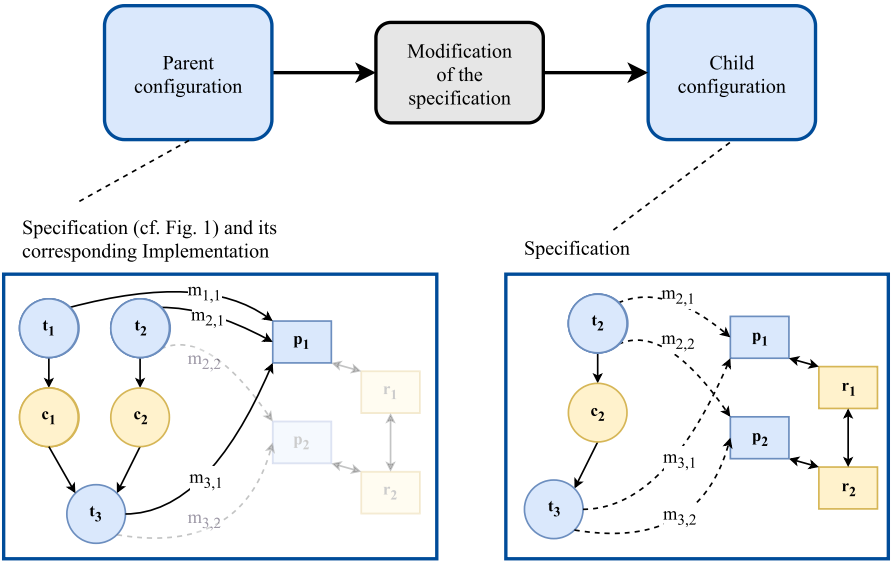

Fig. 2 Exemplary instances given to the development process of a new product version

optimal solutions, are excluded. So the influence of strategies needs to be thought out properly before being applied to a specific use case.

3.3.2 Heuristics

Heuristics take a weaker influence on the solving process than strategies. Compared, these do not exclude any solution from the feasible solution space. Instead, a heuristic steers the search into promising regions of the search space. Similarly, areas, where poor solutions are expected, are avoided. The idea is that domain knowledge can be exploited to find good or even optimal solutions in a short time. But because heuristics, as opposed to strategies, do not exclude solutions, they also do not necessarily reduce the time it takes to perform a complete DSE. This means, that the optimal solution is still guaranteed to be found, but only if the design space has been explored exhaustively.

4 Similarity of Design Points

This paper aims at exploiting the potential of evolutionary product design by applying knowledge from earlier product generations. For our approach, we consider a fully designed hardware/software system already released on the market as a reference for the development process of a successor version. In the following, this is referred to as the *parent configuration*. As shown in Fig. 2 it consists of a specification and one corresponding implementation. The previously introduced specification from Fig. 1 is further used as an exemplary instance for the parent

configuration. Together with its corresponding implementation it is shown in the blue box on the left side in Fig. 2. Parts, which are grayed out, are not used in the implementation of the parent configuration. The selected mapping options represent the bindings of the tasks given in the application graph. Since all tasks are bound to processor p_1, the routing is already indirectly decided and only processor p_1 had to be allocated. Note that the implementation of the parent configuration is not guaranteed to be an optimal solution, but a very good one concerning its application.

A new version, namely the *child configuration*, is derived from the parent configuration by modifying the given specification and then exploring for design possibilities according to the newly specified behavior and structure. In Fig. 2 one newly generated child specification is shown in the blue box on the right side. This specification serves as an example and shows one out of many possible modifications. It can be seen, that task t_1 and correspondingly communication message c_1 as well as mapping option $m_{1,1}$ have been removed, whereas the mapping options of the remaining tasks and the architecture are unchanged. In reality, this could be the case, for example, when a functionality becomes obsolete in a new product generation or when the set of functionalities is reduced in order to release a low-budget version of an existing product. In a next step, the decisions regarding the system synthesis need to be explored in order to determine an implementation for the child configuration.

To refer back to the concept of PGE, the idea is that both product generations contain the same core functionality and core architecture and only small changes to the specification are done. Nevertheless, depending on the kind and extent of the modification, such change can require a large as well as nearly no adjustment in the implementation of the final derived product.

For example, when only exchanging hardware components in a new system version due to improved technologies, potentially, every design decision, concerning the allocated hardware components and the task binding, can be kept exactly the same and only slight changes to the scheduling are necessary. A greater impact on the system can be expected, when, e.g., an additional feature is added to the product which necessitates special and so far unused hardware. To enable the new functionality, the allocation of an extra processing unit as well as additional interconnections to ensure the communication ability of the new processor might be required.

In our approach, first of all, the similarity of both specifications is evaluated. For that, the concept from [3] is used. Regarding every element of a specification, each equality and difference is identified. With respect to the parent configuration it is recorded, if a derived specification contains the same (*equal*) or newly *added* elements or if it is *missing* any, which means that those have been deleted. Exemplified on the basis of the element type *task*, the comparison of both specifications in Fig. 2 results in two *equal*, one *missing* and zero *added tasks* in the derived child specification. For all further element types in the specification, the evaluation is done accordingly to gain the similarity information among the two specifications.

Secondly, we consider the similarity of the implementations of both systems. In our prior work [3] we presented a first intuitive similarity analysis. To offer a complete and formalized definition of the similarity we base our comparison in this approach on the concept of the Hamming distance. In [31], the authors define a Hamming distance between assignments, where an assignment is a vector

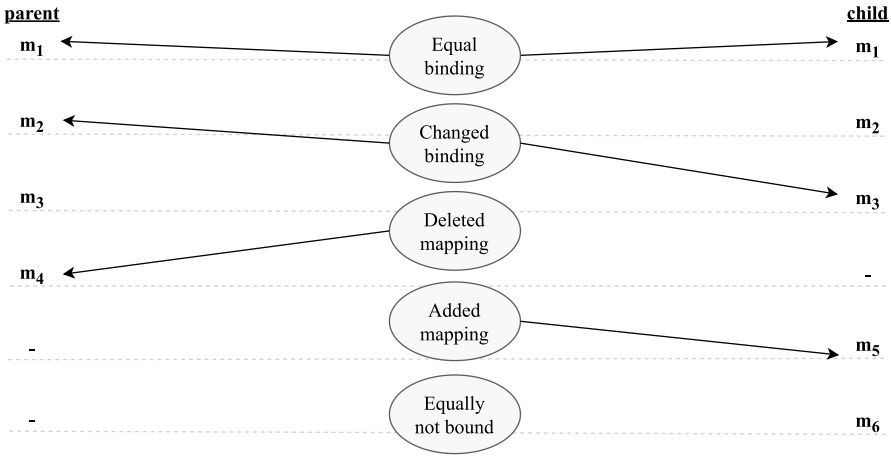

Fig. 3 Bit vector as a basis for the Hamming distance calculation

containing all variables to be decided on. Regarding two bit vectors of the same size, the Hamming distance then counts each entry, in which the two opposing bits are different.

To be able to apply this idea in the context of evolutionary product design, we analyze the implementations of both systems. In detail, each design decision, including the allocation, binding, routing and scheduling, is inspected. Therefore, the idea is to create a bit vector describing for all possible design decisions if a decision has been taken (1) or not (0). Taking the task binding as a representative case, the problem, if a task is equally bound on a specific processing unit can be translated to the question, if a mapping option, which connects the application and the architecture graph, is used equally. Thus, the bit vector $M_{parent} \cup M_{child}$, illustrated in Fig. 3, consists of all the mapping options of the parent and of the child configuration, which includes the common as well as only in one specification existing elements. Due to this and that the mapping options remain unchanged during the DSE, the bit vector has a constant size.

The black arrows in Fig. 3 point out which mapping options have been selected to be bound in each configuration. Thereby, five types of similarity or difference are defined. On the one hand, a binding decision in the parent configuration which bases on a mapping option, which exists in both configurations, can either be equally or unequally decided in the child configuration. These cases are illustrated by either m_1 or m_2 and m_3. On the other hand, there are mapping options which only appear in one of the specifications. For example, mapping option m_4 was deleted in the specification of the child system, even though it is used in a binding decision in the reference implementation. Therefore, it clearly causes a difference between the systems. Similarly, a binding based on a newly added mapping option (m_5) increases the Hamming distance. The last case is represented by m_6. This mapping option is only specified in the child configuration. But, since it is not used in the child

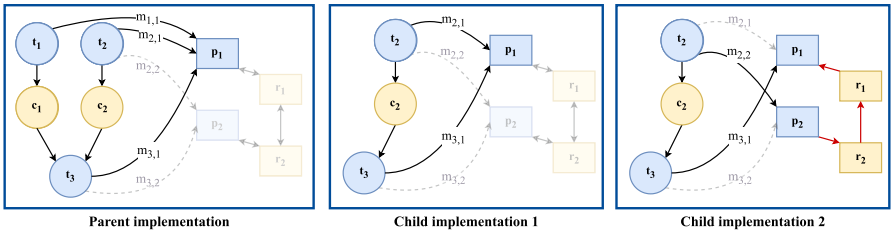

Fig. 4 Two exemplary child implementations to be compared to the parent implementation

implementation, the entry in both vectors is zero and therefore equal. Similarly, this applies to equally not used mapping options which are available in both specifications.

Moreover, regarding the Hamming distance definition in Fig. 3, the type of a *changed binding* causes two changes to the bit vector. But in reality, this could also be seen as one change, because only one task is executed on a different processing unit. Because of the analogue behavior of both definitions, in the following, we only use the metric of the normal Hamming distance, even though our encoding is based on an adapted version of the Hamming distance.

The theoretical analysis of the design decisions and the definition of a bit vector for the Hamming distance evaluation ensure that our similarity information is complete, visible and measurable. In comparison, our prior work [3] does not consider the case of *equally not taken* design decisions.

In total, the comparison of the two implementations results in two types of equality and three types of inequality. In difference to [3], we introduce this categorization in our encoding. Subsequently, each design decision is assigned either as `equal(TYPE,DECISION)` or as `unequal(TYPE,DECISION)`. This abstraction of the similarity information allows the general applicability in a variety of ASP methods and an uncomplicated usage even if the underlying definition of one of the specific types is adjusted.

To receive the Hamming distance, all differences are counted, i.e., $\forall m = (t, p) \in (M_{parent} \cup M_{child})$ a difference is counted if it is valid that either $(m \in \beta_{parent} \wedge m \notin \beta_{child})$ or $(m \in \beta_{child} \wedge m \notin \beta_{parent})$.

In Fig. 4 three exemplary implementations are presented. The parent implementation was previously in Fig. 2 introduced as well as the child specification, for which the given child implementations are two out of several possible solutions. With respect to the parent implementation, the Hamming distance for the binding decisions is calculated for each of the given implementations. The resulting entries in the bit vector, consisting of five entries, are shown in Table 1. Regarding the modification done to derive the child specification, implementation 1 is the closest possible assignment compared to the previous version. Since mapping option $m_{1,1}$ does not exist in the child specification, there is no option available to reach a Hamming distance of zero. Additionally, in the second child implementation t_2 is executed on a different processing unit. Due to this, one

Table 1 Hamming distance calculation for the binding decisions for the exemplary instances given in Fig. 4

	Parent implementation	Child implementation 1	Child implementation 2
$m_{1,1}$	1	0	0
$m_{2,1}$	1	1	0
$m_{2,2}$	0	0	1
$m_{3,1}$	1	1	1
$m_{3,2}$	0	0	0
Hamming distance	0	1	3

mapping option is newly used, while another newly stays unused. Therefore, the *changed binding* increases the Hamming distance by two and that shows, that implementation 2 is less similar to the parent system than implementation 1.

Further, the Hamming similarity can be calculated as follows. Its value ranges from zero to one, with equals one corresponding to 100% equality.

$$Hamming\ similarity = 1 - \frac{Hamming\ distance}{\mid M_{parent} \cup M_{child} \mid}$$

Accordingly, the Hamming distance and similarity for the synthesis step of the routing is calculated. The underlying bit vector $(C_{parent} \times E_{H,parent}) \cup (C_{child} \times E_{H,child})$ results from all combinations of the communication messages C and the linking edges in the hardware architecture E_H in both configurations. Regarding the routing decisions in Fig. 4, it can be seen that, even though communication message c_1 has been removed from the specification, in child implementation 1 no change to the routing path is caused, because in both, the parent and the first child implementation, the communication infrastructure is *equally not used*. In difference, in the second child implementation the establishment of a routing path is needed, because the communicating tasks t_2 and t_3 are executed on different processors.

For simplicity, the hardware allocation is not considered in the analysis of the implementations, but indirectly set by the binding and routing decisions. During the scheduling, start times are assigned to all tasks and communications. So, theoretically, a corresponding bit vector needs to consist of all tasks combined with every possible time slot for execution. In order to simplify it, we adapted the bit vector and based it on the execution order of the tasks.

5 Use of Similarity Information

The decisions taken in the design process of a product are of high value for the development process of new product generations and versions. The intention of the evolutionary product design is to change only minor aspects while the general functionality and architecture remains and it is expected, that small changes in the

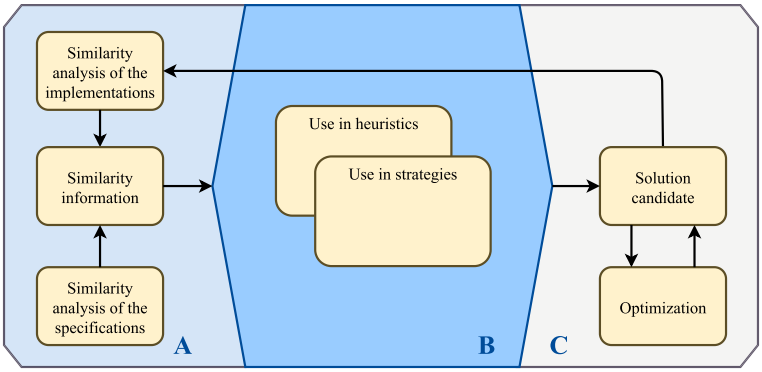

Fig. 5 Overview of the proposed approach (A+B) in connection to the state-of-the-art approach (C)

specification cause only small adaptions in the implementation as well. This means that good design solutions of the child configuration are similar to the previous product implementation, i.e. the implementation of the parent configuration.

In our approach, the DSE can easily be augmented with similarity information. In contrast to [3] t he modular structure of our novel encoding allows the applicability not only in one specific problem definition, but also in connection with various ASP methods. In this paper, we improve the DSE in the context of evolutionary product design by the use of strategies and heuristics.

In Fig. 5 the proposed approach is presented. First of all, in block C the state-of-the-art exploration process is shown [18]. Each solution candidate is evaluated according its validity concerning all requirements given in the specification and according its quality in context of the objective functions defined in the exploration model. In addition to that, the DSE is improved by the use of similarity information, as considered in block A. There are two sources of knowledge available. First, the specifications of both configurations are given and can be analyzed beforehand. Secondly, each solution candidate has implementation details which can be compared with the design decisions of the parent configuration. During the DSE, whenever a new solution candidate is evaluated, the similarity of the two implementations is analyzed. Finally, in block B in Fig. 5 the gained similarity information is used in different methods. These and their influence on the DSE are explained in the following sub Chapters 5.1 and 5.2. The complete encoding can be found in [32]. For a detailed description of our ASP encoding, including the system model, the system synthesis problem definition, the comparison of the specifications and implementations as well as the definition of objective functions, strategies and heuristics, an interested reader is referred to [33]. The state-of-the-art approach, shown in block C, serves as a comparison case for the evaluation of the approaches given in block B.

Further, another differentiation of similarity information has to be noted: Equality and Inequality. In the following, equal aspects are rewarded, while

unequal decisions are avoided. By strictly separating both concepts in the encoding, the size of the grounding is reduced.

5.1 Constraining the Search Space—Use in Strategies

Strategies are used to constrain the search space. In this approach, it is desired to exclude unequal design decisions from the design space. A first strategy, for example, can state that it cannot happen that *any* decision made to solve the system synthesis problem for the child configuration ever leads to *inequality* between the two product versions. Because of the general formulation of the similarity information a strategy can be applied to all synthesis steps and all types of inequality at once: `unequal(_,_)`. This strategy is quite efficient and removes a lot of unwanted implementations from the solution space. But there are configurations where the use of it is too restrictive. Consider the case that one processor which is used in the parent configuration is deleted in the structure description of a successor version. In that case it is forbidden to bound a task which was bound to the specific processing unit to any other. There is no way to find a satisfying solution to the system synthesis problem. It is the expected but not the desired behavior.

So we need to adapt our first strategy with extra information which we can obtain from the comparison of the two specifications. To refer to the previous example, the strategy can be modified in such a way, that it now only bans the case that a task which exists in both configurations is not bound equally even though the original execution target is given in both specifications.

In order to find valid solutions regarding the routing, it needs to be examined not only if the previously allocated routers are existing in the child specification, but also if the sending and the receiving task are bound equally. This includes that the respective processors are still existing in the predecessor system. Only in that case, an identical routing path can be forced. Otherwise, the starting location or destination will inevitably be different. Therefore, our first strategy must be adapted with this information.

When creating an adapted strategy considering the scheduling decisions, in this approach, we only force an equal execution order of pairs of tasks in the case that the respective tasks are contained in *both* configurations.

In the presented adapted strategies, it is not allowed to make *unequal* decisions if possible. Similarly, the equality information can be handled. It is stated, that it cannot happen that any decision is made *not equally*, i.e. every decision leading to a child implementation has to be made exactly the same like it has been done in order to receive the parent implementation. Likewise, the adaptions for the steps of the binding, routing and scheduling are encoded.

5.2 Directing the Search—Use in Heuristics

In this sub chapter, we present another method to use similarity information in a less restrictive way. The usage of domain-specific heuristics allows to steer the search towards promising regions of the search space where favorable solutions are expected without excluding any design options.

For example, the first strategy presented in the previous Chapter 5.1, was too restrictive. But the stated formulation can be left unchanged (without any adaptions) in the utilization with heuristics. In that case, design decisions leading to similarities between the systems are treated preferentially or their truth value is set to true. Respectively, decisions leading to differences between the two implementations are initially avoided or their truth value can be directly set to false. But if a certain preferred decision leads to unsatisfiability, it still can be discarded and decided differently in the course of the DSE.

The solver clingo, which we utilized in this approach, allows an user to modify its internal heuristic by enabling domain-specific heuristics [25]. Due to our improved formulation, the similarity information can easily be used in domain-specific heuristics, so that the DSE is easy to augment with prior knowledge. Again, no differentiation of the system synthesis steps and of the specific inequality type is required, whereas in our prior work [3] every inequality case per each synthesis step is taken up inside at least one own heuristic. When considering the *not equally* decisions made, we still need to consider each synthesis step in our new approach, but the implementation is independent from the specific equality types.

The encoding of a heuristic directive is described in detail in [25]. In the following, we will only give a brief introduction to two elements, namely the heuristic modifier and the modifier's value, which can be seen as a key value pair *[value, modifier]*.

By their usage, on the one hand, the decisions to be taken during the DSE can be biased. For example, the key value pair *[2, factor]* causes that the score of a certain decision is doubled, so that this decision becomes more and more important in the course of the DSE. A stricter modification, on the other hand, could be to control the truth value of a certain decision variable. In order to find a solution close to the implementation of the parent configuration, a specific decision can be intended or not. Consequently, when the solver is about to make a specific decision, the corresponding truth value is set to true (*[1, sign]*) or to false (*[-1, sign]*), depending on the information gained from the similarity evaluation. Further, a decision can be made preferentially over all others by using the key value pair (*[1, level]*).

The previous presented modifiers can be combined as *[1, true] = [1,sign] + [1,level]* and *[1, false] = [-1,sign] + [1,level]* which indicates that the decision is not only preferred, but also set to *true* or *false*, respectively to the outcome of the similarity evaluation.

In this paper, similarity information will be used in domain-specific heuristics and the influence of different heuristic modifiers on the search process is investigated.

6 Experiments

This project is implemented as ASP and Python code. For the evaluation of the experiments we further used Python, C++ and Bash scripts. The tool clingo is used in version 5.5 [34], the extension clingo-dl in version 1.3 [35] and it is at least Python version 3.9 required. For setting time stamps during the project execution the tool runlim in version 2.0.0rc3 is used.[1] The subsequently introduced experiments run on two hardware platforms. Parts of the experiments are tested on a platform basing on a Intel Core Xeon E3-1260Lv5 CPU and 32 GiB RAM (considered as machine 1), while others are executed on a platform containing a Intel Core Xeon E5-2650v4 CPU and 64GiB RAM (considered as machine 2). The surrounding environment in each case is Linux Debian 10.

6.1 Experimental Setup

As a setup, our benchmark set consists of 35 parent instances which are generated by an ASP-based benchmark generator [36]. Each of those has the same underlying hardware platform, namely a $3 \times 3 \times 1$ grid structure, consisting of nine routers bidirectionally connected to each other and additionally each to one processor and back. This hardware platform represents a heterogeneous multi-processor system. Additionally, middle-sized and large-sized instances have one or respectively two more layers in z-direction, resulting in a $3 \times 3 \times 2$ or a $3 \times 3 \times 3$ grid structure. The applications of the instances consist of one up to four application graphs, which are generated as "series parallel graphs" (SPG). In total, the complexity of the instances differs in the range of 6 to 160 tasks. The use of synthetically generated test instances allows sufficient scalability, requiring only a few setup parameters. And we see a lot of applications in the field of streaming applications and digital signal processing.

We consider the parent configuration to be a released product on the market. To assume an implementation with good, but not necessarily optimal characteristics as a basis for the experiments, three DSEs (each one using a different routing encoding) for each specification are executed for 12 h on machine 1. Randomly, one design point is picked from the best solutions of all three runs and considered as the implementation of the parent configuration.

From each parent specification, one modified child specification is generated. The modification is composed of a randomly decided combination of different changes including the deletion, addition or exchange of components and is applied to 20% of the task elements and thereby also to the corresponding communication messages as well as to 20% of the processing units. In case of deletion of a processor, the communication structure remains unchanged. An added processing device is interconnected with the communication structure via a new router.

In our prior work, we analyzed the influence of different percentages and kind of modifications in the DSE utilizing one specific domain-specific heuristic [3]. In contrast, the aim of this work is to investigate the influence of different ASP

[1] The tool runlim is available at https://github.com/arminbiere/runlim.

Table 2 Overview of the applied methods and the adjustment points

Case	Inequality information	Equality information
Strategies	s3	s4
Heuristics	h1-factor-2	h2-factor-2
	h1-factor-4	h2-factor-4
	h1-factor-8	h2-factor-8
	h1-sign	h2-sign
	h1-true	h2-false
Adjustment point	Value	
Routing encoding	xyz	
	bound	
	arb	
Synthesis step	B	
	B & R	
	B & R & S	

methods on the search process. In Table 2, in the upper part, a strategy and five heuristics each for the use of equality and inequality information are given. Note that in our approach, the heuristic modifier *factor* is always used in combination with the modifier *sign* to increase the importance of the desired design decisions. Further adjustment points are the different synthesis steps, including the binding (B), the routing (R) and the scheduling (S), and the routing encodings, which are introduced in Chapter 3.1. These points are summarized in the lower part of Table 2. For comparison, we also run one DSE from scratch for each routing encoding.

In the present categories, the experiments from [3] are to be classified as dimensional ordered routing (xyz), using exactly one heuristic which weights each of the synthesis steps (B & R & S) differently utilizing the heuristic modifier *true*.

In total, we examine $(3 * 12 + 1) * 3 = 111$ cases and conduce $111 * 35 = 3885$ DSE runs on machine 2, each for up to 900 s. Instead of exhaustively exploring all design solutions in order to find the one optimal design, we aim at improving the search process to the able to find good solutions early. We simulate this by a relatively short time-out. It also allows us to test a wide range of methods applied to instances of differing complexity.

The quality of the resulting implementations are evaluated concerning their ϵ -dominance [37]. Therefore, for each instance a reference front is generated, which consists of the best solutions found up to the timeout during the 111 DSEs applying all cases. This reference front is considered as the optimal solution front for a certain instance. Further, all resulting child implementations are compared to the respective parent solution and evaluated concerning their Hamming similarity as previously defined in Chapter 4. In detail, every intermediate solution is assigned a time stamp during the DSE and evaluated with respect to both metrics. In this context, in each time step, we especially consider the design points which are not dominated and therefore, are forming the best solution front found so far. Moreover, we look at the

states of the DSE runs after the timeout and the times to find a first solution or all feasible solutions.

6.2 Experimental Results

To get an overview of the results from the test cases, we summarize the status information as well as the quality metrics regarding the applied methods and the complexity of the instances.

Firstly in Table 3, the DSE runs are classified according to their outcome, i.e. whether the system synthesis problem was satisfiable (I and II) or unsatisfiable (V), the latter being the worst case. In case of satisfiability, we further examine the times it took to find a first solution and if it was possible to exhaustively explore the solution space in the given time (if yes I, else II). The event of a timeout can occur at any stage, even in the grounding phase (IV) or during the solving process without having found any solutions yet (III). The cases III and IV cause that the result of the system synthesis problem is unknown. In Table 3 the percentage of DSE runs resulting in each category regarding each routing encoding are presented. The categories I to V are each depicted with an own color and are more desirable, the smaller the number of the category. An interested reader is referred to the extended Table 7 in the appendix which contains the actual percentage numbers.

All cases, which combined knowledge from previous scheduling decisions with strategies turned out to be unsatifiable. Although we have used adapted strategies, the decisions on the scheduling are still too small-stepped. The start time of the execution of each element depends on every other scheduling decision, so that they can not be forced without being too restrictive. Moreover, a differently decided task order will have the least influence on modifications in the development process of a successor product of all synthesis steps, i.e. it is the least important synthesis step in our experiments. Due to the given reasons, we exclude cases regarding the use of previous scheduling decisions from the further course of the evaluation as well as from Table 3 and focus on the strategies and heuristics applying the knowledge of previous binding decisions (*B*) or binding and routing decisions (B & R).

In Table 3 the increasing complexity of different routing encodings can be seen. Due to the lower number of design solutions in the dimensional order routing, more DSEs have been finished before the timeout (I) or are at least satisfiable (II) than for the routing which considers a bounded number of hops.

The reduced complexity is also seen in the lower values in category IV, because a timeout during the grounding phase indicates a high overhead of that test case, either caused by the complexity of the routing method or introduced by the applied similarity evaluation and usage. These trends are the same for the comparison of the results for *encoding_arb* and for *encoding_bound*.

A similar relation exists for the increasing complexity of the instances and the categorization of the test cases. The bigger an instance is, the less methods are able to explore the search space completely or even find any solution at all.

A detailed evaluation of the results for the use of strategies and heuristics is given in the following sub chapters.

Table 3 Distribution of the DSE states after timeout

Cases		Encoding_arb
All		
Strategies	All	
	B	
	B & R	
Heuristics	All	
	B	
	B & R	
From scratch		

Cases		Encoding_bound
All		
Strategies	All	
	B	
	B & R	
Heuristics	All	
	B	
	B & R	
From scratch		

Cases		Encoding_xyz
All		
Strategies	All	
	B	
	B & R	
Heuristics	All	
	B	
	B & R	
From scratch		

The data was taken from mdfiles/statusSummarized.md and mdfiles/statusV2Summarized.md from the experimental results within [32]

Satisfiable, no timeout (I):

Satisfiable, timeout (II):

Timeout during solving (III):

Timeout during grounding (IV):

Unsatisfiable (V):

6.2.1 Strategies

The cases in Table 3 applying strategies mainly result in category I, II or V, i.e. the use of strategies during the DSE works out pretty good or very bad. First of all, a big difference in the performance of strategies can be noticed, when applying them only to the binding decisions or to both, the binding and routing decisions. In the context of the routing, there are many more decisions to be made than for the binding. It can be seen that the more decisions we try to force, the more restricted the system is. This applies even more clearly to the previous excluded approach of applying the strategies to all synthesis decisions at once, including the scheduling. Later on, we will analyze further reasons for unsatisfiability.

The partly good values in category I in Table 3 indicate that some of the strategies are able to execute the complete DSE for more instances than the other cases. In general, the benchmark set contains six instances, which are small enough that they can be completely explored by several adapted DSEs. Those contain 6 up to 21 tasks. Besides the smallest instance, which is exhaustively explored in all cases considered in the evaluation (in total 75), 21 to 46 cases are able to do the same for the other five small instances. In comparison, the best strategy is able to completely solve the synthesis problem for 10 out of 35 instances before the timeout. These actually do not contain two of the small instances, but instead some middle sized instances with up to 73 tasks.

This is possible, because the use of strategies excludes many design options from the feasible solution space. Therefore, the amount of possible solutions is decreased massively. Further, the percentages in category III and IV in Table 3 underpin this. Small numbers in this category mean that the search process has been simplified in means of the number of valid decision variables assignments. Dealing with a lower problem overhead, the proof of satisfiability or unsatisfiability can be provided more quickly, i.e., before the timeout.

The reduction of the amount of feasible solutions is further noticeable when comparing the search times.[2] When looking at all 35 instances for the cases, which have found a first solution the earliest, the two strategies forbidding unequal (*s3*) and not equal (*s4*) binding decisions are 15× and 13× under the best three cases. That means, that these strategies win nearly up to half of the amount of all instances in this category. Besides, it can be noted that only cases using the *encoding_xyz* are within the best three cases for all instances.

The picture becomes even clearer, when analyzing the complete search time. The two strategies named before, which forbid unequal (*s3*) and not equal (*s4*) binding decisions, not only solve the synthesis problem completely for most of the instances, but they also solve all of these instances as the fastest. Further, five out of twelve strategies are at least once within the best three cases for the complete solving time, evaluated for each instance.

[2] The data was taken from mdfiles/topSummarized.md (Table “First solution time” and Table “Complete search time”) from the experimental results within [32].

Table 4 Comparison of both types of strategies

Cases	s3	s4
Encoding_arb		
Encoding_bound		
Encoding_xyz		

The very good searching times cannot hide the big amount of failed DSEs given in Table 3. Even though, we used already adapted strategies, these still seem to be too restrictive.

In Table 4 the number of satisfiable (I and II) and unsatisfiable (V) cases for both types of strategies, distinguished by which kind of similarity information is used, is compared. Furthermore, the table is subdivided according to the routing encodings.

The actual percentages numbers can be found in the extended Table 8 in the appendix.

It clearly can be seen, that the DSEs adapted with strategy *s4* in all cases behave worse than adapted with strategy *s3*. For example, when applying *s3* one forth to one half of the DSE runs are unsatisfiable while for the use of *s4* up to three forth of the explorations are unsuccessful.

When considering two different systems, it can be assumed, that they differ in a few design decisions. But, at the same time, they have a lot more aspects in common, especially, because every decision which does *equally not exist* in both systems is registered as a similarity. Since the strategy *s4* considers the equality information, it takes influence on many more design decisions than the strategy *s3* and thus, can cause more restrictions. As we have previously seen in Table 3, the more constrained the DSE is by restrictive strategies, the more unsatisfiability cases are caused.

Further, when comparing the results for the different routing encodings, there is a small increase of the percentage in category V, the more restrictive the routing encoding gets. This can additionally be explained when considering that the parent implementation was randomly chosen. That way, e.g., the parent implementation can contain routing paths which are reachable by the use of the *encoding_arb*, but not for the *encoding_bound* and even less for the *encoding_xyz*. Then, these routing paths cannot be forced to exist in the child implementation without any rule violation. Another reason for unsatisfiability can be that, when creating the child instances, we did only modify the task graph and the hardware architecture, but not the period. Therefore, there is no guarantee, that additions to the task graph or modifications on the hardware platform do not increase the execution time of the application. This way, the execution time of the new application may exceed specified allowed period.

6.2.2 Heuristics

In this chapter, we analyze the search states after timeout as well as the worthiness of the discovered solutions during the heuristic DSEs.

The goal is to steer the search towards areas of the design space potentially containing solutions with optimal properties. These properties have been defined in our exploration model in Chapter 3.2 and are considered in a multi-objective optimization. Each DSE results in a front of Pareto-optimal design points. Considering, all DSE runs (applying each case) a reference front, containing all non-dominated design points, per instance is identified. The *quality* of each single solution front is evaluated utilizing the ϵ-dominance [37] which is a metric evaluating the convergence between a solution front and a non-dominated reference front, where the better the convergence, the better the quality.

As a second metric, we will evaluate the Hamming similarity of the found solutions with regard to the parent configuration (as presented in Chapter 5). Knowing an implementation of good quality, we expect a similar implementation of a derived product version to be of good quality too.

Evaluation of DSE states

For the evaluation of the heuristics, we also analyze the results shown in Table 3. The first, most obvious aspect is, that there are no heuristic DSEs which are unsatisfiable. These results conform our expectations that heuristics take influence on the search strategy of the solver without excluding any feasible solution of the given system synthesis problem.

Neglecting rounding deviations, the values in category IV for the heuristic DSE and the DSE from scratch are the same. This means, that the introduction of atoms containing similarity information, did not add a noticeable size offset to the grounding of the problem instance. As a drawback, category IV indicates that the chosen instances are too large to be handled up to the short time-out. Depending on the routing encoding this applies to five instances with four applications containing 107 up to 160 tasks. Interestingly, all instances with a lower number of applications, containing up to 125 tasks, could be handled.

When considering the remaining categories, it can be seen that the heuristic DSE behave in average as good (in I) or better (in II) than the DSE from scratch. But when applying different heuristics, the results differ considerably. For example, between heuristics emphasizing equal binding decisions and those pushing equal binding and routing decisions, there is a difference nearly up to seven percent. Though there is no clear favorite, it is noticeable that influencing the routing decisions shows a better effect on the DSE outcomes in combination with the arbitrary routing encoding because a routing path can arbitrarily, but according to the prioritization of the applied heuristic, be chosen. In the following, we will take a closer look at the heuristics being beneficial or detrimental to the synthesis problem of a hardware/software system.[3]

[3] The data was taken from mdfiles/statusSummarizedV2.md from the experimental results within [32].

Table 5 Comparison of the best and worst performing heuristic DSEs[4]

		arb		bound		xyz	
		I (%)	II (%)	I (%)	II (%)	I (%)	II (%)
Best case	true / false	2.86	49.29	9.29	76.43	17.14	79.29
	br-h1-true	2.86	60.00	8.57	77.14	17.14	80.00
	br-h2-false	2.86	71.43	5.71	80.00	17.14	77.14
Worst case	h1/h2 sign	2.86	35.00	10.71	62.86	17.14	76.43
	b-h1-sign	2.86	31.43	11.43	65.71	17.14	71.43
	br-h2-sign	2.86	20.00	8.57	57.14	17.14	80.00
All heuristics		2.86	43.71	10.14	68.57	17.14	78.71
From scratch		2.86	40.00	11.43	65.71	17.14	77.14

First of all, for the comparison in Table 5 we pick two sample heuristics for each, the best and the worst outcome of the DSE. Since, in both cases each a specific heuristic modifier was used, we also give the averaged status information for all four cases which applied the respective modifier. The values are contrasted for the three routing encodings and compared with the DSE from scratch and the behavior of all heuristics in average.

It can be seen a remarkable difference, especially, in the context of the arbitrary routing, i.e. when the search is the least restricted and the heuristic is allowed to show its full strength. Simultaneously, with the increased complexity the heuristics have their greatest impact on the search because the short timeout does not offer sufficient exploration time. Consequently, the outcome depends on the solutions found first. Comparing the amount of cases solved by the DSE from scratch to the specific heuristic DSEs a deviation from −20% up to +31.43% can be seen. It shows clearly, that when applying different heuristics the solver is steered towards varying regions in the design space and favors different solutions.

Evaluation of the resulting single design points

First of all, the best three DSE runs in the categories *First solution, Complete search time*, *ϵ-dominance* and *Hamming similarity* per instance are identified and the occurrences of the single cases are counted over the whole set of 35 instances.[4] Considering the categories that take into account the times, there is no clear difference between the heuristic DSE and the DSE from scratch nor in between the different heuristics. Besides the applied strategies (see Chapter 6.2.1), there are no clear winners. In contrast, the results, when searching for the maximum Hamming similarity of all design points,[5] are clearly dominated by the heuristic DSEs and constrained DSEs. 12 out 12 strategies appear at least once in the list. That is reasonable, because they only allow very similar solutions. 49 out of 60 heuristic DSE reach

[4] The data was taken from mdfiles/top.md and mdfiles/topSummarized.md from the experimental results within [32]

[5] Regarding the Table “Hamming total” in the same file.

at least once a top ranking, whereas a few clear winners are the cases *br-h1-true* and *br-h2-false* for all routing encodings as well as *br-h1-factor-2*, *br-h1-factor-4* and *br-h1-factor-8* in combination with the hop-bound routing encoding. In comparison, on single DSE run from scratch reached a good ranking in this category.

Note, that in the total similarity value considering all design decisions, the binding decisions are overlaid by routing decisions, because of the remarkable size difference of the respective bit vector. For example, on a platform structured as a 3× 3×1 grid one task can potentially be bound on nine processors, whereas for one communication the combination with 21 connecting hardware links has to be checked. Respectively, the disproportion between both values is increasing, the more tasks and communications are considered. Also, a larger platform strengthens the influence. Especially, because of the big amount of *equally not used* bit vector entries, the Hamming similarity for the routing decisions tends to have higher values than the one for the binding decisions.

When regarding the Hamming similarity separately for the binding and the routing decisions, small differences can be noticed. Considering the Hamming similarity only for the binding decisions,[6] some further heuristics, which only take influence on the binding options, additionally reach the top three rankings. Again, the same modifier true/false (in the cases *b-h1-true* and *b-h2-false*) performs exceptionally well. Also, a few more, but significant many DSE runs from scratch appear in the list.

In real-world problems we usually face large and complex instances, whose design options can not be exhaustively explored in a reasonable time. Thus, the idea of the heuristic DSE is to steer the search into promising regions of the search space to find good quality solutions early. Therefore, another important aspect to investigate is the similarity and quality of the *first solution* found.[7]

Here, we will evaluate the top three cases for the Hamming similarity to the first solution. When counting the top three occurrences over the set of instances, we receive values distributed over 26 heuristics, 8 strategies and no DSE runs from scratch. According to the results is prior knowledge beneficial to find solutions early that are closer to the implementation of the parent configuration. The evaluation of the ϵ-dominance of the first solution again delivers a quite distributed picture. 45 (of 60) heuristics, 5 (of 12) strategies, 2 (of 3) DSE from scratch listed, with each not more than 7 occurrences out of 35 instances don't provide a clear winner.

Simultaneously to the top three evaluation, the three worst performing DSE runs for each instance in different categories are investigated.[8] A similar picture is emerging as for the top three evaluation. In both categories, *the Hamming similarity of the first solution* and *the Hamming similarity over all solutions* the negative winners are the DSE

[6] Regarding the Table "Hamming binding" from mdfiles/top.md and mdfiles/topSummarized.md from the experimental results within [32]

[7] Regarding the tables "Hamming total, only first solution" and "Epsilon dominance, only first solution" from mdfiles/topSummarized.md within [32]

[8] Regarding the tables "Hamming total", "Hamming total, only first solution", "Epsilon dominance" and "Epsilon dominance, only first solution" taken from mdfiles/flop.md and mdfiles/flopSummarized.md (Table) within [32]

runs executed without prior knowledge. This applies to all routing encodings, with each having at least 10× (out of 35 instances) the lowest Hamming similarity in the first solution, especially, the dimensional order routing has low similarity values (counted 14× out of 35 instances). Further, the heuristic modifier *sign* is appearing several times in this context. Especially, the case *b-h2-sign* in combination with the dimensional order routing is counted at least 10× in both categories as well.

Considering the ϵ-dominance, again, no clear negative winner is identifiable. Only, the arbitrary routing is slightly more often present in the ranking due to the increased complexity of the search space.

Evaluation of resulting Pareto optimal solution front

So far, we have only looked at single design points. We have seen, that the heuristic DSE find solutions similar to the parent implementation in total and also early. In the following, we consider the ϵ-dominance and the similarity of the non-dominated solution front per DSE to investigate whether there is a relation between good and similar solutions. For the evaluation of the Hamming similarity we take all design points which are part of the best front and determine the maximum and average Hamming similarity regarding those.

All DSE runs (each DSE corresponds to one case) are ranked in the categories *Maximum Hamming similarity for all design points*, *Maximum Hamming similarity for the best solution front*, *Average Hamming similarity for the best solution front* and *ϵ-dominance* according to their outcome. If one DSE didn't find any results it is punished with the worst rank for this instance. Further, in these categories, for each case, the average rank over the set of instances and the corresponding standard deviation is evaluated.[9] The new results are ordered again, according to their average ranks. An excerpt of the results is presented in Table 6.

In Table 6, in the upper part, the values for the maximum Hamming similarity regarding all design points are given, while in the lower part the average Hamming similarity for the non-dominated solution fronts is considered. For selected cases, the average rank and the respective standard deviation along with the respective *position P* in the whole ranking is presented. Further, the table is subdivided into several groups, beginning with the best five cases and ending with the DSE with the worst result. Since, the first group is completely dominated by the hop-bound encoding, in the second group we name the best cases for each further routing encoding. The hop-bound routing encoding seems to offer a good compromise between complexity and restrictiveness.

For comparison, the second last group presents the results of the DSEs from scratch. It is noticeable that the cases which don't utilize any prior knowledge have quite high average rank values. Especially in the table on the bottom side, each of the cases of DSE from scratch is among the 15-worst performing cases. Since, the DSE from scratch selects solutions in means of the similarity of two systems

[9] The following data was taken from mdfiles/ranks.md and mdfiles/ranksAveragedOrdered.md (Table "Hamming total", Table "Maximum Hamming total", Table "Average Hamming total" and Table "Epsilon dominance") from the experimental results within [32]

randomly, the standard deviation is higher in these cases. Also, the results for the poorly performing heuristic DSE deviate more.

It has to be noted, that the standard deviation given in Table 6, in general, has high values, i.e., the cases perform different when applied to varying instances. Further, we track and plot the Hamming similarity and the ϵ-dominance over time for each solution in each DSE. But due to varying behavior, it is not possible to select exemplary cases representing all heuristic modifiers and strategies. An analysis of the combination of each ASP method with the several adjusting points and each instance, is beyond the scope of this paper. However, an interested reader can find those results within [32].

When considering the DSE from scratch in the context of the respective standard deviation, the average ranks, especially in the category of the average Hamming similarity for the solution front, are still far from a good positioning.

The previous consideration of the non-dominated solution front per DSE run is particularly important, because it contains the information that high-quality solutions are of good similarity as well, i.e., the utilization of partially equal implementation details can lead to high-quality solutions. For example, in 14 out of 35 instances the design point with the highest similarity to the parent implementation, is part of the best solution front.

As deeper analysis, we want to evaluate the average ranking for each case over all instances in the category of the ϵ-dominance. On the first view, the heuristic DSE and DSE from scratch perform equally. Only the xyz-encoding is significantly better than the other routing encodings for all the cases. But when connecting the information on the average ϵ-dominance with the content from Table 6, we receive a more clear picture. For this purpose, in Table 6 we take a look at the third group of cases, which have top ten up to average positioning. The selected cases present the best five DSE runs according to their average ranking in the category of the ϵ-dominance. In brackets behind the case name, the respective position is given. It can be seen, that in most of the cases these are among the best fifty percent of the cases, and have better positions than the DSE from scratch. Further, we notice that in all categories one heuristic modifier is dominating the best positions, namely true/false.

The detailed examination of the experimental results has shown, that the utilization of heuristics does not only steer the search towards solutions close to the design of the parent system, but also does find high-quality solutions which are more similar to a previous product implementation than the solutions found in the DSE from scratch. Given a set of non-dominated solutions, a product designer can choose a design point according to the company's preferences and quality demands. Since, both, the DSE with and without use of prior knowledge, can discover solutions of good quality in a reasonable amount of time, the utilization of both approaches is equally valid. But in comparison, the solution's similarity to a prior product version is an advantage, because it allows the reuse of common implementation details and therefore, to save time in the product development process and to create a less error-prone system.

Table 6 An excerpt from the averaged rankings in the categories "Maximum Hamming similarity for design points" (upper part) and "Average Hamming similarity for solution fronts" (lower part)

P	Ranking + Deviation		Case
1	4.80	6.63	bound-br-h1-true
2	7.94	10.04	bound-br-h1-factor-8
3	8.26	9.84	bound-br-h1-factor-4
4	8.49	7.29	bound-b-h2-false
5	9.23	7.70	bound-b-h1-true
8	12.77	9.61	xyz-b-h1-true
14	15.14	11.58	arb-br-h2-false
8	12.77	9.61	xyz-b-h1-true (2)
9	13.09	9.36	xyz-b-h2-false (4)
12	13.63	9.56	xyz-br-h1-true (5)
29	18.86	11.61	xyz-br-h2-false (1)
41	23.80	12.96	xyz-br-h2-factor8 (3)
45	25.37	16.90	bound
56	34.26	14.80	xyz
64	36.63	16.55	arb
75	42.11	16.03	arb-b-h2-sign
P	Ranking + Deviation		Case
1	8.60	9.03	bound-br-h1-true
2	13.06	13.01	bound-br-h1-factor-8
3	13.29	12.26	bound-br-h1-factor-4
4	14.83	10.17	bound-br-h2-false
5	15.06	12.64	bound-br-h1-factor-2
8	17.63	14.00	arb-br-h2-false
15	21.98	18.25	xyz-b-s3
16	22.20	15.10	xyz-br-h1-true (5)
18	22.57	13.37	xyz-b-h2-false (4)
20	23.06	15.01	xyz-b-h1-true (2)
36	30.11	16.83	xyz-br-h2-false (1)
43	34.17	16.00	xyz-br-h2-factor8 (3)
62	42.06	17.98	xyz
65	42.89	16.64	bound
72	48.94	17.98	arb
75	48.97	19.26	xyz-br-s4

7 Conclusion

In this paper, a systematic approach to enhance the product design process in the context of *product generation engineering* is presented. Taking up the idea of the Hamming distance, we propose a novel similarity metric on basis of the specification and implementation details of a parent configuration and a derived child configuration. The gained knowledge is generally usable in different ASP methods, like strategies in order to constrain the search space as well as heuristics to steer the search into regions of the design space potentially containing high-quality design solutions of the next product generation.

In order to ensure a meaningful evaluation, the proposed similarity metric was utilized in an extensive number of ASP methods and their impact investigated by varying the underlying routing encoding and considering different synthesis steps.

In the experiments, it could be seen that the usage of strategies can massively reduce the number of feasible design points. Thus, the search space can be exhaustively explored in a significantly short time. However, in many cases the search space is too restricted, i.e. any feasible solution is excluded. Therefore, we investigated another less restrictive methodology how to augment the DSE with knowledge gained from prior product versions, namely domain-specific heuristics.

The results of the heuristic DSE runs were as good or better than the DSEs from scratch. But in difference, heuristics can steer the search towards solutions which are closer to the parent system than randomly chosen solutions. Only providing short searching times, this is especially important when exploring complex problem instances, since the first solutions found already deliver the desired outcome. As a drawback, results have shown that the performance of different heuristic can massively vary. Thus, we can conclude that the use of knowledge on prior product decisions can enhance the system-level DSE, but the selection of the appropriate ASP methods depends on the use case and has to be made wisely.

Future work needs to investigate the combination of the presented methods. Depending on the type of modification in the specification of a product, the use of heuristics or strategies should be preferred. For example, the usage of strategies can be valuable in case of the exchange of a specific component with a better performing element, because there as few as possible changes are demanded. Whereas, heuristics may be utilized when product elements have been deleted or newly added. Further, the proposed similarity metric might be improved by taking into account the range of the influence of a change. It can be investigated which modification causes local or global changes.

Appendix A: Extended Data

See Tables 7 and 8.

Table 7 Distribution of the DSE states after timeout

Cases		Encoding_arb				
		I (%)	II (%)	III (%)	IV (%)	V (%)
All		3.20	39.43	37.37	13.14	6.86
Strategies	All	5.00	18.57	26.43	7.14	42.86
	B	2.86	32.86	44.29	7.14	12.86
	B & R	7.14	4.29	8.57	7.14	72.86
Heuristics	All	2.86	43.71	39.29	14.29	0.00
	B	2.86	40.29	42.57	14.29	0.00
	B & R	2.86	47.14	36.00	14.29	0.00
From scratch		2.86	40.00	42.86	14,20	0.00
Cases		Encoding_bound				
		I (%)	II (%)	III (%)	IV (%)	V (%)
All		10.4	61.14	8.46	10.63	9.37
Strategies	All	11.43	22.86	0.71	6.43	58.57
	B	14.29	40.00	0.00	7.14	38.57
	B & R	8.57	5.71	1.43	5.71	78.57
Heuristics	All	10.14	68.57	9.86	11.43	0.00
	B	11.43	70.00	7.14	11.43	0.00
	B & R	8.86	67.14	12.57	11.43	0.00
From scratch		11.43	65.71	11.43	11.43	0.00
Cases		Encoding_xyz				
		I (%)	II (%)	III (%)	IV (%)	V (%)
All		16.8	69.49	1.14	2.63	9.94
Strategies	All	15.00	21.43	0.00	1.43	62.12
	B	27.14	32.86	0.00	1.43	38.57
	B & R	2.86	10.00	0.00	1.43	85.71
Heuristics	All	17.14	78.71	1.29	2.86	0.00
	B	17.14	78.29	1.71	2.86	0.00
	B & R	17.14	79.14	0.86	2.86	0.00
From scratch		17.14	77.14	2.86	2.86	0.00

Table 8 Comparison of both types of strategies

Cases	s3			s4		
	I (%)	II (%)	V (%)	I (%)	II (%)	V (%)
Encoding_arb	7.14	21.43	25.71	2.86	15.71	60.00
Encoding_bound	14.29	28.57	42.86	8.57	17.14	74.29
Encoding_xyz	14.29	32.86	50.00	15.71	10.00	74.29

Acknowledgements This work was funded by the German Science Foundation (DFG) under grants HA 4463/4-2 and SCHA 550/11-1.

Author Contributions L.M. and P.W. have developed ideas for this approach. All authors have discussed the ideas. L.M. has written the complete manuscript and prepared all pictures as well as all tables. All authors have reviewed it.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose. All of the material, except for Fig. 1 is owned by the authors and no permissions are required. For Fig. 1 there is a reuse agreement with Springer Nature (license number 5480211390158).

References

1. Albers, A., Bursac, N., Wintergerst, E.: Product generation development-importance and challenges from a design research perspective, 16–21 (2015)
2. Dannenberg, J., Burgard, J., Oliver, W..C.: 2015 Car Innovation: A Comprehensive Study on Innovation in the Automotive Industry. Oliver Wyman Automotive, s.l. (2007). https://books.google.de/books?id=jtcGkAEACAAJ Accessed 20 Nov 2022
3. Müller, L., Neubauer, K., Haubelt, C.: Exploiting similarity in evolutionary product design for improved design space exploration. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation vol. 13227, pp. 33–49. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-04580-6_3. Series Title: Lecture Notes in Computer Science
4. Grimm, K.: Software technology in an automotive company—major challenges. In: Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 498–503. IEEE, Portland, OR, USA (2003). https://doi.org/10.1109/ICSE.2003.1201228
5. Duffy, S.M., Duffy, A.H.B., MacCallum, K.J.: A design reuse model. In: Proceedings of the International Conference on Engineering Design (ICED 95), pp. 490–495 (1995)
6. Girczyc, E., Carlson, S.: Increasing design quality and engineering productivity through design reuse. In: Proceedings of the 30th International on Design Automation Conference—DAC '93, pp. 48–53. ACM Press, Dallas, Texas, United States (1993). https://doi.org/10.1145/157485.164565
7. Nazareth, D.L., Rothenberger, M.A.: Assessing the cost-effectiveness of software reuse: a model for planned reuse. J. Syst. Softw. **73**(2), 245–255 (2004). https://doi.org/10.1016/S0164-1212(03)00248-6
8. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Comput. Surv. **28**(2), 415–435 (1996). https://doi.org/10.1145/234528.234531
9. Simpson, T.W.: Product platform design and customization: status and promise. Artif. Intell. Eng. Des. Anal. Manuf. **18**(1), 3–20 (2004). https://doi.org/10.1017/S0890060404040028

10. Ochoa, L., González-Rojas, O., Juliana, A.P., Castro, H., Saake, G.: A systematic literature review on the semi-automatic configuration of extended product lines. J. Syst. Softw. **144**, 511–532 (2018). https://doi.org/10.1016/j.jss.2018.07.054
11. Pirmoradi, Z., Wang, G.G., Simpson, T.W.: A review of recent literature in product family design and platform-based product development. In: Simpson, T.W., Jiao, J., Siddique, Z., Hölttä-Otto, K. (eds.) Advances in product family and product platform design, pp. 1–46. Springer, New York, NY (2014). https://doi.org/10.1007/978-1-4614-7937-6_1
12. Axelsson, J.: Evolutionary architecting of embedded automotive product lines: an industrial case study. In: 2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, pp. 101–110. IEEE, Cambridge, United Kingdom (2009). https://doi.org/10.1109/WICSA.2009.5290796
13. Svahnberg, M., Bosch, J.: Evolution in software product lines: two cases. J. Softw. Maint. Res. Pract. **11**(6), 391–422 (1999). https://doi.org/10.1002/(SICI)1096-908X(199911/12)11:6<391::AID-SMR199>3.0.CO;2-8
14. Lim, W.C.: Effects of reuse on quality, productivity, and economics. IEEE Softw. **11**(5), 23–30 (1994). https://doi.org/10.1109/52.311048
15. Haubelt, C., Feldmann, R.: SAT-based techniques in system synthesis. In: 2003 Design, Automation and Test in Europe Conference and Exhibition, pp. 1168–1169. IEEE Comput. Soc, Munich, Germany (2003). https://doi.org/10.1109/DATE.2003.1253784
16. Neubauer, K., Haubelt, C., Glab, M.: Supporting composition in symbolic system synthesis. In: 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pp. 132–139. IEEE, Agios Konstantinos, Samos Island, Greece (2016). https://doi.org/10.1109/SAMOS.2016.7818340
17. Andres, B., Gebser, M., Schaub, T., Haubelt, C., Reimann, F., Glaß, M.: Symbolic system synthesis using answer set programming. In: Cabalar, P., Son, T.C. (eds.) Logic Programming and Nonmonotonic Reasoning, pp. 79–91. Springer, Berlin (2013)
18. Neubauer, K., Wanko, P., Schaub, T., Haubelt, C.: Exact multi-objective design space exploration using ASPmT. In: 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 257–260. IEEE, Dresden (2018). https://doi.org/10.23919/DATE.2018.8342014
19. Neubauer, K., Wanko, P., Schaub, T., Haubelt, C.: Enhancing symbolic system synthesis through ASPmT with partial assignment evaluation. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2017, pp. 306–309. IEEE, Lausanne, Switzerland (2017). https://doi.org/10.23919/DATE.2017.7927005
20. Andres, B., Biewer, A., Romero, J., Haubelt, C., Schaub, T.: Improving coordinated SMT-based system synthesis by utilizing domain-specific heuristics. In: LPNMR. Lecture notes in computer science, vol. 9345, pp. 55–68. Springer, s.l. (2015). https://doi.org/10.1007/978-3-319-23264-5_6
21. Biewer, A., Andres, B., Gladigau, J., Schaub, T., Haubelt, C.: A Symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 357–362. IEEE Conference Publications, Grenoble, France (2015). https://doi.org/10.7873/DATE.2015.0606
22. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory Solving Made Easy with Clingo **5**, 15 (2016). https://doi.org/10.4230/OASICS.ICLP.2016.2. Artwork Size: 15 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany
23. Kaminski, R., Romero, J., Schaub, T., Wanko, P.: How to build your own ASP-based system?! Theory Pract. Log. Program. (2021). https://doi.org/10.1017/S1471068421000508
24. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th conference on design automation—DAC '01, pp. 530–535. ACM Press, Las Vegas, Nevada, United States (2001). https://doi.org/10.1145/378239.379017
25. Gebser, M., Kaufmann, B., Romero, J., Otero, R., Schaub, T., Wanko, P.: Domain-specific heuristics in answer set programming. Proc. AAAI Conf. Artif. Intell. **27**(1), 350–356 (2013). https://doi.org/10.1609/aaai.v27i1.8585
26. Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., Shchekotykhin, K.: Combining Answer Set Programming and domain heuristics for solving hard industrial problems (Application Paper). Theory Pract. Log. Program. **16**(5–6), 653–669 (2016). https://doi.org/10.1017/S1471068416000284
27. Gebser, M., Ryabokon, A., Schenner, G.: Combining Heuristics for Configuration Problems Using Answer Set Programming. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.), Logic Programming

and Nonmonotonic Reasoning vol. 9345, pp. 384–397. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23264-5_32. Series Title: Lecture Notes in Computer Science
28. Thompson, M., Pimentel, A.D.: Exploiting domain knowledge in system-level MPSoC design space exploration. J. Syst. Archit. **59**(7), 351–360 (2013). https://doi.org/10.1016/j.sysarc.2013.05.023
29. Richthammer, V., Scheinert, T., Glaß, M.: Data Mining in System-Level Design Space Exploration of Embedded Systems. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation vol. 12471, pp. 52–66. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60939-9_4. Series Title: Lecture Notes in Computer Science
30. Blickle, T., Teich, J., Thiele, L.: System-level synthesis using evolutionary algorithms. Des. Autom. Embed. Syst. **3**(1), 23–58 (1998). https://doi.org/10.1023/A:1008899229802
31. Crescenzi, P., Rossi, G.: On the Hamming distance of constraint satisfaction problems. Theor. Comput. Sci. **288**(1), 85–100 (2002). https://doi.org/10.1016/S0304-3975(01)00146-3
32. Müller, L., Wanko, P., Neubauer, K.: Version 1.0.0—Publication on IJPP Special Issue Samos. https://github.com/krr-up/asp-dse/releases/tag/v1.0.0 Accessed 29 Jan 2023
33. Haubelt, C., Müller, L., Neubauer, K., Schaub, T., Wanko, P.: Evolutionary system design with answer set programming. Algorithms (2023). https://doi.org/10.3390/a16040179
34. Potassco: Clingo homepage. https://potassco.org/clingo/. Accessed 20 Nov 2022
35. Potassco: Clingo-dl Homepage. https://potassco.org/labs/clingodl/ Accessed 2022-12-01
36. Neubauer, K., Haubelt, C., Wanko, P., Schaub, T.: Systematic test case instance generation for the assessment of system-level design space exploration approaches. In: Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation Von Schaltungen Und Systemen, Tübingen, p. 10 (2018). https://doi.org/10.15496/publikation-25685
37. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evolut. Comput. **7**(2), 117–132 (2003). https://doi.org/10.1109/TEVC.2003.810758