

Improving Automatic Design Space Exploration by Integrating Symbolic Techniques into Multi-Objective Evolutionary Algorithms

Christian Haubelt, Thomas Schlichter, and Jürgen Teich

Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
email: {haubelt, schlichter, teich}@cs.fau.de

Abstract: Solving Multi-objective Combinatorial Optimization Problems (MCOPs) is often a twofold problem: Firstly, the feasible region has to be identified in order to, secondly, improve the set of non-dominated solutions. In particular, problems where the construction of a single feasible solution is \mathcal{NP} -complete are most challenging. In the present paper, we will propose a combination of Multi-Objective Evolutionary Algorithms (MOEAs) with Symbolic Techniques (STs) to solve this problem. Different Symbolic Techniques, such as Binary Decision Diagrams (BDDs), Multi-valued Decision Diagrams (MDDs), and SAT solvers as known from digital hardware verification will be considered in our methodology. Experimental results from the area of automatic design space exploration of embedded systems illustrate the benefits of our proposed approach. As a key result, the integration of STs in MOEAs is particularly useful in the presence of large search spaces containing only few feasible solutions.

Keywords: MOEA, Symbolic Techniques, Design Space Exploration

I. Introduction

Multi-Objective Evolutionary Algorithms (MOEAs) have been shown to perform well in solving Multi-objective Optimization Problems [37, 40, 12, 10, 7, 2, 3]. Beside the improvement of the set of non-dominated solutions, MOEAs have to face the problem of identifying and guiding the search towards the feasible region. This is especially interesting if we consider Multi-objective Combinatorial Optimization Problems (MCOPs) where the construction of a single feasible solution is \mathcal{NP} -complete as is the case, e.g., for the *Hamiltonian path* [20] and for the *system synthesis* problem [6].

The prototype of these problems is the well known *satisfiability problem* (SAT). In SAT, a 0/1-assignment to variables is sought such that a given Boolean formula evaluates to true

(is satisfied). If such an assignment exists, the given Boolean formula is said to be *satisfiable*. This basic problem can be reduced in polynomial time to all \mathcal{NP} -complete problems. Many methods and tools exist to solve SAT problems. These methods are usually known as *Symbolic Techniques* (STs). On the other hand, each \mathcal{NP} -complete problem can be reduced polynomially to the SAT problem, thus allowing to apply Symbolic Techniques to these problems.

In this paper, we consider the problem of *automatic design space exploration* of embedded systems which is the (multi-objective) optimization variant of the system synthesis problem. Most popular examples of embedded systems include automotive and telecommunication electronics as well as industrial and domestic automation systems. The goal in automatic design space exploration is to optimize an embedded system with respect to many objectives while meeting several constraints imposed on the solution, the so-called *implementation*. Typical objectives in embedded system design are: the cost of a system, its power consumption, its weight, the data throughput, etc.

In order to allow an unbiased search, the task of *design space exploration* is performed before selecting (*decision making*) the actual implementation. Here, MOEAs have been shown to perform well [26, 23, 15, 6]. The success of these approaches lies in the fact that (i) MOEAs improve a set of solutions, (ii) MOEAs do not assume any properties of the objective functions, and (iii) MOEAs work well in large and non-convex search spaces [5].

However, all these approaches fail or at least perform poorly in the presence of search spaces containing only very few feasible solutions. To overcome this drawback, we will propose the integration of Symbolic Techniques in the MOEA in order to guide the search towards the feasible region. Although we will use the example of automatic design space exploration to develop these novel ideas, the resulting methodology is not limited to this particular application.

The remaining of the paper is organized as follows: Section II discusses some related work, before Section III introduces the \mathcal{NP} -complete problem of system synthesis. The corresponding multi-objective optimization problem is known as *automatic design space exploration* which can be solved by using MOEAs. This issue will be discussed in Section IV. The main contributions of this paper, the integration of Symbolic Techniques into MOEAs, are presented in Section V. An H.264 encoder/decoder system will be used throughout the paper to illustrate our approach and derive first results (Section VI). In order to thoroughly evaluate our novel methodology, benchmark results will be presented in Section VII. Finally, Section VIII will conclude the paper and will show some future research directions.

II. Related Work

An overview of MCOPs can be found in [16]. Different techniques including MOEAs are proposed in literature to solve these problems cf. [33, 32, 29, 19]. One of the most popular MCOPs is the Multi-Objective Traveling Salesperson Problem (MOTSP) [39]. In MOTSP, a permutation of vertices in a fully connected graph is sought such that the route given by the permutation is optimal in several objectives simultaneously. When considering a non-fully connected graph, already the construction of a single feasible solution, i.e., a route to visit all vertices by only using the given edges, is \mathcal{NP} -complete. This decision problem is known as the Hamiltonian path problem. As a consequence, the objectives of an infeasible solution cannot be determined as no valid evaluation function exists.

Another challenging constrained MCOP with similar properties is the automatic design space exploration of embedded systems [23, 26, 22]. The basic optimization problem is a combined selection and assignment problem where appropriate hardware resources must be selected for the implementation of the embedded system and the processes which represent the application must be assigned to the selected resources. The task of assigning processes to resources is known to be \mathcal{NP} -complete [6]. Obviously, there is an inherent tradeoff between the number of resources used and the performance of the systems. Many successful design space exploration methodologies based on MOEAs are reported in literature [26, 23, 15, 6]. However, due to data dependencies among the processes, nearly all possible implementations are *infeasible*. As in the case of the Hamiltonian path problem, this infeasibility prohibits the evaluation of the design point. In order to overcome this drawback, we will use Symbolic Techniques inside the MOEA to guide the search towards the feasible region. Other *constraint optimization problems* known from literature are often constrained in the objective space only (see e.g. [12, 1]).

III. System Synthesis

In this paper, we consider the problem of automatic design space exploration for embedded systems. In contrast to general purpose computers, embedded systems are designed for a particular application while satisfying several constraints. Basically, the design space exploration problem is a multi-objective selection and assignment problem. In this section, we will introduce the necessary background and mathematical notation.

A. Defining the Search Space

To allow a mathematical model of the search space, the concept of a so-called *specification graph* is needed. A specification graph specifies an embedded system by means of its applications, architectures, and the relation between these two views. Here, we use a graph-based approach already proposed in [6]. The specification graph consists of three parts:

- The application by means of its processes and their data dependencies.
- The possible target architectures are represented by computing elements (e.g. ASIC, FPGA, DSP) and their possible interconnects.
- Mapping edges that indicate which process can be implemented on which computing element.

The application is modeled by a so-called *process graph* $g_p = (V_p, E_p)$. Vertices $v \in V_p$ model processes whereas edges $e \in E_p \subseteq V_p \times V_p$ represent data dependencies between processes.

Example 1: The example in Figure 1 shows the process graph of an H.264 video codec for image compression [34].

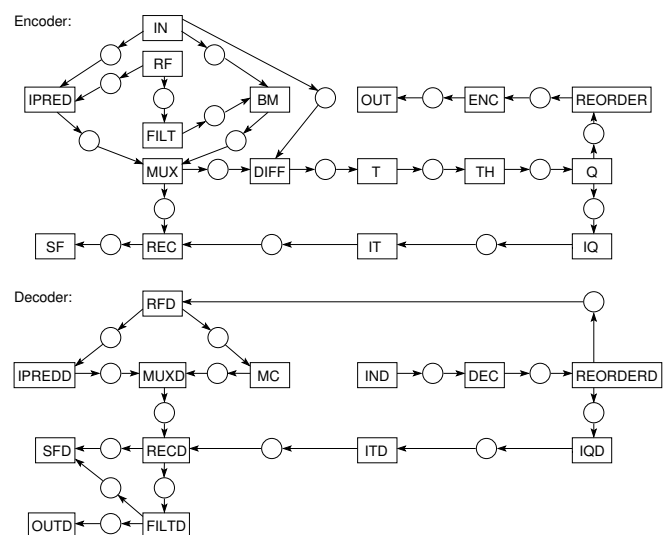


Figure 1: A process graph g_p modeling an H.264 encoder in the upper part and an H.264 decoder in the lower part.

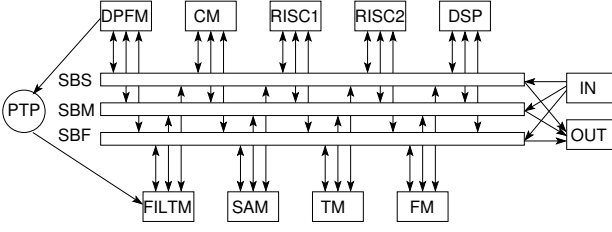


Figure 2: An architecture graph g_a representing a template for the possible architectures available for the H.264 encoder and decoder shown in Figure 1. Three different buses SBS, SBM, and SBF can be allocated.

The upper part of Figure 1 shows the encoder and the lower part shows the decoder. The encoder consists of 37, the decoder of 28 processes. Here, we use rectangular vertices and circles to distinguish between operation and communication processes. For an explanation of the algorithm we refer to [34].

The architecture including functional resources and buses is also modeled by a directed graph termed *architecture graph* $g_a = (V_a, E_a)$. Vertices $v \in V_a$ model functional resources (RISC processor, coprocessors, or ASIC) and communication resources (shared buses or point-to-point connections). An edge $e \in E_a$ models a directed link between two resources. All the resources are viewed as *potentially allocatable* components.

Example 2: The process graph given in the previous example is mapped onto a target architecture shown in Figure 2. For the sake of readability, edges with two arrows are used to indicate bidirectional links. The architecture may consist of three shared bus with different communication bandwidths, two memory modules (a single and a dual ported memory), two programmable RISC processors, a signal processor (DSP), several predefined hardware modules namely a filter module (FILTM), a module for performing the transformation operations (TM), an subtract/adder module (SAM), an entropy encoder/decoder module (CM), and I/O devices (INM and OUTM).

Next, it is shown how user-defined mapping constraints representing possible bindings of processes onto resources can be specified in a graph based model.

Definition 1 [Specification Graph]: A *specification graph* $g_s(V_s, E_s)$ consists of a process graph $g_p(V_p, E_p)$, an architecture graph $g_a(V_a, E_a)$, and a set of *mapping edges* E_m . In particular, $V_s = V_p \cup V_a$, $E_s = E_p \cup E_a \cup E_m$, where $E_m \subseteq V_p \times V_a$.

Mapping edges relate the vertices of the process graph to vertices of the architecture graph. The edges represent user-defined mapping constraints in the form of a relation: “can be implemented by”.

Example 3: Figure 3 shows an example of a specification graph. As process graph we use a part of the H.264 process graph from Figure 1 and as architecture graph we use a part of the architecture graph from Figure 2. The dashed edges

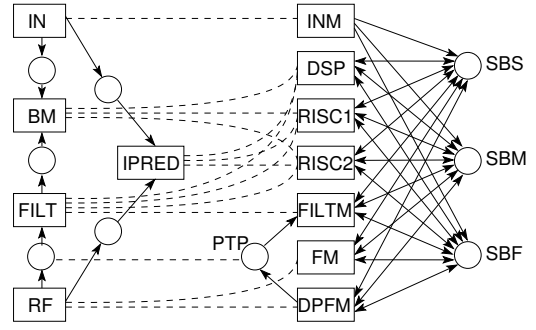


Figure 3: An example specification graph g_s consisting of a process graph g_p (part of Figure 1) and an architecture graph g_a (part of Figure 2). Additionally, the dashed lines model mapping edges E_m . In this example, it is possible to bind all communication nodes to the buses SBS, SBM, and SBF (not shown in figure).

between the two graphs are the additional mapping edges E_m that describe the possible mappings. For example, process BM can be executed on RISC1, RISC2, or the DSP. In this example, it is possible to bind all communication nodes to the buses SBS, SBM, and SBF (not shown in figure).

B. System Synthesis

Before discussing automatic design space exploration in detail, we formalize the notion of a *feasible implementation* (cf. [6]). An implementation, being the result of a system synthesis, consists of two parts:

- (1) The *allocation* that indicates which elements of the architecture graph are used in the implementation and
- (2) the *binding*, i.e., the set of mapping edges which define the binding of vertices in the process graph to components of the architecture graph.

Before defining the term *implementation* formally, Blickle et al. [6] introduce the so-called *activation* of vertices and edges:

Definition 2 [Activation]: The *activation* of a specification graph $g_s = (V_s, E_s)$ is a function $a : V_s \times E_s \mapsto \{0, 1\}$ that assigns to each edge $e \in E_s$ and to each vertex $v \in V_s$ the value 1 (activated) or 0 (not activated).

The task of system synthesis is to determine an implementation, i.e., assigning activity values to vertices and edges of the specification graph. An implementation consists of an *allocation* and a *binding*. For the sake of simplicity, in the following it is assumed that all vertices $v \in V_p$ and all edges $e \in E_p$ of the process graph g_p are activated.

Definition 3 [Allocation]: An *allocation* α of a given specification graph g_s is the subset of all activated vertices and edges of the architecture graph g_a , i.e., $\alpha = \{v \in V_a \mid a(v) = 1\} \cup \{e \in E_a \mid a(e) = 1\}$

Definition 4 [Binding]: A *binding* β of a given specification graph g_s is the subset of activated mapping edges E_m , i.e., $\beta = \{e \in E_m \mid a(e) = 1\}$

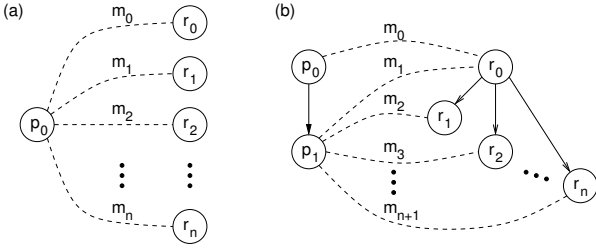


Figure 4: (a) For each process $p \in V_p$ exactly one outgoing mapping edge $m_i \in \{m_0, m_1, \dots, m_n\}$ has to be activated. (b) In order to establish the required communication $(p_0, p_1) \in E_p$, we have to execute the processes p_0 and p_1 on the same resource r_0 or on an adjacent resource $r_i \in \{r_1, r_2, \dots, r_n\}$ where $(r_0, r_i) \in E_a$.

To identify the feasible region, it is necessary to determine the set of *feasible allocations* and *feasible bindings*. A feasible binding guarantees that communications demanded by the process graph can be established in the allocated architecture. This property makes the resulting optimization problem hard to be solved.

Definition 5 [Feasible Binding]: Given a specification graph g_s and an allocation α , a *feasible binding* is a binding β that satisfies the following requirements:

- (1) Each activated mapping edge $e \in \beta$ ends at an activated vertex, i.e., $\forall e = (v_p, v_a) \in \beta : v_a \in \alpha$.
- (2) For each process graph vertex $v \in V_p$, exactly one outgoing mapping edge $e \in E_m$ is activated, i.e., $|\{e \in \beta \mid e = (v_p, v_a), v_a \in V_a\}| = 1$.
- (3) For each process graph edge $e = (v_i, v_j) \in E_p$:

- either both processes are mapped onto the same vertex, i.e., $\tilde{v}_i = \tilde{v}_j$ with $(v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$,
- or there exists an activated edge $\tilde{e} = (\tilde{v}_i, \tilde{v}_j) \in E_a \cap \alpha$ in the architecture graph to handle the communication associated with edge e , i.e., $(\tilde{v}_i, \tilde{v}_j) \in E_a \cap \alpha$ with $(v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$.

Example 4: The second and third requirement are depicted in Figure 4: To achieve a feasible binding, the second requirement makes the process p_0 in Figure 4(a) to be bound to exactly one of the allocated resources $r_0 \dots r_n$. The third requirement is depicted in Figure 4(b). Here, the process p_0 must be bound to the resource r_0 and thus the dependent process p_1 must be bound either to the same resource r_0 or to one of the adjacent resources $r_1 \dots r_n$.

Definition 6 [Feasible Allocation]: A *feasible allocation* is an allocation α that allows at least one feasible binding β .

With the above discussion, we can define an *implementation* by means of a feasible allocation and a feasible binding.

Definition 7 [Implementation]: Given a specification graph g_s , a (*feasible*) *implementation* ψ is a pair (α, β) where α is a feasible allocation and β is a corresponding feasible binding.

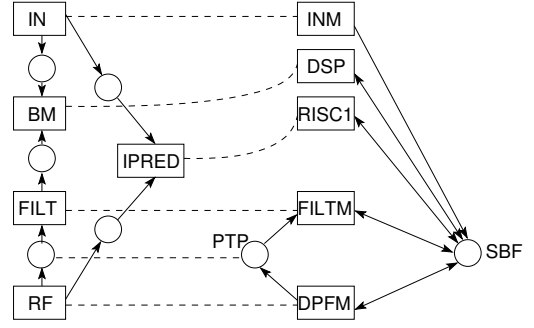


Figure 5: One possible implementation for the specification given in Figure 1. Only the fast shared bus (SBF) is allocated, but every module is implemented on an individual hardware resource.

Example 5: Consider the case that the allocation of vertices in Figure 3 is given as $\alpha = \{\text{INM}, \text{DSP}, \text{RISC1}, \text{FILTM}, \text{DPFM}, \text{SBF}, \text{PTP}\}$. A feasible binding can be given by $\beta = \{(\text{IN}, \text{INM}), (\text{BM}, \text{DSP}), (\text{FILT}, \text{FILTM}), (\text{RF}, \text{DPFM}), (\text{IPRED}, \text{RISC1}), (\text{C_IN_BM}, \text{SBF}), (\text{C_IN_IPRED}, \text{SBF}), (\text{C_FILT_BM}, \text{SBF}), (\text{C_RF_IPRED}, \text{SBF}), (\text{C_RF_FILT}, \text{PTP})\}$. This allocation and binding (both shown in Figure 5) is indeed feasible, i.e., the implementation $\psi = (\alpha, \beta)$ is feasible.

Given the implementation ψ , some properties of ψ can be calculated. This can be done analytically or simulation-based. The task of evaluation of objectives is not in the scope of the paper. However, in our experiments we will estimate the power dissipation, the implementation cost, as well as the latency of the implementation which are typical objectives in embedded system design.

IV. Automatic Design Space Exploration Using Multi-Objective Evolutionary Algorithms

Beside the problem of determining a single feasible solution, it is also important to identify the set of optimal solutions. This is done during the automatic design space exploration. In the following we will discuss the optimization problem as well as the application of MOEAs to this challenging task. This introduction is needed to develop the idea of integrating Symbolic Techniques into MOEAs.

A. The Optimization Problem

The task of automatic design space exploration can be formulated as a *Multi-objective Combinatorial Optimization Problem*.

Definition 8 [Automatic Design Space Exploration]: The task of *automatic design space exploration* is the following multi-objective optimization problem (see e.g., [40]) where without loss of generality, only minimization problems are assumed here:

minimize $f(x)$,
 subject to:
 x represents a feasible implementation ψ ,
 $c_i(x) \leq 0, \forall i \in \{1, \dots, q\}$

where $x = (x_1, x_2, \dots, x_m) \in X$ is the *decision vector*, X is the *decision space*, $f(x) = (f_1(x), f_2(x), \dots, f_n(x)) \in Y$ is the *objective function* and Y is the *objective space*.

Here, x is an encoding called *decision vector* representing an implementation ψ . Moreover, there are q constraints $c_i(x), i = 1, \dots, q$ imposed on x defining the set of feasible implementations. The *objective function* f is n -dimensional, i.e., n objectives are optimized simultaneously. For example, in embedded system design it is required that the monetary cost and the power dissipation of an implementation are minimized simultaneously. Often, objectives in embedded system design are conflicting [17].

Only those *design points* $x \in X$ that represent a feasible implementation ψ and that satisfy all constraints c_i , are in the set of feasible solutions, or for short in the *feasible set* called $X_f = \{x \mid \psi(x) \text{ being feasible} \wedge c(x) \leq 0\} \subseteq X$. The objective space of X is defined as $Y = f(X) \subset \mathbb{R}^n$, where the objective function f on the set X is given by $f(X) = \{f(x) \mid x \in X\}$. Analogously, the *feasible region* in the objective space is denoted by $Y_f = f(X_f) = \{f(x) \mid x \in X_f\}$.

In the following, the necessary definitions for multi-objective optimization problems are given (cf. [40, 27]). Without loss of generality, only minimization problems are considered.

Definition 9 [Pareto dominance]: For any two decision vectors a and b ,

$$\begin{aligned}
 a \succ b & \quad (a \text{ dominates } b) & \text{iff} & \quad \forall i : f_i(a) \leq f_i(b) \\
 & & & \quad \wedge \quad \exists i : f_i(a) < f_i(b) \\
 a \succeq b & \quad (a \text{ weakly dominates } b) & \text{iff} & \quad \forall i : f_i(a) \leq f_i(b) \\
 a \sim b & \quad (a \text{ is indifferent to } b) & \text{iff} & \quad \forall i : f_i(a) = f_i(b) \\
 a \parallel b & \quad (a \text{ is incomparable to } b) & \text{iff} & \quad \exists i, j : f_i(a) > f_i(b) \\
 & & & \quad \wedge \quad f_j(a) < f_j(b).
 \end{aligned}$$

Definition 10 [Pareto optimality]: A decision vector $x \in X_f$ is said to be non-dominated regarding a set $A \subseteq X_f$ iff $\nexists a \in A : a \succ x$. A decision vector x is said to be Pareto-optimal iff x is non-dominated regarding X_f .

The set of all Pareto-optimal solutions is called the *Pareto-optimal set*, or the *Pareto set* X_p for short. An approximation of the Pareto set X_p will be termed *approximation set* X_a in the following. Furthermore, the *Pareto-optimal front* is given by $Y_p = f(X_p)$.

B. The Basic Multi-Objective Evolutionary Algorithm

In this section, we will show how to solve the automatic design space exploration problem by using Multi-Objective Evolutionary Algorithms. This basic algorithm was already presented in [6] and [23]. The MOEA determines the allocation and bindings. In this paper, we are mainly concerned about the encoding and decoding of solutions. The

decoding

IN: The individual j consisting of allocation $alloc$, repair allocation priority list L_R , binding order list L_O , and binding priority list $L_B(v)$.

OUT: The allocation α and the binding β if both are feasible, (\emptyset, \emptyset) if no feasible binding is represented by the individual j

BEGIN

$\bar{\alpha} \leftarrow \text{allocation}(alloc(j), L_R(j))$

$\bar{\beta} \leftarrow \text{binding}(L_B(j), L_O(j), \bar{\alpha})$

IF $\bar{\beta} = \emptyset$

$\text{RETURN } (\emptyset, \emptyset)$

ENDIF

$\beta \leftarrow \bar{\beta}$

$\alpha \leftarrow \text{update_allocation}(\bar{\alpha}, \bar{\beta})$

$\text{RETURN } (\alpha, \beta)$

END

Figure 6: Algorithm to decode the allocation α and the binding β from a given individual j .

actual Evolutionary optimization can be performed with any state-of-the-art MOEA like NSGA-II [13], SPEA2 [42], ECEA [28], CNSGA-II [14], or IBEA [41].

The main decoding idea is outlined in Figure 6:

- (1) The allocation of resources $v \in V_a$ is decoded from the individual and repaired with a simple heuristic (the function `allocation`),
- (2) the binding of the edges $e \in E_m$ is performed (the function `binding`), and
- (3) the allocation is updated in order to eliminate unnecessary vertices $v \in V_a$ from the allocation and all necessary edges $e \in E_a$ are added to the allocation (the function `update_allocation`()).

Thus, the **decoding** function results in a feasible allocation and binding of the vertices and edges of the process graph g_p to the vertices and edges of the architecture graph g_a . If no feasible binding could be found, the whole decoding of the individual is aborted.

The allocation of vertices is directly encoded in the *chromosome*, i.e., the elements in a vector $alloc$ encode for each vertex $v \in V_a$ if it is activated or not, i.e., $a(v) = alloc(v)$. This simple encoding might result in many infeasible allocations. The repair heuristic only adds new vertices $v \in V_a$ to the allocation and reflects the simplest case of infeasibility that may arise from non-executable functional vertices: Consider the set $V_B \subseteq V_p$ that contains all vertices that can not be executed, because not a single corresponding resource vertex is allocated, i.e., $V_B = \{v \in V_p \mid \forall (v, \tilde{v}) \in E_m : a(\tilde{v}) = 0\}$. To make the allocation feasible (in this sense), we add for each $v \in V_B$, at most one $\tilde{v} \in V_a$, until feasibility in the sense above is achieved.

The order in which additional resources are added has a large influence on the resulting allocation. For example, one

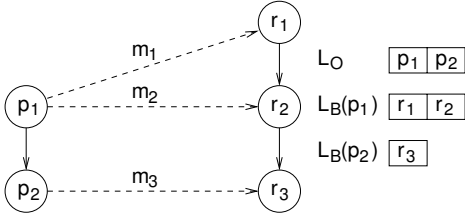


Figure 7: Example of the binding order list L_O and the binding priority lists L_B . For this example, the sequential decoder does not find a feasible solution.

could be interested in an additional allocation with minimal cost. As this depends on the optimization goal expressed in the objective function f , the order should automatically be adapted. This will be achieved by the introduction of a *repair allocation priority list* L_R coded in the individual. In this list, all resources $v \in V_a$ are contained and the order in the list determines the order the vertices will be added to the allocation.

In this paper, the function *binding* is of special interest and should be discussed in more detail. A binding is obtained by activating exactly one incident edge $e \in E_m$ for each allocated vertex $v \in V_p$. This is achieved in the following way: All processes are bound in the order they appear in the binding order list L_O . For each process $p \in V_p$, a list L_B is encoded as allele that contains all incident edges $e \in E_m$ to p . This list is seen as a priority list and the first edge e_k with $e_k = (v, \tilde{v})$ that gives a feasible binding is included in the binding, i.e., $a(e_k) := 1$.

Example 6: An example of the binding order list L_O and binding priority list L_B are shown in Figure 7. Process p_1 is bound before process p_2 . For process p_1 , it is tested if it can be bound to resource r_1 before checking r_2 .

The test of feasibility is directly related to the definition of a feasible binding (see Definition 5). As the priority lists contain all incident edges $e \in E_m$, each individual contains a feasible binding iff it contains a feasible allocation. Calculating a feasible binding for a given allocation is equivalent to solve the underlying satisfiability problem. To solve this problem, we can use fast heuristics that try to find a feasible binding, or we use prohibitively slow exact methods. Different methods are described and compared in the next section. When no feasible binding is found, β is the empty set, and the individual will be given a penalty value as its fitness value. Finally, in the function *update_allocation*, vertices of the allocation that are not used will be removed from the allocation. Furthermore, all edges $e \in E_a$ in the architecture graph g_a are added to the allocation that are necessary to obtain a feasible binding.

V. Integrating Symbolic Techniques

As the determination of a feasible binding is \mathcal{NP} -complete [6], the basic MOEA as presented above uses a simple

heuristic to bind one mapping edge after each other sequentially and focuses only on a small scope of the specification. Hence, this approach might be trapped (as will be shown later). As each \mathcal{NP} -complete problem can be reduced polynomially to the SAT problem, it is possible to apply Symbolic Techniques (STs) to these problems. There are two advantages:

(1) The optimization might benefit from sophisticated heuristics to solve the SAT problem and (2) a wider scope in contrast to the basic MOEA can be considered during decoding. The first advantage is obvious while the latter needs some explanation: Choosing a mapping edge for the binding resembles the assignment of a 1 to a variable in the corresponding Boolean formula. STs permit to check if there exists at least one feasible completion that satisfies the Boolean formula.

We will discuss this topic in the following in more depth. We will start with the simple heuristic called *sequential decoder* that is used by the basic MOEA presented above and is used by many automatic design space exploration tools. Next, we will show how the problem of determining a feasible binding can be reduced polynomially to the SAT problem. With this result, we will show how to use a) Binary Decision Diagrams (BDDs), b) Multi-valued Decision Diagrams (MDDs), and c) SAT solvers to improve the decoding of solutions.

A. Sequential Decoder

This is the most simple and a well known decoding algorithm. All processes are bound in the order they appear in the binding order list L_O of the individual. When binding a process, the different mapping edges are tested in the order of the binding priority list L_B . To find a feasible binding of the complete problem, only those mapping edges are chosen which satisfy a simple feasibility test. This simple feasibility test checks if the tested mapping edge is infeasible with the binding performed up to this point.

The mapping edges are activated sequentially and the outcome of this sequential decoder strongly depends on both, the binding order list L_O and the binding priority list L_B . As only the yet bound processes are considered, the sequential decoder might be trapped, even if the allocation is feasible, i.e., a feasible completion of the binding exists.

Example 7: An example is shown in Figure 7. Using the given binding order list L_O and the given binding priority lists L_B , the sequential decoder does not find the valid binding $\beta = \{(p_1, r_2), (p_2, r_3)\}$. This happens because p_1 is mapped to r_1 first, what does not conflict with any already activated mapping edge (as there is none of that ilk). However, that binding prohibits a feasible mapping of process p_2 . As a consequence, the sequential decoder fails to find the valid binding.

B. Symbolic Representation of the Specification

The problem of finding a feasible binding can be reduced polynomially to the satisfiability problem [18, 35, 31] by assigning

Boolean variables to mapping edges and resources, resulting in a symbolic representation. These Boolean variables indicate that the associated element is activated. Hence, we can use the activation introduced in Definition 5. The Boolean formula can be deduced by interpretation of the three requirements for a feasible binding from Definition 5:

The first requirement states that each activated mapping edge $m \in E_m$ has to end at an activated resource $r \in V_a$: This implication written in conjunctive normal form (CNF) is:

$$b_1(a(m), a(r)) = \bigwedge_{m=(p,r) \in E_m} \overline{a(m)} \vee a(r) \quad (1)$$

The second requirement states that exactly one mapping edge $m = (p, r) \in E_m$ is activated for each process $p \in V_p$ what can be split into two statements. At least one mapping edge m for each process p has to be activated:

$$b_2(a(m)) = \bigwedge_{p \in V_p} \bigvee_{m=(p,r) \in E_m} a(m) \quad (2)$$

And at most one mapping edge m for each process p has to be activated.

$$b_3(a(m)) = \bigwedge_{m_i=(p,r_i), m_j=(p,r_j) \in E_m: r_i \neq r_j} \overline{a(m_i)} \vee \overline{a(m_j)} \quad (3)$$

The last requirement states that communicating processes have to be mapped to the same or to an adjacent resource. This can be expressed by the following equation.

$$b_4(a(m)) = \bigwedge_{\substack{m_i=(p_i,r_i) \in E_m: \\ (p_i,p_j) \in E_p}} \left[\overline{a(m_i)} \vee \bigvee_{\substack{m_j=(p_j,r_j) \in E_m: \\ r_i=r_j \vee (r_i,r_j) \in E_a}} a(m_j) \right] \quad (4)$$

There is a feasible implementation for a given allocation α represented by $a(r)$ if and only if the conjunction of Equation (1) - (4) is satisfiable, i.e., an assignment to $a(m)$ exists:

$$\exists a(m) : b_1(a(m), a(r)) \wedge b_2(a(m)) \wedge b_3(a(m)) \wedge b_4(a(m)) \quad (5)$$

In the following we will present three different approaches to solve this satisfiability problem.

C. BDD Decoder

As stated before, finding a feasible binding for a given allocation is a complex task. The binding is performed by binding one process after each other in the order of the binding order list L_O of the individual. When binding a process, the different mapping edges are tested in the order of the binding priority list L_B . To find a feasible binding of the complete problem, only those mapping edges are chosen which satisfy a feasibility test.

One possibility to perform this decoding is to check whether there still exists a valid completion for all the unbound processes if the currently tested mapping edge is chosen. This will always find the correct binding for each process, and each feasible allocation results in a feasible binding β . Solving this problem means to solve the satisfiability problem stated in the Equation (5).

The basic idea of using Binary Decision Diagrams (BDD [8]) is to encode Equation (5) in a single BDD. A BDD is a tree-based data structure to represent Boolean formula. The leaves of the BDDs are the Boolean values 0 and 1. All paths in the BDD from the root to the 1 represent assignments that satisfy the encoded Boolean formula. Thus, when applied to the decoding in automatic design space exploration, all paths from the root of the BDD to the 1 represent feasible bindings.

Summary Pros

The BDD for Equation (5) represents the feasible region X_f symbolically. This BDD can be built once and be used for each individual. After BDD construction, the test if a mapping edge m still allows a feasible binding is just as simple as combining $a(m)$ with the BDD by a logic AND-function. This test can be done in $\mathcal{O}(|E_m| + |V_a|)$ once the BDD is built and is known as functional simulation with BDDs [36].

Summary Cons

Unfortunately, for real-world problems, the BDD is prohibitively large (memory requirements) and thus this approach is not viable.

To overcome this drawback, we constructed a small BDD for each process as compromise. It is tested if the selection of a certain mapping edge $m = (p, r)$ for a process p prohibits a feasible binding of any direct predecessor or successor of p . To do so, Equations (1) - (3) are split into parts that contain only mapping edges incident to a single process p . These Boolean equations are encoded in the BDD of the corresponding process. Equation (4) always contains mapping edges of different processes. Hence, Equation (4) must be considered in the BDDs associated with p_i and p_j .

With these BDDs, we can test if the activation of mapping edge m_i prohibits the feasibility of the adjacent processes. Therefore, we have to set $a(m_i) = 1$ for the BDD associated with process p_i belonging to m_i , and for all the BDDs associated with succeeding or proceeding processes p_j . If one of the BDDs collapses to a logic 0 value, the mapping edge m_i would not allow to find a feasible binding and thus will be rejected.

This method solves the problem shown in Figure 7. Of course, this BDD decoder generally is not able to find a feasible binding for each feasible allocation.

Example 8: This is illustrated in Figure 8 where only two feasible bindings exists: $\{m_1, m_2, m_4, m_6\}$ and $\{m_3, m_5, m_7, m_8\}$. But as p_1 and p_4 do not have common neighbors, even the BDD feasibility test may select the mapping edges m_1 and m_8 what prohibits a feasible binding.

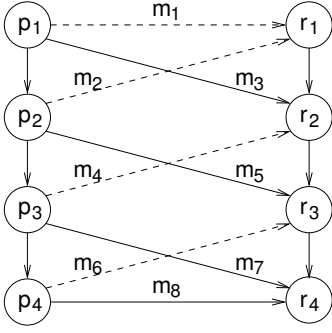


Figure 8: Example of four processes that should be bound onto four resources. In the worst case the BDD decoder will be trapped by this example (binding p_1 and p_4 before p_2 and p_3).

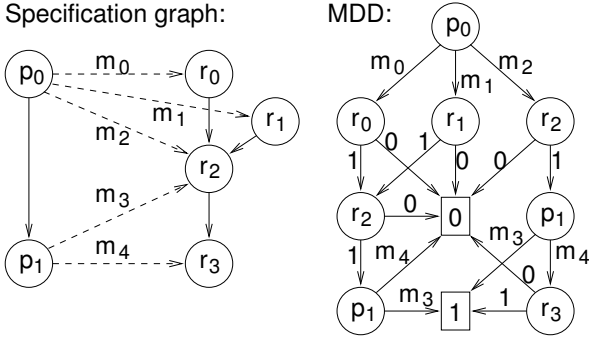


Figure 9: An example specification graph and the corresponding MDD.

Later in this paper, we will show, that the BDD decoder can improve the convergence speed compared with the sequential decoder even though the test itself is slower.

D. MDD Decoder

The MDD decoder is similar to the approach by using a single BDD that encodes the complete satisfiability problem (Equation (5)) of finding a valid completion for a binding. The difference is that it is based on Multi-valued Decision Diagrams (MDD [25]) which support integer valued variables. The operator that supports integer valued variables in MDDs is the *CASE*-operator. It selects a value of a subtree depending on an additional integer valued variable. Due to the use of integer values instead of binary values, the size (memory requirement) of the MDD is expected to be less than the size of the BDD. In order to use Equation (5) with MDDs, we assign different integer values for the mapping edges $m = (p, r)$ belonging to a process p . Thus we implicitly take care that exactly one mapping edge is selected per process $p \in V_p$ during the evaluation of the MDD.

We use the recursive algorithm shown in Figure 10 to build the MDD used for the decoding. This algorithm mainly uses the *CASE*-operator and the Boolean *AND*-function to create

Build_MDD

```

mdd ← true
for each root process p
  for each  $m_i$  of p
    mddi ← resource_of( $m_i$ )
    for each successor process s of p
      mddi ← mddi AND build_mdd_rekurs(s, resource_of( $m_i$ ))
    mdd ← mdd AND CASE( $p$ , mdd0, mdd1, ...)
RETURN (mdd)

```

build_mdd_rekurs(p, r)

```

for each  $m_i$  of p
  if not connected( $r$ , resource_of( $m_i$ ))
    mddi ← false
  else
    mddi ← resource_of( $m_i$ )
    for each successor process s of p
      mddi ← mddi AND build_mdd_rekurs(s, resource_of( $m_i$ ))
RETURN CASE( $p$ , mdd0, mdd1, ...)

```

Figure 10: Algorithm to build an MDD that encodes all feasible bindings of a given specification graph. The allocation of resources is represented by binary decision variables whereas the selection of mapping edges is encoded by integer values for each process.

the MDD. However, one can see from this algorithm that this approach is limited to acyclic process graphs.

Example 9: In Figure 9, an example of a specification graph together with its corresponding MDD is given. All paths in the MDD beginning with the root node p_0 and ending at 1 indicate valid bindings. All the paths ending at 0 indicate invalid bindings. Even though the MDD is smaller compared to a corresponding BDD, it still requires a lot of memory for large problems.

The decoding itself is the same as for the BDD decoder (functional simulation), the binding is performed by selecting one process after each other in the order of the binding order list L_O of the individual. When binding a process, the different mapping edges are tested in the order of the binding priority list L_B and the MDD is used to calculate the feasibility for this mapping edge. If the MDD collapses to zero, we know that using this particular mapping edges prohibits a feasible completion of the binding.

E. SAT Decoder

Finally, we will present a decoder based on SAT solvers. SAT solvers are designed to solve satisfiability problems as shown in Equation (5). The most important task is to force the SAT solver to regard the information encoded in the individual. Here we describe how to manage this by using a SAT solver based on the Davis-Putnam backtrack search algorithm [11].

A decision strategy used by the Davis-Putnam algorithm determines which unassigned variable will be set to a Boolean

value (0 or 1). If a clause of the CNF given to the SAT-solver is recognized as unsatisfiable, the SAT solver tries to resolve this conflict by a backtracking procedure. This continues until all variables are assigned, and this assignment represents a feasible solution. Thus, we developed a problem-specific decision strategy. Considering the encoded information of the individual, this decision strategy can be described by the following three steps:

(1) The Boolean value 1 is assigned to all allocated resources. The order does not matter as an allocation of a resource does not prohibit a feasible binding.

(2) The Boolean value 0 is assigned to every not activated resource in the reverse order of the allocation priority list L_R . This ensures the sequential addition of resources respecting the priority in L_R if the CNF is unsatisfiable with the given resource allocation.

(3) In the third step, we try to find a feasible binding based on the allocation determined in step 1 and step 2. The Boolean variables associated to mapping edges are set to true regarding the order of the binding order list L_O and the corresponding binding priority list L_B . If this assignment prohibits a feasible binding, the next variable in the binding priority list is tested. If the end of the binding priority list is reached without finding an assignment that allows a feasible solution, the backtracking algorithm will automatically return to step 2 and add additional resources.

Next we will present first results when using the three described Symbolic Techniques in automatic design space exploration.

VI. Results

In this section, first results of applying our four different decoders, Sequential decoder, BDD decoder, MDD decoder, and SAT decoder, to the problem of the H.264 decoder/encoder example will be presented. This example consists of 65 processes, 15 resources, and 274 mapping edges. The search space contains about 2^{140} solutions, where most of them are infeasible. The optimization is performed for three objectives namely *implementation cost*, *power dissipation*, and *latency*. The implementation uses the PISA (Platform independent Interface for Search Algorithms [4]) framework for optimization purposes. In the present work, the SPEA2 selection procedure [42] was applied. For the SAT decoding technique, we used the zChaff [30] SAT solver, for the BDD decoder we used the Buddy BDD library [9]. We implemented the MDD decoder using an MDD package from the University of Colorado at Boulder [38].

A. Evaluation Methods

For each individual decoding technique, we performed 10 optimization runs. Consequently, 40 approximation sets X_a were obtained. From these 40 approximation sets, we build a reference set X_R containing the non-dominated solutions over all runs. The performance assessment was done using

the ϵ -dominance and entropy indicator. For a discussion on indicators for performance assessment in multi-objective optimization, see [43].

ϵ -Dominance Laumanns et al. [28] introduce the concept of ϵ -dominance. A point a is said to *weakly ϵ -dominate* (in a minimization problem) a point b , denoted by $a \succeq_\epsilon b$ if and only if $a \succeq \epsilon \cdot b$. By scaling point b by a factor ϵ , point a is superior to point b .

Using the definition of ϵ -dominance, a binary quality indicator \mathcal{D}_ϵ can be defined.

$$\mathcal{D}_\epsilon(A, B) = \inf_\epsilon \{b \in B \mid \exists a \in A : a \succeq_\epsilon b\} \quad (6)$$

In practice, $\mathcal{D}_\epsilon(A, B)$ can be calculated in time $\mathcal{O}(n \cdot |A| \cdot |B|)$, where n is the number of objectives and $|A|$ and $|B|$ denote the cardinalities of A and B , respectively [43]:

$$\mathcal{D}_\epsilon = \max_{b \in B} \min_{a \in A} \max_{1 \leq i \leq n} \frac{a_i}{b_i}$$

Thus, a value of $\mathcal{D}_\epsilon(B, A) > 1$ and $\mathcal{D}_\epsilon(A, B) \leq 1$ indicates that A is better than B , i.e., every $b \in B$ is weakly dominated by at least one $a \in A$ and $A \neq B$, while $\mathcal{D}_\epsilon(A, B) = \mathcal{D}_\epsilon(B, A) = 1$ corresponds to the fact that $A = B$. A generalization of ϵ -dominance and Pareto-dominance called E-dominance and its application to archiving strategies is proposed in [24].

Entropy Method Gunawan et al. [21] propose the *entropy method* $\mathcal{E}(A)$ for measuring the diversity of an approximation set A . Hence, $\mathcal{E}(A)$ is a unary quality indicator. The basic idea is to use Shannon's entropy for probability distribution functions which is defined as:

$$H(P) = - \sum_{i=1}^n p_i \ln(p_i) \quad \text{with} \quad \sum_{i=1}^n p_i = 1 \quad (7)$$

where p_i is the probability that some random variable takes the value x_i . To apply the entropy method to an approximation set A , a Gaussian influence function $\omega_i(x)$ is associated with each member $a_i \in A$. The influence function for a particular point a_i has its maximum at that point ($x = a_i$) and decreases gradually with the distance. From all influence functions associated with an approximation set A , an aggregative density function D can be defined:

$$D(x) = \sum_{i=1}^{|A|} \omega_i(d_{a_i, x}), \quad (8)$$

where $d_{a_i, x}$ denotes the Euclidean distance from x to point a_i . In a last step, a mesh is constructed in the objective space and the normalized density function $\delta(x)$ can be measured at each point x_i of this mesh:

$$\delta(x_i) = \frac{D(x_i)}{\sum_{x_i} D(x_i)} \quad (9)$$

Note, that $\sum_{x_i} \delta(x_i) = 1$. As a result, a normalized distribution function is obtained and Shannon's entropy can be applied to this function, leading to the entropy quality indicator:

$$\mathcal{E}(A) = - \sum_{x_i} \delta(x_i) \ln(\delta(x_i)) \quad (10)$$

The observation is, that an approximation set A with a high entropy $\mathcal{E}(A)$ has a high diversity as well and vice versa.

B. Quantitative Results

For our experiments, we chose the parameters for the MOEA as follows: The population size was set to 400. For recombination, 100 children were created from 100 parents by single-point crossover. The mutation rate was set to $|\text{decision variables}|^{-1}$. The mutation operation is either a single bit flip or order-based mutation.

The ϵ -dominance results are shown in Figure 11 and Figure 12. We can see that all three decoders using symbolic techniques reach a better ϵ -dominance than the original sequential decoder. As expected, the best values are obtained by the SAT and MDD decoder. However, having a look at Figure 12, we can see that the construction of the MDD takes more time than the complete optimization of 200 generations using any of the other methods. Including this construction time, the complete optimization of 200 generations lasts 5544 seconds, making the MDD decoder not competitive. Note that the BDD decoder constructs a set of small BDDs (instead of a single BDD for the entire problem) making this procedure much faster in comparison to the MDD decoder.

The entropy values and their standard deviations are shown in Figure 13. One can see that all methods have a comparable diversity, where the SAT decoder seems to be slightly better while the sequential decoder is clearly performing worst.

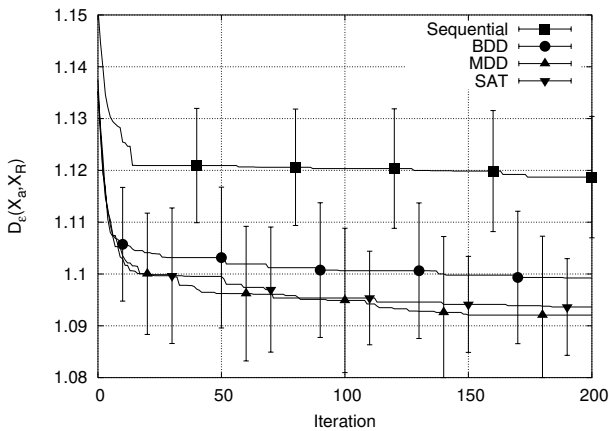


Figure 11: H.264 example: The mean ϵ -dominance and its standard deviation over the number of generations.

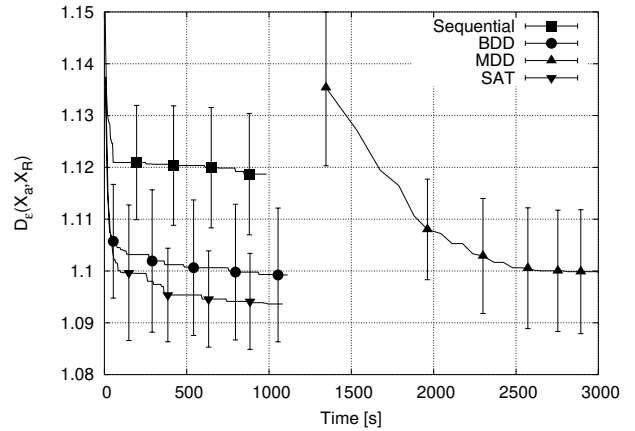


Figure 12: H.264 example: The mean ϵ -dominance and its standard deviation over the average optimization time.

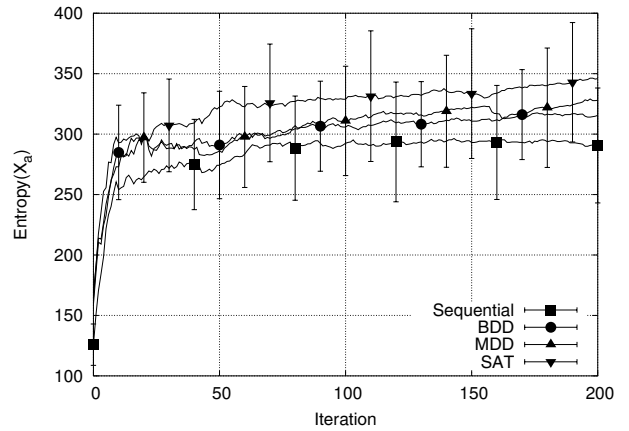


Figure 13: H.264 example: The mean entropy value and its standard deviation over the number of generations.

VII. Synthetic Benchmark Results

In this section, we present additional experimental results from applying the sequential decoder, the BDD decoder, and the SAT decoding technique to a set of synthetic benchmarks in order to test the performance in the presence of small and respectively large feasible regions X_f . We do not take the MDD decoder into account here, because the MDD could not be build for most of our test cases due to memory restrictions. The three objectives used during the optimization are again the technical properties of embedded systems, namely *implementation cost*, *power dissipation*, and *latency*. In the following, we provide quantitative results from the comparison between the sequential decoder, the BDD decoder, and the SAT decoder.

The experiments are performed as follows: A generator program is used to construct specification graphs. Due to different random values, the generated problem instances are similar in structure, but not equal. Each MCOP instance is

optimized using all three methods. After the optimization of each problem instance, the non-dominated solutions X_a found by these methods are combined in a single *reference set* X_R . This reference set is Pareto-filtered and is used to quantitatively assess the performance of the three decoding techniques.

A. Problem Instances and Parameters

We created nine different classes of MCOP instances (specification graphs). First we created three classes that differ in size. These are generated from following parameters:

(i) The number of available resources in the architecture graph is 25, 50, respectively 75.

(ii) The number of processes in the process graph g_p is either 50, 100, or 150.

(iii) Each process has 3...6, 4...8, respectively 5...9 random mapping edges.

(iv) The number of edges in the process graph is determined by a value that indicates the probability that two processes are connected by an edge. This probability value is 25% for every problem class. All these values are typical in system level design. The resulting MCOP instances are of approximate sizes $2^{104} - 2^{154}$, $2^{250} - 2^{350}$, and $2^{423} - 2^{550}$, respectively.

Then for each of these three problem classes we created three different subclasses with feasibility probabilities 15%, 25%, and 45%. This probability is used when the edges E_a of the architecture graph are created. For each possible mapping edge $m_i = (p_i, r_i), m_j = (p_j, r_j) \in E_m$ of two adjacent processes p_i and p_j , the resources r_i and r_j are connected with this probability to satisfy the data dependency. Smaller probability values result in less created edges, and less feasible solutions exist.

For each of these nine problem classes, we created 10 different problem instances. The optimization was run 10 times for each problem instance with all three decoding techniques.

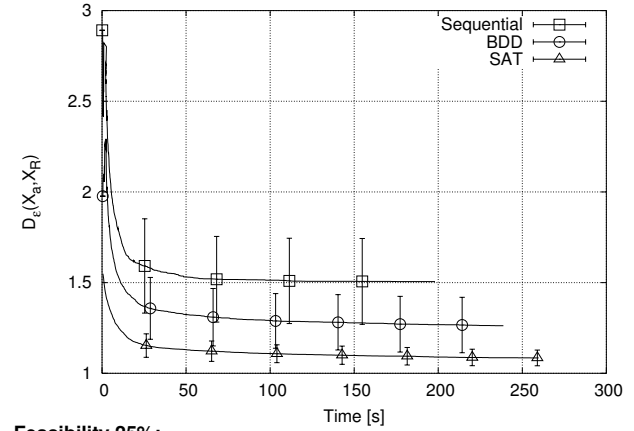
Especially for very hard problems, the sequential decoder only finds solutions in a reasonable time when all resources are allocated for the individuals in the initial population. So we changed the generation of the initial population for the sequential decoder to contain only this kind of individuals. For the SAT decoding technique and the BDD decoder we used the original algorithm which randomly allocates approximately half of the resources.

The parameters for the MOEA are: Population size 100, 25 children were created from 25 parents. The mutation rate was set to $|decision\ variables|^{-1}$ again.

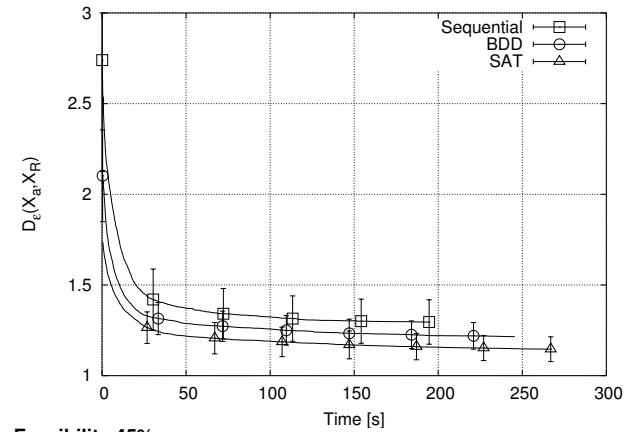
B. Quantitative Results

Each optimization was run for 1000 generations, and for every generation the average time to reach this generation is calculated and used to draw the Figures 14 - 16. An optimization run contributes to the average value computation only after it found at least one feasible solution.

Feasibility 15%:



Feasibility 25%:



Feasibility 45%:

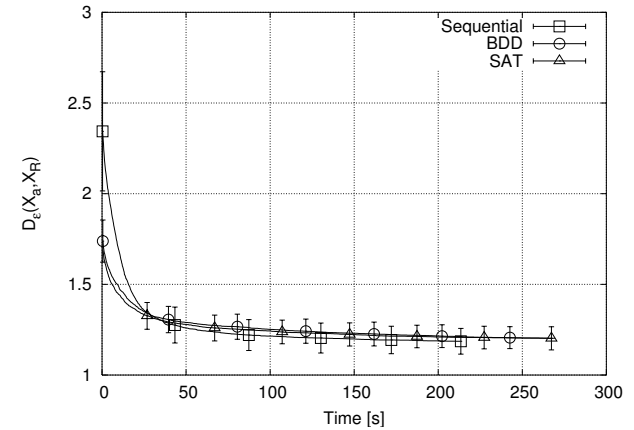


Figure 14: 50 processes: The mean ϵ -dominance and its standard deviation over the average optimization time (5 minutes).

Figure 14 shows that all three methods behave similar for problems with a small number of processes and many feasible solutions. But the less feasible solutions exist, the worse is the sequential decoder as compared to the SAT decoding technique, the BDD decoder is right in between.

In Figure 15, one can see, that the sequential decoder performs worse for problems with an increasing number of

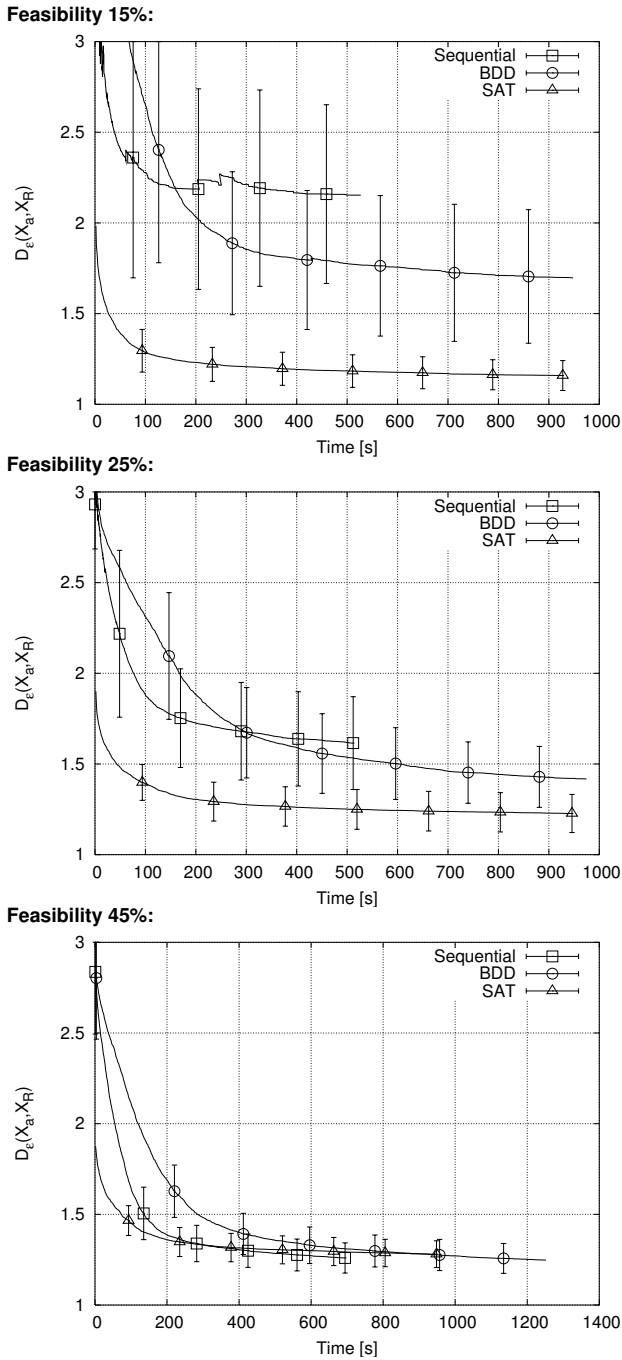


Figure. 15: 100 processes: The mean ϵ -dominance and its standard deviation over the average optimization time (16 minutes).

processes. The curve for the sequential decoder with a feasibility of 15% has some spikes. These spikes are due to optimization runs that find a first feasible solution quite late. The BDD decoder is always worse than the SAT decoding technique, and in the beginning even worse than the sequential decoder. But the harder the problem becomes, the earlier does the BDD decoder overtake the sequential one. As these

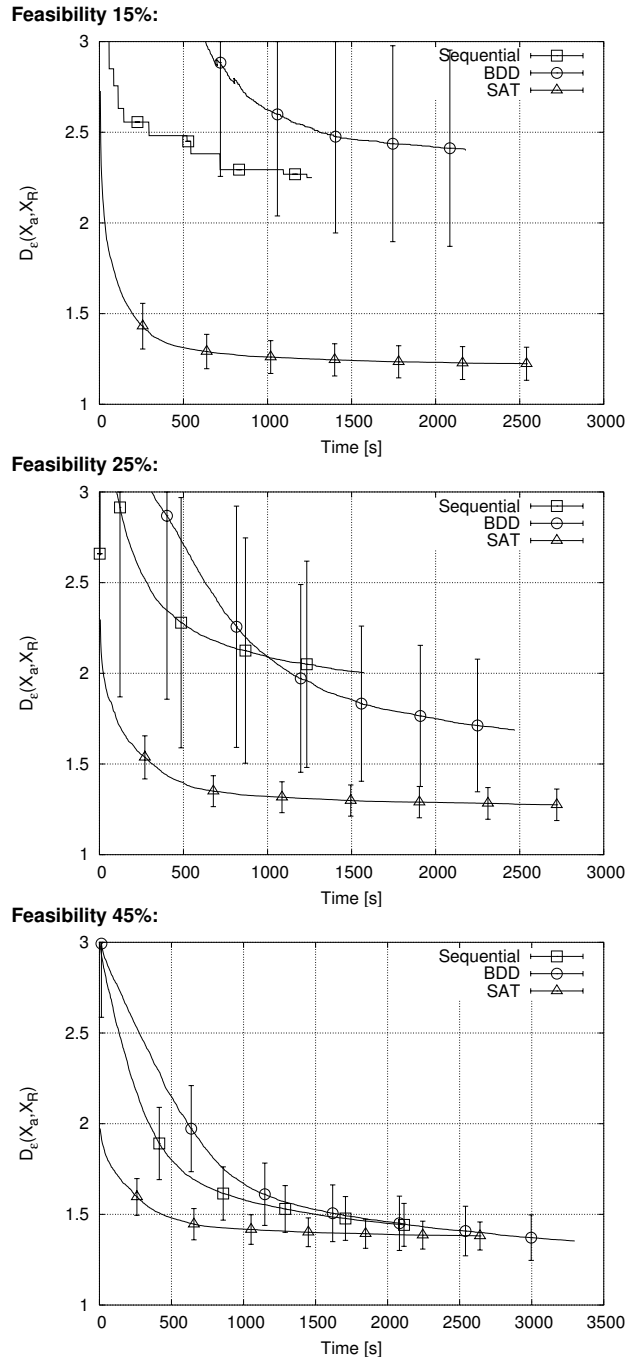


Figure. 16: 150 processes: The mean ϵ -dominance and its standard deviation over the average time.

feasible solutions are worse than the already found solutions of the other runs the mean ϵ -value becomes larger.

Figure 16 indicates that the SAT decoding technique still works good for a large number of processes. The steps in the curve of the sequential decoder with a feasibility of 15% indicate that only a single run out of 100 was able to find a feasible solution at all. The BDD decoder behaves similar to Figure 15.

Table 1: Entropy and its standard deviation values for the 9 test cases after 1000 generations.

Size	Feasibility	Sequential	BDD	SAT
50	15%	57.0607 ±44.7296	100.92 ±69.99	182.361 ±65.2609
	25%	137.399 ±74.0809	193.318 ±85.7739	236.229 ±84.3631
	45%	410.258 ±180.861	445.456 ±168.586	456.928 ±175.727
100	15%	32.8091 ±52.5102	120.17 ±84.7359	295.661 ±196.698
	25%	154.877 ±110.429	233.829 ±114.949	434.02 ±193.905
	45%	424.993 ±170.27	509.767 ±196.147	553.053 ±223.847
150	15%	1.99421 ±19.8421	45.9437 ±86.2474	300.613 ±134.402
	25%	160.92 ±113.338	236.785 ±132.53	429.89 ±199.177
	45%	391.455 ±153.231	475.613 ±187.923	594.463 ±242.524

If we compare the times for the different optimization runs, we can see that the runs of both symbolic decoding techniques take a longer time for bigger problems. This is due to the increased number of mapping edges, processes and resources, that increase the runtime for the evaluation of different quality functions. We can also see that the SAT-solver requires more time to compute 1000 generations than the sequential decoder. This has two reasons:

(1) The SAT decoding technique requires slightly more time to compute a binding for each individual (but one has to keep in mind that it always finds a valid binding by adjusting the resource allocation, if such exists).

(2) As the SAT decoding technique finds more feasible solutions in every generation, the quality functions have to be executed for this increased number of individuals.

But even with these increased durations, the SAT decoding technique generates better solutions earlier for big problems that contain few feasible solutions. 150 processes is not the upper limit for the SAT decoding technique, internal test have shown that even 500 processes can be handled in acceptable time with regard to the MOEA.

Table 1 shows the average entropy values for the test cases and their standard deviations. We can see that both the SAT decoder as well as the BDD decoder produce results with a higher diversity than the sequential decoder. With these results, we can conclude (at least for our test cases) that using Symbolic Techniques can help a MOEA to find feasible solutions. Moreover, the produced results by symbolic decoding converge faster to the Pareto-optimal set and do this, with a high diversity.

VIII. Conclusions

In many Multi-objective Combinatorial Optimization Problems (MCOPs), already the construction of a single feasible

solution is an \mathcal{NP} -complete problem. In order to guide the search towards the feasible region, we propose the use of Symbolic Techniques (STs) in Multi-Objective Evolutionary Algorithms. We illustrate our approach on the example of automatic design space exploration. In automatic design space exploration, the goal is to optimally select hardware resources and bind processes of an application onto these resources such that the communication between processes is guaranteed. By polynomially reducing the problem of finding a feasible binding to the Boolean satisfiability problem, the application of STs can be established. We have shown how to integrate STs into the decoding phase of an individual. From our experimental results, we can conclude that using STs in MOEAs are particularly useful in the presence of complex search spaces containing only very few feasible solutions. Although we focused on automatic design space exploration of embedded systems, there is however the potential to generalize our results to other constrained MCOPs.

References

- [1] A. H. Aguirre, S. B. Rionda, C. A. C. Coello, G. L. Lizárraga, and E. M. Montes. Handling Constraints using Multiobjective Optimization Concepts, *International Journal for Numerical Methods in Engineering*, 59(15), pp. 1989–2017, Apr. 2004.
- [2] H. E. Aguirre and K. Tanaka. Effects of Elitism and Population Climbing on Multiobjective MNK-Landscapes, In *Proceedings 2004 IEEE Congress on Evolutionary Computation (CEC 2004)*, San Diego, USA, volume 1, pp. 449–456, June 2004.
- [3] H. E. Aguirre and K. Tanaka. Selection, Drift, Recombination, and Mutation in Multiobjective Evolutionary Algorithms on Scalable MNK-Landscapes, In C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, New York, pp. 355–369, Mar. 2005, Springer.
- [4] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms, In *Lecture Notes in Computer Science (LNCS)*, Faro, Portugal, volume 2632, pp. 494–508, Apr. 2003.
- [5] T. Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*, PhD thesis, Swiss Federal Institute of Technology Zurich, Nov. 1996.
- [6] T. Blickle, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms, In R. Gupta, editor, *Design Automation for Embedded Systems*, Kluwer Academic Publishers, Boston, 3, pp. 23–62, Jan. 1998.

- [7] P. A. N. Bosman and D. Thierens. The Balance Between Proximity and Diversity in Multiobjective Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation*, 7(2), pp. 174–188, Apr. 2003.
- [8] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, C-35(8), pp. 677–691, Aug. 1986.
- [9] <http://sourceforge.net/projects/buddy>.
- [10] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers, 2002.
- [11] M. Davis and H. Putnam. A computing procedure for quantification theory, *J. Assoc. Comput. Mach.*, 7, pp. 201–215, 1960.
- [12] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, Ltd., Chichester, New York, Weinheim, Brisbane, Singapore, Toronto, 2001.
- [13] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, Paris, France, pp. 849–858, Sept. 2000.
- [14] K. Deb, M. Mohan, and S. Mishra. Towards a Quick Computation of Well-Spread Pareto-Optimal Solutions. In C. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pp. 222–236. Springer, Apr. 2003.
- [15] R. P. Dick and N. K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10), pp. 920–935, Oct. 1998.
- [16] M. Ehrgott and X. Gandibleux. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization, *OR Spektrum*, pp. 425–460, 2000.
- [17] M. Eisenring, L. Thiele, and E. Zitzler. Conflicting Criteria in Embedded System Design, *IEEE Design & Test of Computers*, 17(2), pp. 51–59, June 2000.
- [18] R. Feldmann, C. Haubelt, B. Monien, and J. Teich. Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques, In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *Field-Programmable Logic and Applications*, *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, volume 2778, pp. 478–487, Sept. 2003, Springer.
- [19] X. Gandibleux and M. Ehrgott. 1984-2004 – 20 Years of Multiobjective Metaheuristics. But What About the Solution of Combinatorial Problems with Multiple Objectives? In C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, New York, pp. 33–46, Mar. 2005, Springer.
- [20] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.
- [21] S. Gunawan, A. Farhang-Mehr, and S. Azarm. Multi-Level Multi-Objective Genetic Algorithm Using Entropy to Preserve Diversity, In *Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pp. 148–161, Apr. 2003.
- [22] C. Haubelt. *Automatic Model-Based Design Space Exploration for Embedded Systems – A System Level Approach*, PhD thesis, University of Erlangen-Nuremberg, Germany, Berlin, July 2005.
- [23] C. Haubelt, S. Mostaghim, F. Slomka, J. Teich, and A. Tyagi. Hierarchical Synthesis of Embedded Systems Using Evolutionary Algorithms. In R. Drechsler and N. Drechsler, editors, *Evolutionary Algorithms for Embedded System Design, Genetic Algorithms and Evolutionary Computation (GENA)*, Kluwer Academic Publishers, Boston, Dordrecht, London, pp. 63–104, 2003.
- [24] H. Jin and M.-L. Wong. Adaptive Diversity Maintenance and Convergence Guarantee in Multiobjective Evolutionary Algorithms, In *The 2003 Congress on Evolutionary Computation (CEC03)*, Canberra, Australia, pp. 2498–2505, Dec. 2003.
- [25] T. Kam, T. Villa, and R. K. Brayton. Multi-valued Decision Diagrams: Theory and Applications. *Multiple Valued Logic*, 4(1–2), pp. 9–62, 1998.
- [26] V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems, In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, Galveston, U.S.A., pp. 28–40, Sept. 2004.
- [27] M. Laumanns. *Analysis and Applications of Evolutionary Multiobjective Optimization Algorithms*, PhD thesis, Eidgenössische Technische Hochschule Zürich, Aug. 2003.
- [28] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining Convergence and Diversity in Evolutionary

- Multi-Objective Optimization, *Evolutionary Computation*, 10(3), pp. 263–282, 2002.
- [29] W. Li. Finding Pareto-Optimal Set by Merging Attractors for a Bi-objective Traveling Salesmen Problem, In C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, New York, pp. 797–810, Mar. 2005, Springer.
- [30] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver, In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
- [31] S. Neema. System Level Synthesis of Adaptive Computing Systems, PhD thesis, Vanderbilt University, Nashville, Tennessee, May 2001.
- [32] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study, *Multiobjective Metaheuristics*, *Lecture Notes in Economics and Mathematical Systems*, 2004.
- [33] L. Paquete and T. Stützle. A Two-Phase Local Search for the Biobjective Traveling Salesman Problem, In *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization*, *Lecture Notes in Computer Science (LNCS)*, Faro, Portugal, volume 2632, pp. 479–493, Apr. 2003.
- [34] I. E. G. Richardson. *Overview of H.264*, vcodex, Oct. 2002. www.vcodex.com/h264.html.
- [35] T. Schlichter, C. Haubelt, F. Hannig, and J. Teich. Using Symbolic Feasibility Tests during Design Space Exploration of Heterogeneous Multi-Processor Systems, In *Proceedings of Application-specific Systems, Architectures and Processors (ASAP)*, Samos, Greece, July 2005.
- [36] C. Scholl, R. Drechsler, and B. Becker. Functional Simulation Using Binary Decision Diagrams, In *Proceedings of the 1997 IEEE/ACM Int. Conference on Computer-Aided Design*, San Jose, USA, pp. 8–12, Nov. 1997.
- [37] D. A. V. Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*, PhD thesis, Graduate School of Engineering, Air Force Institute of Technology, June 1999.
- [38] <http://vlsi.colorado.edu/~vis/>.
- [39] Z. Yan, L. Zhang, L. Kang, and G. Lin. A New MOEA for Multi-objective TSP and Its Convergence Property Analysis. In *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization*, *Lecture Notes in Computer Science (LNCS)*, Faro, Portugal, volume 2632, pp. 342–354, Apr. 2003.
- [40] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Eidgenössische Technische Hochschule Zürich, Nov. 1999.
- [41] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search, In X. Yao et al., editors, *Parallel Problem Solving from Nature (PPSN VIII)*, Berlin, Germany, Springer-Verlag, pp. 832–842, 2004.
- [42] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization, In *Evolutionary Methods for Design, Optimisation, and Control*, Barcelona, Spain, pp. 19–26, 2002.
- [43] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review, *IEEE Transactions on Evolutionary Computation*, 7(2), pp. 117–132, Apr. 2003.

Author Biographies



Christian Haubelt received his diploma degree in Electrical Engineering from the University of Paderborn, Germany, in 2001 and received his Ph.D. degree in Computer Science from the Friedrich-Alexander-University of Erlangen-Nuremberg, Germany, in 2005. Christian Haubelt leads the System-level Design Automation group in the Department of Hardware-Software-Co-Design at the University of Erlangen-Nuremberg. Dr. Haubelt serves as a reviewer for several well-known international conferences and journals. His special research interests focus on system-level design, design space exploration, and Multi-Objective Evolutionary Algorithms.



Thomas Schlichter received his diploma degree in Computer Engineering from the University of Mannheim,

Germany, in 2003. He is currently a Trainer at the Fraunhofer Institute for Integrated Circuits, Erlangen, Germany, and a Ph.D. candidate in the Department of Hardware-Software-Co-Design, University of Erlangen-Nuremberg, Germany. His research interests include formal verification, using symbolic methods for design space exploration of embedded systems, and automatic system synthesis.



Jürgen Teich received his diploma degree (with honours) in 1989 from the University of Kaiserslautern, Germany, and his PhD degree (summa cum laude) from the University of Saarland, Saarbrücken, Germany, in 1993. In 1994,

Dr. Teich joined the DSP design group of Prof. E. A. Lee and D.G. Messerschmitt in the Department of Electrical Engineering and Computer Sciences (EECS) at UC Berkeley where he was working in the Ptolemy project (PostDoc). From 1995–1998, he held a position at the Institute of Computer Engineering and Communications Networks Laboratory (TIK) at ETH Zürich, Switzerland. From 1998–2002, he was a Full professor in the Electrical Engineering and Information Technology Department of the University of Paderborn, holding a Chair in Computer Engineering. Since 2003, he has been a full professor in the Computer Science Institute of the Friedrich-Alexander University Erlangen-Nuremberg holding a Chair in Hardware-Software-Co-Design. Prof. Teich serves on numerous program committees of well-known conferences and workshops. He is a member of the IEEE and the author of a textbook on Co-Design edited by Springer in 1997. His research interests include massive parallelism, embedded systems, Co-Design, and computer architecture. In 2004, Prof. Teich was elected a reviewer of the German Science Foundation (DFG) for the area of Computer Architecture and Embedded Systems.