Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design

Cagkan Erbas, Selin Cerav-Erbas, Andy D. Pimentel

August 24, 2005

Abstract

Sesame is a software framework which aims at developing a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. Since Sesame recognizes separate application and architecture models within a single system simulation, it needs an explicit mapping step to relate these models for co-simulation. The design trade-offs during the mapping stage, namely the processing time, power consumption, and the architecture cost are captured by a multiobjective nonlinear mixed integer program. This paper aims at investigating the performance of multiobjective evolutionary algorithms (MOEAs) on solving large instances of the mapping problem. With two comparative case studies, it is shown that MOEAs provide the designer with a highly accurate set of solutions in a reasonable amount of time. Additionally, analyses for different crossover types, mutation usage, and repair strategies for the purpose of constraints handling are carried out. Finally, a number of multiobjective optimization results are simulated for verification.

Multiprocessor System-on-Chip (SoC) design, design space exploration, multiobjective optimization, evolutionary algorithms, mixed integer programming.

1 Introduction

Modern embedded systems come with contradictory design constraints. On one hand, these systems target mass production and battery-based devices, and therefore should be cheap and power efficient. On the other hand, they still need to show high (sometimes real-time) performance, and often support multiple applications and standards which requires high programmability. This wide spectrum of design requirements leads to complex System-on-Chip (SoC) architectures, consisting of several types of processors from fully programmable microprocessors to configurable processing cores and customized

hardware components. The ensuant high complexity of embedded systems design has led to a new design paradigm, known as the *system-level design* [26], which has the following two important ingredients.

- Using a platform architecture that is shared among multiple applications, rather than designing one for each of them.
- Starting modeling and exploration with abstract executable components, and gradually lowering the abstraction level by inserting more implementation details in every step, with the intention of reaching an optimal SoC architecture.

In system-level design, early exploration of the design space plays a crucial role as it allows evaluate of different architecture alternatives without the burden of low level primitives. In terms of design time it would otherwise be impossible to evaluate several alternatives, if one started at a lower level which requires the synthesis and verification of different parts of the design. The models at the system-level normally capture the behavior of the application, characteristics of the architecture, and the various relations between the application and the architecture, such as the allocation (which components of the platform architecture are used), the binding (mapping of application processes onto architecture resources), or the scheduling (execution order of processes). The analytical and simulation models synthesized at the system-level can provide reasonable estimations of performance [4], power consumption [38], or cost of the design [20], while minimizing the requirements in terms of modeling effort and simulation time that is needed in the early design stages.

The Sesame framework[1] [11], [31], which we develop within the context of the Artemis project [32], provides methods and tools for the efficient design space exploration of heterogeneous embedded multimedia systems. Using Sesame, a designer can model embedded applications and SoC architectures at the system-level, map the former onto the latter using evolutionary optimizers which consider multiple design objectives simultaneously, and perform application-architecture co-simulations for rapid performance evaluations. Based on these evaluations, the designer can further refine (parts of) the design, experiment with different hardware/software partitionings, perform co-simulations at multiple levels of abstraction, or mixed level co-simulations where architecture model components operate at different levels of abstraction. To achieve this flexibility, the Sesame environment recognizes separate application and architecture models within a single system simulation. The application model defines the functional behavior of an application, including both computation and communication behaviors. The

[1] `http://sesamesim.sourceforge.net`

architecture model defines architecture resources and captures their performance constraints. An explicit mapping step maps an application model onto an architecture model for co-simulation.

Until recently, the mapping step in Sesame was assumed to be performed by an experienced designer, intuitively. However, this assumption was increasingly becoming inappropriate for efficient design space exploration. First of all, the Sesame environment targets exploration at an early design stage where the design space is very large. At this stage, it is very hard to make critical decisions such as *mapping* without using any analytical method or a design tool, since these decisions seriously affect the rest of the design process, and in turn, the success of the final design. Besides, modern embedded systems are already quite complicated, generally having a heterogeneous combination of hardware and software parts possibly with dynamic behavior. It is also very likely that these embedded systems will become even more complex in the near future, and intuitive mapping decisions will eventually become obsolete for future designs. Moreover, coping with the design constraints of embedded systems, there exist multiple criteria to consider, like the processing times, power consumption and cost of the architecture, all of which further complicate the mapping decision.

In Sesame, these issues are captured by means of a multiobjective combinatorial optimization problem [17]. Due to its large size and nonlinear nature, it is realized that the integration of a fast and accurate optimizer is of crucial importance for this problem. The primary aim of the multiobjective optimization process is to provide the designer with a set of tradable solutions, rather than a single optimal point. The evolutionary algorithms (EAs), in general, seem to be the best choice for attacking such problems, as they evolve over a population rather than a single solution. For this reason, numerous multiobjective evolutionary algorithms (MOEAs) have been proposed in the literature. The earlier MOEAs such as VEGA [35], MOGA [18], and NSGA [37] have been followed by the elitist versions, e.g., NSGA-II [14] and SPEA2 [47]. More recent work has focussed on the possible performance improvements by incorporating sophisticated strategies into MOEAs. For example, Jensen has employed advanced data structures to improve the runtime complexity of some popular MOEAs (e.g. NSGA-II) [24], while Yen et al. have proposed an approach based on the usage of dynamic populations [44]. In another recent work [30], the idea of transforming a high-dimentional multiobjective problem into a biobjective optimization problem is exploited within an MOEA.

In this article, we extend the work of [17] with the following new contributions:

- We employ two state-of-the-art MOEAs [14], [47] in two case studies from system-on-chip (SoC) design, and report performance results. Previously, these MOEAs have mostly been tested on simple and well-

known mathematical functions, but detailed performance results on real life engineering problems from different domains are very rare if any.

- A mathematical model is developed to exploit the multiprocessor mapping problem under multiple objectives.
- In order to determine the accuracy of the MOEAs, the mathematical model is first linearized and then solved by using an exact method, namely the lexicographic weighted Tchebycheff method.
- We perform two case studies in which we demonstrate *i*) the successful application of MOEAs to SoC design, especially in the early stages where the design space is very large, *ii*) the quantitative performance analysis of two state-of-the-art MOEAs examined in conjunction with an exact approach with respect to multiple criteria (e.g., accuracy, coverage of design space), and *iii*) the verification of multiobjective optimization results by further investigating a number of tradable solutions by means of simulation.
- In addition, we perform comparative experiments on variation operators and report performance results for different crossover types and mutation usage. More specifically, we analyze the consequences of using one-point, two-point and uniform crossover operators on MOEA convergence and exploration of the search space. Besides, we also show that mutation still remains as a vital operator in multiobjective search to achieve good exploration. Hence, the MOEAs stay in accordance with the standard EAs in this respect.
- We define three new metrics which will allow us to compare different aspects of MOEAs.
- We examine the performance consequences of using different fitness assignment schemes (finer-grained and computationally more expensive vs. more coarse-grained and computationally less expensive) in MOEAs.
- We study the outcome of using three different repair algorithms in constraint handling and compare them with respect to multiple criteria such as convergence and coverage of search space.

The rest of the paper is organized as follows. Section 2 discusses related work. Problem and model definitions and constraint linearizations are described in Section 3. Section 4 consists of four parts discussing the preliminaries for multiobjective optimization, the lexicographic weighted Tchebycheff method, the different attributes of multiobjective evolutionary algorithms and the repair algorithm, and the metrics for comparing

nondominated sets. In Section 5, two case studies are performed, comparative performance analysis of MOEAs are given, followed by some simulation results. The last section presents concluding remarks.

2 Related Work

In the domain of embedded systems and hardware/software codesign, several studies have been performed for *system-level synthesis* [7], [15], [43] and *platform configuration* [22], [21], [3], [42]. The former means the problem of optimally mapping a task-level specification onto a heterogeneous hardware/software architecture, while the latter includes tuning the platform architecture parameters and exploring its configuration space.

Blickle et al. [7] partition the synthesis problem into two steps: the selection of the architecture (allocation), and the mapping of the algorithm onto the selected architecture in space (binding) and time (scheduling). In their framework, they only consider cost and speed of the architecture, power consumption is ignored. To cope with infeasibility, they use penalty functions which reduce the number of infeasible individuals to an acceptable degree. In [43], a similar synthesis approach is applied to evaluate the design tradeoffs in packet processor architectures. But additionally, this model includes a real-time calculus to reason about packet streams and their processing.

In the MOGAC framework [15], starting from a task graph specification, the synthesis problem is solved for three objectives: cost, speed and power consumption of the target architecture. To accomplish this, an adaptive genetic algorithm which can escape local minima is utilized. However, this framework lacks the management of possible infeasibility as it treats all nondominated solutions equally even if they violate hard constraints. No repair algorithm is used in any stage of the search process, the invalid individuals are just removed at the end of evolution.

In [22], the configuration space of a parameterized system-on-chip (SoC) architecture is optimized with respect to a certain application mapped onto that architecture. The exploration takes into account power/performance trade-offs and takes advantage of parameter dependencies to guide the search. The configuration space is first clustered by means of a dependency graph, and each cluster is searched exhaustively for local Pareto-optimal solutions. In the second step, the clusters are merged iteratively until a single cluster remains. The Pareto-optimal configurations within this last cluster form the global Pareto-optimal solutions. In [21], the exploration framework of [22] is used in combination with a simulation framework. The simulation models of SoC components (e.g. processors, memories, interconnect busses) are used to capture dynamic information which is essential for the computation of power and performance metrics. More recently, Ascia et al. [3] have also applied a genetic algorithm to solve the same problem.

The work in [42] presents an exploration algorithm for parameterized memory architectures. The inputs to the exploration algorithm are timing and energy constraints obtained from the application tasks and the memory architecture specifications. The goal is to identify the system time/energy trade-off, when each task data member is assigned a target memory component. Exact and heuristic algorithms are given for solving different instances of the problem. However, only one type of heuristic (based on a branch and bound algorithm) is used, and no comparison with other heuristics is given.

In the Sesame framework, we do not target the problem of system synthesis. Therefore, a schedule is not constructed at the end of the design process. Our aim is to develop a methodology which allows for evaluating a large design space and provides us with a number of approximated Pareto-optimal solutions. These solutions are then input to our simulation framework for further evaluation. After simulation, figures about system-level trade-offs (e.g. utilization of components, data throughput, communication media contention) are provided to the designer. Thus, our goal is efficient design space exploration in terms of simulation. In addition, our framework also differs from the mentioned frameworks in the sense that it uses process networks for algorithm specification rather than task graphs.

Most of the aforementioned system-level synthesis/exploration and platform configuration frameworks have relied on evolutionary search techniques. Besides these studies, evolutionary algorithms are utilized at many abstraction levels of electronic systems design, such as in analog integrated circuit synthesis [2] and in the design of digital signal processing (DSP) systems [8] and evolvable hardware [19].

3 Problem and Model Definition

In the Sesame framework, applications are modeled using the Kahn Process Network (KPN) [25] model of computation in which parallel processes – implemented in a high level language – communicate with each other via unbounded FIFO channels. The workload of an application is captured by instrumenting the code of each Kahn process with annotations. By executing the application model, each process generates its own trace of application events.

The architecture models in Sesame, simulate the performance consequences of the application events generated by an application model. They solely account for performance constraints and only model timing behavior, since the functional behavior is already captured in the application model. An architecture model is constructed from generic building blocks provided by a library which contains template models for processing cores and various types of memory.

Since Sesame makes a distinction between application and architecture

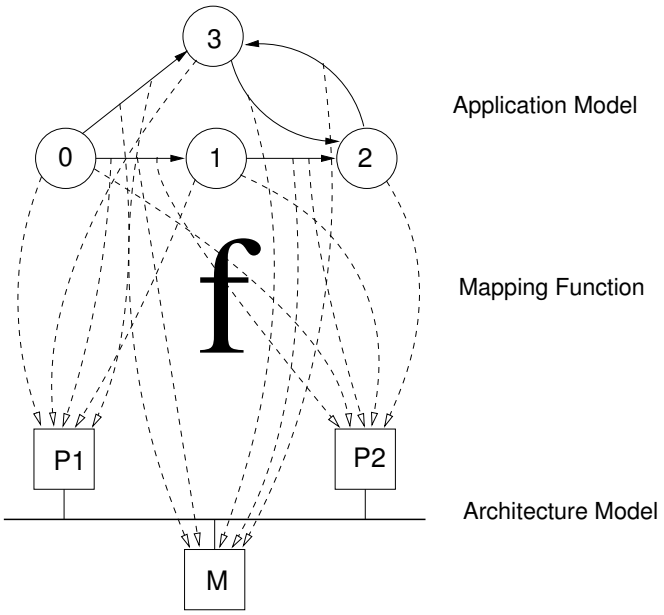


Figure 1: The mapping problem on a simple example. The mapping function has to consider multiple conflicting design objectives and should identify the set of Pareto-optimal mappings.

models, it needs an explicit mapping step to relate these models for co-simulation. In this step, the designer decides for each application process and FIFO channel a destination architecture model component to simulate its workload. Thus, this step is one of the most important stages in the design process, since the final success of the design is highly dependent on these mapping choices. In Figure 1, we illustrate this mapping step on a very simple example. In this example, the application model consists of four Kahn processes and five FIFO channels. The architecture model contains two processors and one shared memory. To decide on an optimum mapping, there exist multiple criteria to consider: maximum processing time in the system, power consumption and the total cost. This section aims at defining a mapping function, shown with **f** in Figure 1, to supply the designer with a set of best alternative mappings under the mentioned system criteria.

3.1 Application Modeling

The application models in Sesame are represented by a graph $KPN = (V_K, E_K)$ where the set V_K and E_K refer to the Kahn nodes and the directed FIFO channels between these nodes, respectively. For each node $a \in V_K$, we define $B_a \subseteq E_K$ to be the set of FIFO channels connected to node a, $B_a = \{b_{a1}, \ldots, b_{an}\}$. For each Kahn node, we define a computation requirement, shown with α_a, representing the computational workload im-

posed by that Kahn node onto a particular component in the architecture model. The communication requirement of a Kahn node is not defined explicitly, rather it is derived from the channels attached to it. We have chosen this type of definition for the following reason: if the Kahn node and one of its channels are mapped onto the same architecture component, the communication overhead experienced by the Kahn node due to that specific channel is simply neglected. Only its channels that are mapped onto different architecture components are taken into account. So our model neglects internal communications and only considers external communications. Formally, we denote the communication requirement of the channel b with β_b. To include memory latencies into our model, we require that mapping a channel onto a specific memory asks computation tasks from the memory. To express this, we define the computational requirement of the channel b from the memory as α_b. The formulation of our model ensures that the parameters β_b and α_b are only taken into account when the channel b is mapped onto an external memory.

3.2 Architecture Modeling

Similarly to the application model, the architecture model is also represented by a graph $ARC = (V_A, E_A)$ where the sets V_A and E_A denote the architecture components and the connections between the architecture components, respectively. In our model, the set of architecture components consists of two disjoint subsets: the set of processors (P) and the set of memories (M), $V_A = P \cup M$ and $P \cap M = \emptyset$. For each processor $p \in P$, the set $M_p = \{m_{p1}, \ldots, m_{pj}\}$ represents the memories which are reachable from the processor p. We define processing capacities for both the processors and the memories as c_p and c_m, respectively. These parameters are set such that they reflect processing capabilities for processors, and memory access latencies for memories.

One of the key considerations in the design of embedded systems is the power consumption. In our model, we consider two types of power consumption for the processors. We represent the power dissipation of the processor p during *execution* with w_{pe}, while w_{pc} represents its power dissipation during *communication* with the external memories. For the memories, we only define w_{me}, the power dissipation during *execution*. For both processors and memories, we neglect the power dissipation during idle times. In our model, we also consider the financial costs associated with the architecture model components. Using an architecture component in the system adds a fixed amount to the total cost. We represent the fixed costs as u_p and u_m for the processors and the memories, respectively.

3.3 The Mapping Problem

We have the following decision variables in the model: $x_{ap} = 1$ if Kahn node a is mapped onto processor p, $x_{bm} = 1$ if channel b is mapped onto memory m, $x_{bp} = 1$ if channel b is mapped onto processor p, $y_p = 1$ if processor p is used in the system, $y_m = 1$ if memory m is used in the system. All the decision variables get a value of zero in all other cases. The constraints in the model are:

- Each Kahn node has to be mapped onto a single processor,

$$\sum_{p \in P} x_{ap} = 1 \qquad \forall a \in V_K. \tag{1}$$

- Each channel in the application model has to be mapped onto a processor or a memory,

$$\sum_{p \in P} x_{bp} + \sum_{m \in M} x_{bm} = 1 \qquad \forall b \in E_K. \tag{2}$$

- If two communicating Kahn nodes are mapped onto the same processor, then the communication channel(s) between these nodes have to be mapped onto the same processor.

$$x_{a_i p} x_{a_j p} = x_{bp} \qquad \forall b = (a_i, a_j) \in E_K. \tag{3}$$

- The constraint given below ensures that when two communicating Kahn nodes are mapped onto two separate processors, the channel(s) between these nodes are to be mapped onto an external memory.

$$x_{a_i p_k} x_{a_j p_l} \leq \sum_{m \in M_{p_k} \cap M_{p_l}} x_{bm} \qquad \begin{array}{l} \forall a_i, a_j \in V_K, \\ \forall b \in B_{a_i} \cap B_{a_j}, \\ \forall p_k \neq p_l \in P. \end{array} \tag{4}$$

- The following constraints are used to settle the values of y_p and y_m's in the model. We multiply the right-hand side of the first equation series by the total number of Kahn nodes and FIFO channels, since this gives an upper bound on the number of application model components that can be mapped to any processor. Similar logic is applied to the equations related with memory.

$$\sum_{a \in V_K} x_{ap} + \sum_{b \in E_K} x_{bp} \leq (\mid V_K \mid + \mid E_K \mid) y_p \quad \forall p \in P, \tag{5}$$

$$\sum_{b \in E_K} x_{bm} \leq \mid E_K \mid y_m \quad \forall m \in M. \tag{6}$$

Three conflicting objective functions exist in the optimization problem:

- The first objective function tries to minimize the maximum processing time in the system. For each processor and memory, f_p and f_m represent the total processing time of the processor and memory, respectively. We also show the total time spent by the processor for the execution events as f_p^e and for the communication events as f_p^c.

$$f_p = f_p^e + f_p^c, \tag{7}$$

$$f_p^e = \frac{1}{c_p} \sum_{a \in V_K} \alpha_a x_{ap}, \tag{8}$$

$$f_p^c = \frac{1}{c_p} \sum_{a \in V_K} x_{ap} \sum_{b \in B_a, m \in M_p} \beta_b x_{bm}, \tag{9}$$

$$f_m = \frac{1}{c_m} \sum_{b \in E_K} \alpha_b x_{bm}. \tag{10}$$

  So the objective function is expressed as

$$\min \max_{i \in V_A} f_i. \tag{11}$$

- The second objective function tries to minimize the power consumption of the whole system. Similarly, g_p and g_m denote the total power consumption of processor p and memory m.

$$g_p = f_p^e w_{pe} + f_p^c w_{pc}, \tag{12}$$

$$g_m = f_m w_{me}. \tag{13}$$

$$\min \sum_{i \in V_A} g_i. \tag{14}$$

- The last objective function aims at minimizing the total cost of the architecture model.

$$\min \sum_{p \in P} u_p y_p + \sum_{m \in M} u_m y_m. \tag{15}$$

Table 1: Table of symbols for the MMPN problem.

Application parameters	
V_K	set of Kahn nodes
E_K	set of channels
B_a	set of channels connected to node a
α_a	computation requirement of node a
β_b	communication requirement of channel b
α_b	computation requirement of channel b
Architecture parameters	
V_A	set of architecture components
E_A	connectivity set of architecture components
P	set of processors
M	set of memories
c_p	processing capacity of processor p
c_m	processing capacity of memory m
w_{pe}	power dissipation of p during execution
w_{pc}	power dissipation of p during communication
w_{me}	power dissipation of m during execution
u_p	fixed cost of p
u_m	fixed cost of m
Binary decision variables	
x_{ap}	whether a is mapped onto p
x_{bm}	whether b is mapped onto m
x_{bp}	whether b is mapped onto p
y_p	whether p is used
y_m	whether m is used
Functions	
f_i	total processing time of component i
g_i	total power dissipation of component i

Definition 1 *(MMPN problem) Multiprocessor Mappings of Process Networks (MMPN) Problem is the following multiobjective integer optimization problem:*

$$\begin{aligned} \min \quad & \mathbf{f} = (\max_{i \in V_A} f_i, \sum_{i \in V_A} g_i, \sum_{p \in P} u_p y_p + \sum_{m \in M} u_m y_m) \qquad (16) \\ \textit{s.t.} \quad & \textit{(1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (12), (13),} \end{aligned}$$

$$\begin{aligned} x_{ap}, x_{bm}, x_{bp}, y_p, y_m \in \{0,1\} \qquad & \forall a \in V_K, \ \forall b \in E_K, \\ & \forall m \in M, \ \forall p \in P. \qquad (17) \end{aligned}$$

For the sake of convenience Table 1 presents the set of mathematical symbols for the MMPN problem.

3.4 Constraint Linearizations

In Section 5, we will solve an instance of the MMPN problem using both exact and heuristic methods. Because the problem has some nonlinear constraints, one has to linearize them before using a mathematical optimizer. Next we show how this is done.

(3) can be linearized by replacing it with these three constraints:

$$x_{bp} \geq x_{a_ip} + x_{a_jp} - 1, \tag{18}$$
$$x_{bp} \leq x_{a_ip}, \tag{19}$$
$$x_{bp} \leq x_{a_jp}. \tag{20}$$

Similarly, (4) can be linearized by introducing a new binary variable $x_{a_ip_ka_jp_l} = x_{a_ip_k}x_{a_jp_l}$ and adding the constraints:

$$x_{a_ip_ka_jp_l} \geq x_{a_ip_k} + x_{a_jp_l} - 1, \tag{21}$$
$$x_{a_ip_ka_jp_l} \leq x_{a_ip_k}, \tag{22}$$
$$x_{a_ip_ka_jp_l} \leq x_{a_jp_l}. \tag{23}$$

Finally, (9) can be linearized by introducing the binary variable $x_{apbm} = x_{ap}x_{bm}$ and adding the constraints:

$$x_{apbm} \geq x_{ap} + x_{bm} - 1, \tag{24}$$
$$x_{apbm} \leq x_{ap}, \tag{25}$$
$$x_{apbm} \leq x_{bm}. \tag{26}$$

4 Multiobjective Optimization

4.1 Preliminaries

Definition 2 *A general multiobjective optimization problem with k decision variables and n objective functions is defined as:*

$$\begin{aligned} &\textit{minimize} && \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) \\ &\textit{subject to} && \mathbf{x} \in X_f \end{aligned}$$

where $\mathbf{x}$ *represents a solution and* $X_f \subseteq X$ *is the set of feasible solutions. The objective function vector* $\mathbf{f}(\mathbf{x})$ *maps a decision vector* $\mathbf{x} = (x_1, \ldots, x_k)$ *in decision space (X) to an objective vector* $\mathbf{z} = (z_1, \ldots, z_n)$ *in objective space (Z).*

Definition 3 *(Dominance relations) Given two objective vectors* $\mathbf{z}^1$ *and* $\mathbf{z}^2$*, we say*

- $\mathbf{z}^1 \ll \mathbf{z}^2$ *(*$\mathbf{z}^1$ *strictly dominates* $\mathbf{z}^2$*) iff* $z_i^1 < z_i^2$, $\forall i \in \{1, \ldots, n\}$.

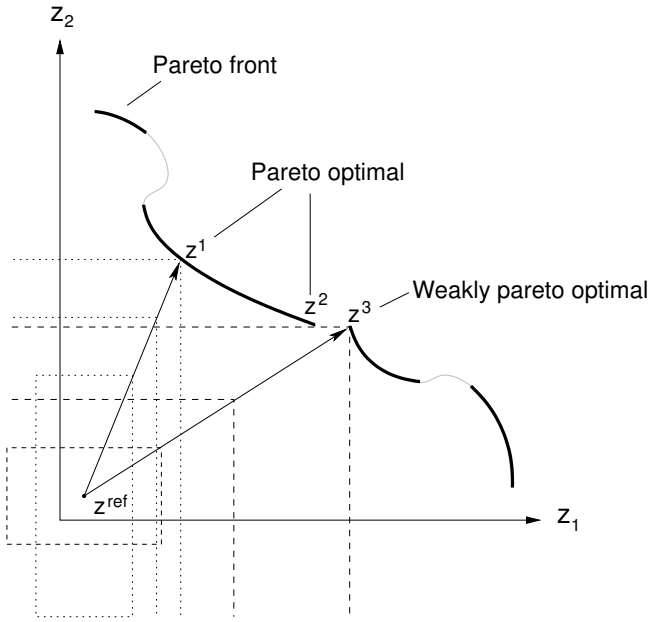


Figure 2: The lexicographic weighted Tchebycheff method can be considered as drawing probing rays emanating from $\mathbf{z}^{\text{ref}}$ towards the Pareto front. The points equidistant from $\mathbf{z}^{\text{ref}}$ form a family of rectangles centered at $\mathbf{z}^{\text{ref}}$.

- $\mathbf{z}^1 < \mathbf{z}^2$ *($\mathbf{z}^1$ dominates $\mathbf{z}^2$) iff* $z_i^1 \leq z_i^2$ *and* $\mathbf{z^1} \neq \mathbf{z^2}$, $\forall i \in \{1, \ldots, n\}$.
- $\mathbf{z}^1 \leq \mathbf{z}^2$ *($\mathbf{z}^1$ weakly dominates $\mathbf{z}^2$) iff* $z_i^1 \leq z_i^2$, $\forall i \in \{1, \ldots, n\}$.
- $\mathbf{z}^1 \sim \mathbf{z}^2$ *($\mathbf{z}^1$ is incomparable with $\mathbf{z}^2$) iff* $\exists i \neq j \in \{1, \ldots, n\}$ *such that* $z_i^1 < z_i^2$ *and* $z_j^2 < z_j^1$.

Definition 4 *A decision vector $\mathbf{x} \in A \subseteq X_f$ is said to be nondominated in set A iff $\nexists \mathbf{a} \in A$ such that $\mathbf{f}(\mathbf{a}) < \mathbf{f}(\mathbf{x})$.*

Definition 5 *(Nondominated set and front) The set containing only nondominated decision vectors $X_{nd} \subseteq X_f$ is called nondominated set. Its image on the objective space, $Z_{nd} = \{\mathbf{z} : \mathbf{z} = \mathbf{f}(\mathbf{x}), \mathbf{x} \in X_{nd}\}$ is called nondominated front.*

Definition 6 *(Pareto set and front) The set $X_{par} = \{\mathbf{x} : \mathbf{x}$ is nondominated in $X_f\}$ is called Pareto set. Its image on the objective space $Z_{eff} = \{\mathbf{z} : \mathbf{z} = \mathbf{f}(\mathbf{x}), \mathbf{x} \in X_{par}\}$ is called Efficient set or equivalently Pareto front.*

Definition 7 *(Euclidean distance) The Euclidean distance between two vectors (of dimension n) $\mathbf{z}^1$ and $\mathbf{z}^2$ is defined as $\|\mathbf{z}^1 - \mathbf{z}^2\| = \sqrt{\sum_{i=1}^{n}(z_i^1 - z_i^2)^2}$.*

After these ground-laying definitions, in the rest of this section we first briefly explain an exact method for solving multiobjective optimization problems, namely the lexicographic weighted Tchebycheff method which was introduced by Steuer and Choo [40]. Then we will move to heuristic methods and introduce two state-of-the-art highly competitive Multiobjective Evolutionary Algorithms (MOEAs) [14], [47]. The discussion on MOEAs is performed within the context of the MMPN problem, especially when it comes to those problem specific parameters. We conclude this section by defining three new metrics for MOEA performance comparisons.

4.2 Lexicographic Weighted Tchebycheff Method

Definition 8 *(Weakly Pareto-optimal point) A solution* $\mathbf{x}^* \in X_f$ *is weakly Pareto-optimal if there is no* $\mathbf{x} \in X_{par}$ *such that* $\mathbf{f}(\mathbf{x}) \ll \mathbf{f}(\mathbf{x}^*)$.

The lexicographic weighted Tchebycheff method [40] works in two steps. In the first step, we take a reference vector in objective space with components

$$z_i^{\text{ref}} = \min\{f_i(\mathbf{x} \mid \mathbf{x} \in X_f)\} - \epsilon_i,$$

where ϵ_i are small positive values. Generally, it is common to set ϵ_i to the value which makes $z_i^{\text{ref}} = \lfloor \min\{f_i(\mathbf{x} \mid \mathbf{x} \in X_f)\} \rfloor$. In this step we solve

$$\begin{aligned} \min \quad & \alpha \\ \text{subject to} \quad & \alpha \geq \lambda_i |f_i(\mathbf{x}) - z_i^{\text{ref}}|, \\ & \sum_{i=1}^{n} \lambda_i = 1,\ 0 < \lambda_i < 1 \text{ and } \mathbf{x} \in X_f, \end{aligned} \tag{27}$$

which guarantees weak Pareto optimality [16]. We denote the set of solutions found in this first step with X_w. In the second step, solutions in X_w are checked for Pareto optimality using

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} f_i(\mathbf{x}) \\ & \mathbf{x} \in X_w \end{aligned} \tag{28}$$

and all weakly Pareto-optimal points are eliminated. After this step, the retained Pareto-optimal solutions form X_{par}. In Figure 2, we illustrate this graphically. The first step in the lexicographic weighted Tchebycheff method can be considered as drawing probing rays emanating from $\mathbf{z}^{\text{ref}}$ towards the Pareto front. The points equidistant from $\mathbf{z}^{\text{ref}}$ form a family of rectangles centered at $\mathbf{z}^{\text{ref}}$. Moreover, the vertices of these rectangles lie in the probing ray in the domain of the problem [39]. The objective in (27) is optimized when the probing ray intersects the Pareto front. In this step, points $\mathbf{z}^1$, $\mathbf{z}^2$ and $\mathbf{z}^3$ can be located. In the second step, weakly Pareto-optimal $\mathbf{z}^3$ is eliminated.

Algorithm 1 A General Elitist Evolutionary Algorithm

input: N: Size of the population
T: Maximum number of generations.

output: Nondominated individuals in P_{t+1}.

step1. *Initialization:* Generate a random initial population P_0, and create an empty child set Q_0. $t \leftarrow 0$.

step2. *Fitness assignment:* $P_{t+1} \leftarrow P_t \cup Q_t$, and then calculate the fitness values of the individuals in P_{t+1}.

step3. *Truncation:* Reduce size of P_{t+1} by keeping best N individuals according to their fitness values.

step4. *Termination:* If $t = T$, output nondominated individuals in P_{t+1} and terminate.

step5. *Selection:* Select individuals from P_{t+1} for mating.

step6. *Variation:* Apply crossover and mutation operations to generate Q_{t+1}. $t \leftarrow t + 1$ and go to **step2**.

4.3 Multiobjective Evolutionary Algorithms (MOEAs)

Evolutionary algorithms have become very popular in multiobjective optimization, as they operate on a set of solutions. Over the years, many multiobjective evolutionary algorithms have been proposed [9], [10]. In this section, we study two state-of-the-art MOEAs: SPEA2 which was proposed by Zitzler et al. [47] and NSGA-II by Deb et al. [14]. Both algorithms are similar in the sense that they follow the main loop in Algorithm 1. To form the next generation, they employ a deterministic truncation by choosing N best individuals from a pool of current and offspring populations. In addition, they both employ binary tournament selection [5]. Nevertheless, the main difference lies in their fitness assignment schemes. Despite the fact that both MOEAs apply a lexicographic fitness assignment scheme, objectives of which are to give first priority to nondominance and second priority to diversity, SPEA2 does so by using a finer-grained and therefore a more computationally expensive approach than its rival NSGA-II. The interesting question here is whether this additional computation effort pays off when we look at the overall performance of SPEA2 and NSGA-II. This issue is investigated experimentally in Section 5.

The distinctive characteristic of SPEA2 and NSGA-II is that both algorithms employ *elitism*, that is to guarantee a strictly positive probability for selecting at least one nondominated individual as an operand for variation operators. In both MOEAs, the following procedure is carried out to introduce elitism: the offspring and current population are combined and subsequently the best N individuals in terms of nondominance and diversity are chosen to build the next generation. Unlike single optimization studies, elitism has attracted high attention from the researchers in multiobjective

optimization. Although it is still a very active research subject, elitism is believed to be an important ingredient in search with multiple objectives. For example in [28] and [46], experiments on continuous test functions show that elitism is beneficial, while in [45] similar results are also reported for two combinatorial (multiobjective 0/1 knapsack and travelling salesman) problems. Apart from these experimental studies, Rudolph has theoretically proven that an elitist MOEA can converge to the Pareto-front in finite number of iterations [34].

After the mentioned validatory studies, NSGA-II has been proposed as an elitist version of its predecessor NSGA. Besides elitism, NSGA-II has additional benefits over NSGA such as: *i*) a lower computational complexity, *ii*) a parameterless mechanism for maintaining diversity among nondominated solutions, *iii*) a deterministic selection algorithm to form the next generation by lexicographically sorting the combination of the current population and the offspring.

Similar to NSGA-II, SPEA2 is an improved successor of SPEA which was one of the first MOEAs with elitism. SPEA2 differs from SPEA in terms of *i*) a finer-grained fitness assignment mechanism, *ii*) a new density estimation technique for maintaining diversity, and *iii*) a new truncation method which prevents the loss of boundary solutions.

In the remainder of this section, we concentrate on problem-specific portions of MOEAs, and the discussion will be based on the MMPN problem, our focus of interest in this paper. The discussion is divided into three parts: individual encoding, constraint violations and variation operations. We conclude this section by defining three new metrics in the interest of comparing MOEAs under different criteria.

4.3.1 Individual Encoding

Each genotype consists of two main parts: a part for Kahn process nodes and a part for FIFO channels. Each gene in the chromosome has its own feasible set which is determined by the type of the gene and the constraints of the problem. For genes representing Kahn process nodes, only the set of processors in the architecture model form the feasible set, while for genes representing the FIFO channels, both the set of processors and the set of memories constitute the feasible set.

The constraints of the problem may include some limitations which should be considered in individual coding. For example, if there exists a dedicated architecture component for a specific Kahn process, then this architecture component has to be included only in the feasible set of this Kahn process.

In Figure 3, an example chromosome is given. The first three genes are those for Kahn process nodes, and the rest are those for FIFO channels. We have placed closely related genes together in order to maintain *locality*. The

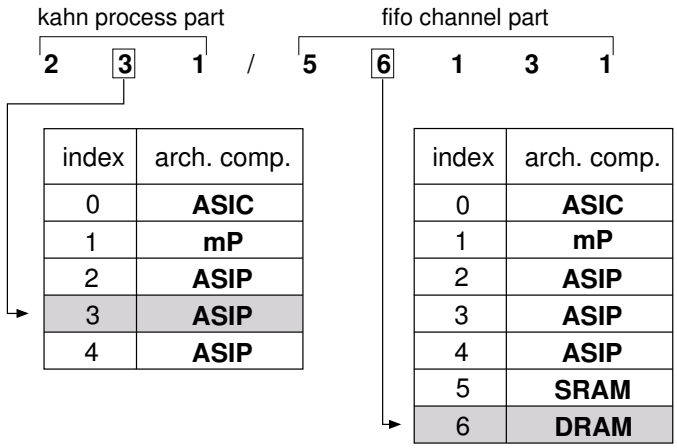


Figure 3: An example individual coding. The closely related genes are put together in order to preserve locality.

latter is vital for the success of an evolutionary algorithm [23], [15]. For this gene, the second Kahn process is mapped onto an Application Specific Instruction Processor (ASIP) while the second FIFO channel is mapped onto a DRAM. We also see that the feasible sets for these two genes are different.

4.3.2 Constraint Violations

We have developed a repair mechanism to deal with constraint violations. Due to randomness in MOEAs (in initialization, crossover and mutation steps), the constraints (1), (2), (3) and (4) are prone to violation. The repair mechanism given in Algorithm 2 first considers whether each Kahn process is mapped onto a processor from its feasible set, and if not, it repairs by randomly mapping the Kahn process to a feasible processor. After having finished processing the Kahn process genes, it proceeds along with the FIFO channel genes. For the latter, the repair algorithm simply checks for each FIFO channel whether the Kahn processes it is connected to are mapped onto the same processor. If this condition holds, then it ensures that the FIFO channel is also mapped onto that processor. If the condition does not hold, which means that the Kahn processes are mapped onto different processors (say, P_1 and P_2), it finds the set of memories reachable from both P_1 and P_2 (mathematically, $M_{P_1} \cap M_{P_2}$). Then it selects a memory from this set randomly and maps the FIFO channel onto that memory. However, it is interesting to note here that if $M_{P_1} \cap M_{P_2} = \emptyset$, then the problem itself may become infeasible[2]. Therefore, we exclude these architectures.

[2]Although, it is possible to repair by mapping both the FIFO channel and one of the Kahn processes onto the processor that the other Kahn process is mapped onto, this would require the individual to re-enter repair as it may cause additional infeasibilities for other

Algorithm 2 Individual Repair Algorithm

input: I (individual)
output: I (individual)

```
for all Kahn process genes do
  check if it is mapped onto a processor from its feasible set.
  if mapping is infeasible then
    repair: map on a random processor from its feasible set.
  end if
end for
for all FIFO channel genes do
  K1 ← source Kahn process of the FIFO channel.
  K2 ← sink Kahn process of the FIFO channel.
  P1 ← processor that K1 is mapped onto.
  P2 ← processor that K2 is mapped onto.
  if P1 = P2 then
    repair: map FIFO channel onto P1.
  else
    M ← a randomly chosen memory from M_P1 ∩ M_P2.
    repair: map FIFO channel on M.
  end if
end for
```

With respect to repair we have developed and tested three strategies. In the first (*no-repair*) strategy none of the individuals is repaired during any step, all are treated as valid individuals during the optimization process. Once the optimization is finished, repair is applied to the invalid individuals, and all nondominated solutions are output. Although this approach does not sound very promising as it neglects infeasibility, it is included here for two reasons: the first reason is that some researchers have applied this strategy to multiobjective combinatorial problems and reported positive results [15]; and the second reason is to see the performance gain/loss when constraint handling is taken into account. In the second strategy, which we call *moderate-repair*, at the end of each variation (step6 in Algorithm 1) all invalid individuals are repaired. This allows infeasible individuals to enter the mutation step. The latter may help to explore new feasible areas over unfeasible solutions. This is especially important for combinatorial problems in which the feasible region may not be connected. The last strategy we employ here is called *extensive-repair*, as it repairs all invalid individuals immediately after every variation step. Hence, all individuals entering mutation are feasible. The experimental results concerning the repair strategies are discussed in Section 5.

FIFO channels. In the worst case, this can be an infinite loop.

4.3.3 Variation Operations

As we have already mentioned, experiments in Section 5 should give us some feedback about *i*) whether the finer-grained computationally-expensive fitness assignment in SPEA2 pays off, and *ii*) the effect of using different repair schemes (no-repair, moderate-repair and extensive-repair strategies). Therefore, we have fixed other factors that may effect MOEA performance. We have used only one type of mutation and crossover operations in all standard runs. For the former, we have used independent bit mutation (each bit of an individual is mutated independently with respect to bit mutation probability), while for the latter standard one-point crossover (two parent chromosomes are cut at a random point and the sections after the cut point are swapped) is employed.

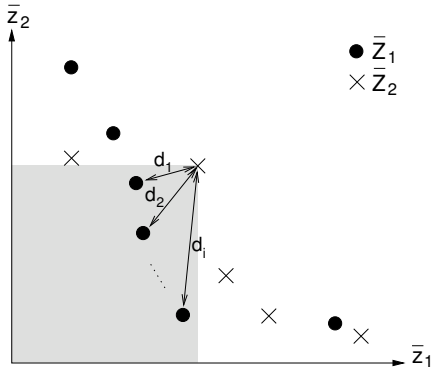
Many researchers have reported comparative performance results on different crossover types and mutation for traditional EAs solving single objective problems [23], [36], [41]. Therefore, it may well be interesting to perform similar comparative experiments with some variation operators in the multiobjective case. In this respect, we have performed additional experiments in Section 5.2 for the comparison of different crossover operators and the effect of mutation usage.

In our analysis with respect to crossover operators we have compared the performance of the one-point crossover with that of the two-point and uniform crossover operators. In two-point crossover the individual is considered as a ring formed by joining the ends together. The ring is cut at two random points forming two segments, and the two mating parents exchange one segment in order to create the children. One should note that the two-point crossover performs the same task as the one-point crossover by exchanging a single segment, however is more general. Uniform crossover is rather different from both one-point and two-point crossover; two parents are selected for reproducing two children, and for each bit position on the two children it is randomly decided which parent contributes its bit value to which child.

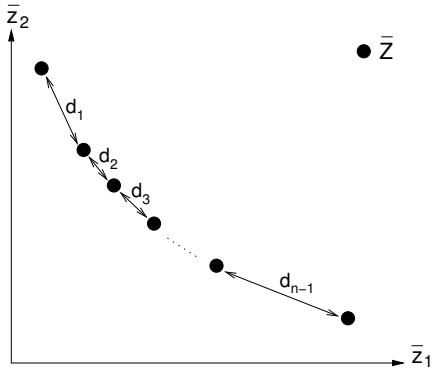
4.4 Metrics for Comparing Nondominated Sets

To properly evaluate and compare MOEA performances, one can identify three important criteria [13]:

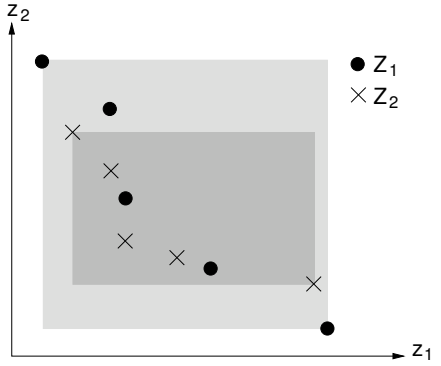
- **Accuracy.** The distance of the resulting nondominated set to the Pareto-optimal front should be minimal.
- **Uniformity.** The solutions should be well distributed (in most cases uniform).
- **Extent.** The nondominated solutions should cover a wide range for each objective function value.



(a) D-metric for accuracy.



(b) Δ-metric for uniformity.



(c) ∇-metric for extent.

Figure 4: Metrics for comparing nondominated sets. For the sake of simplicity, illustrations for the two dimensional case are given. However, the metrics are also valid for any higher dimension.

Unlike single objective optimization problems, where the single aim is to locate a global optimum without being trapped at local optima, multiobjective optimization requires multiple aims to be satisfied simultaneously. Besides the obvious accuracy criterion, that is locating a set of solutions being at minimal distant from the Pareto front, multiobjective optimizers also need to maintain a well distributed solution set (i.e. uniformity) for a more complete view of the trade-off curve and should catch boundary points (i.e. extent) for a better coverage of the objective space.

There has been some effort for measuring the performance assessments of MOEAs [46], [27]. Metrics, in general, can be classified as *i*) metrics evaluating only one nondominated set, *ii*) metrics comparing two nondominated sets, *iii*) metrics requiring knowledge of the Pareto-optimal set, and

iv) metrics measuring single or multiple assessment(s).

In the rest of this section, we propose one metric for each of the three identified criteria. Due to the fact that every objective function scales independently, one should map the limits of the objective function values to a unique interval, before doing any arithmetic operation. Therefore, we first present normalization of vectors, before defining the performance metrics.

4.4.1 Normalization of Vectors in Objective Space

To make calculations scale independent, we normalize vectors before doing any arithmetic. At the end of normalization, each coordinate of the objective space is scaled such that all points get a value in the range $[0, 1]$ for all objective values. Assume we have p nondominated sets, $Z_1 \dots Z_p$. First we form $Z = Z_1 \cup \dots \cup Z_p$. Then we calculate $f_i^{min} = \min\{f_i(\mathbf{x}), \mathbf{f}(\mathbf{x}) = \mathbf{z} \in Z\}$ and $f_i^{max} = \max\{f_i(\mathbf{x}), \mathbf{f}(\mathbf{x}) = \mathbf{z} \in Z\}$ which correspond to the minimum and maximum values for the ith coordinate of the objective space. Then we scale all points according to

$$\overline{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - f_i^{min}}{f_i^{max} - f_i^{min}}. \tag{29}$$

We repeat this process for all coordinates, i.e. $i = 1 \dots n$. We show the normalized vector of a vector $\mathbf{z}$ as $\overline{\mathbf{z}}$. Similarly, the set of normalized vectors are shown as $\overline{Z}$.

4.4.2 D-metric for Accuracy

Given two normalized nondominated sets $\overline{Z}_1$ and $\overline{Z}_2$, $\forall \overline{\mathbf{a}} \in \overline{Z}_1$ we look for $\exists \overline{\mathbf{b}} \in \overline{Z}_{21} \subseteq \overline{Z}_2$ such that $\overline{\mathbf{b}} < \overline{\mathbf{a}}$. Then we compute Euclidean distances from $\overline{\mathbf{a}}$ to all points $\overline{\mathbf{b}} \in \overline{Z}_{21}$. We define $\|\overline{\mathbf{a}} - \overline{\mathbf{b}}\|_{max} = \max\{\|\overline{\mathbf{a}} - \overline{\mathbf{b}}\|, \overline{\mathbf{a}} \in \overline{Z}_1, \overline{\mathbf{b}} \in \overline{Z}_{21}\}$ to be the maximum of such distances. If $\overline{Z}_{21} = \emptyset$ then $\|\overline{\mathbf{a}} - \overline{\mathbf{b}}\|_{max} = 0$.

$$D(\overline{Z}_1, \overline{Z}_2) = \sum_{\overline{\mathbf{a}} \in \overline{Z}_1} \frac{\|\overline{\mathbf{a}} - \overline{\mathbf{b}}\|_{max}}{\sqrt{n}|\overline{Z}_1|}, \tag{30}$$

where n is the dimension of the objective space. The D-metric returns a value in the range $[0, 1]$ where a smaller value is better. As seen in Figure 4(a), the maximum distance from a dominating point is taken as a basis for accuracy.

4.4.3 Δ-metric for Uniformity

Given a normalized nondominated set $\overline{Z}$, we define d_i to be the Euclidean distance between two consecutive vectors, $i = 1 \dots (|\overline{Z}| - 1)$. Let $\hat{d} =$

$\sum_{i=1}^{|\overline{Z}|-1} \frac{d_i}{|\overline{Z}|-1}$. Then we have

$$\Delta(\overline{Z}) = \sum_{i=1}^{|\overline{Z}|-1} \frac{|d_i - \hat{d}|}{\sqrt{n}(|\overline{Z}| - 1)}, \tag{31}$$

where n is the dimension of the objective space. Note that $0 \leq \Delta(\overline{Z}) \leq 1$ where 0 is the best. The underlying idea is to first calculate the average distance between any two consecutive points, and then to check all distances and penalize with respect to the deviation from the average distance. In the ideal case, all distances $d_1, d_2, \cdots, d_{n-1}$ in Figure 4(b) are equal to each other and the metric gets a value of 0.

4.4.4 ∇-metric for Extent

Given a nondominated set Z, we define $f_i^{min} = \min\{f_i(\mathbf{x}), \mathbf{f}(\mathbf{x}) = \mathbf{z} \in Z\}$ and $f_i^{max} = \max\{f_i(\mathbf{x}), \mathbf{f}(\mathbf{x}) = \mathbf{z} \in Z\}$. Then

$$\nabla(Z) = \prod_{i=1}^{n} |f_i^{max} - f_i^{min}|, \tag{32}$$

where n is the dimension of the objective space. For this metric, normalization of vectors is not needed. As shown in Figure 4(c), a bigger value spans a larger portion of the hypervolume and therefore is always better.

5 Experiments

For the experiments we have taken two media applications and a platform architecture to map the former onto the latter. The first application is a modified Motion-JPEG encoder which differs from traditional encoders in three ways: it only supports lossy encoding while traditional encoders support both lossless and lossy encodings, it can operate on YUV and RGB video data whereas traditional encoders usually operate on the YUV format, and finally it can change quantization and Huffman tables dynamically while the traditional encoders have no such behavior. We omit giving further details on the M-JPEG encoder as they are not crucial for the experiments performed here. Interested readers are pointed to [33].

The second application is a Philips in-house JPEG decoder from [12]. Regarding this application, we only have the topology information but not the real implementation. Therefore, we have synthetically generated all its parameter values. Both media applications and the platform architecture are given in Figure 5. Although these two applications match in terms of complexity, the JPEG decoder has a more complex structure since its processes are defined at a finer granularity. In two case studies performed

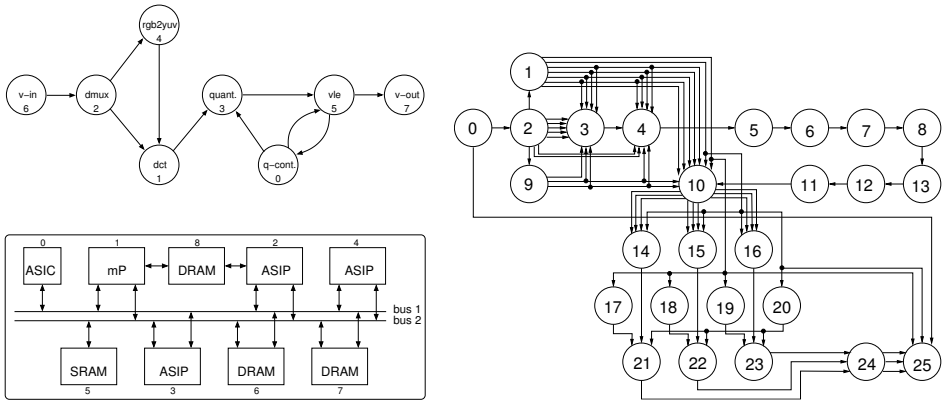


Figure 5: Two multimedia applications and a platform architecture is shown. The M-JPEG encoder application and the multiprocessor System-on-Chip (SoC) architecture model are given on the left; on the right is shown the JPEG decoder process network with 26 processes and 75 FIFO channels.

here, we have mapped these media applications successively onto the same platform architecture consisting of a general purpose microprocessor (mP), three ASIPs, an Application Specific Integrated Circuit (ASIC), an SRAM and three DRAMs. For these architecture components, realistic latency values from [33] have been used to calculate their processing capacities: c_p and c_m. Similarly, for the Kahn processes and FIFO channels in the M-JPEG decoder, computational and communicational requirements (namely the parameters α_a for the nodes and the parameters α_b and β_b for the FIFO channels) have been calculated using statistics obtained from the C++ implementation code of its Kahn process network.

We have implemented the MMPN problem as an optimization problem module in PISA – A platform and programming language independent interface for search algorithms [6]. In PISA, the optimization process is split into two modules. One module contains all parts specific to the optimization problem such as individual encoding, fitness evaluation, mutation, and crossover. The other module contains the problem-independent parts such as selection and truncation. These two modules are implemented as two separate processes communicating through text files. The latter provides huge flexibility because a problem module can be freely combined with an optimizer module and vice versa. Due to the communication via file system, platform, programming language and operating system independence are also achieved.

For M-JPEG encoder and JPEG decoder mapping problems, we have utilized the state-of-the-art highly competitive SPEA2 and NSGA-II multi-objective evolutionary optimizers. As already mentioned in Section 4.3.2, we

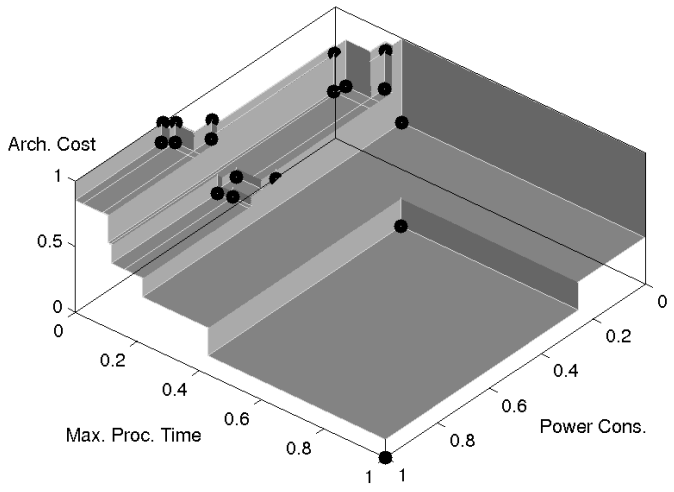


Figure 6: Pareto front for the M-JPEG encoder case study.

have used a repair algorithm (Algorithm 2) to handle constraint violations. In order to examine the effect of repair usage on the MOEA performance, we have utilized three different repair strategies (no-repair, moderate-repair, and intensive-repair), the details of which have already been discussed in Section 4.3.2. In the rest of the paper we present the results obtained under the *no-repair* strategy with SPEA2 (NSGA-II), while the results for the *moderate-repair* and *intensive-repair* strategies are shown by SPEA2r (NSGA-IIr) and SPEA2R (NSGA-IIR), respectively.

In the experiments, we have used the standard one-point crossover and independent bit mutation variators. The population size is kept constant. All performance analyses are carried out at different numbers of generations, ranging from 50 to 1000, collecting information on the whole evolution of MOEA populations. The following values are used for the specific parameters:

- Population size, $N = 100$.
- Maximum number of generations, $T = 50, 100, 200, 300, 500, 1000$.
- Mutation probability[3] 0.5, bit mutation probability[4] 0.01.
- Crossover probability 0.8.

The D-metric for measuring convergence to the Pareto front requires a reference set. To this end, we implemented the lexicographic weighted Tchebycheff method and solved the M-JPEG encoder mapping problem by using

[3]i.e., the likelihood of mutating a particular solution.
[4]i.e., the likelihood of mutating each bit of a solution in mutation.

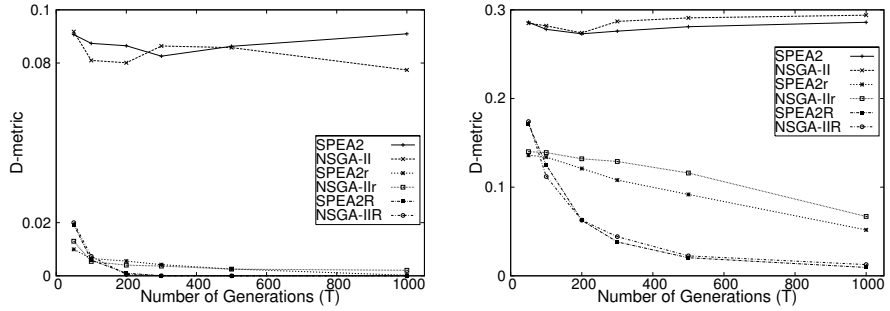


Figure 7: Convergence analyses of the D-metric for the M-JPEG encoder (left) and the JPEG decoder (right). In the case of the M-JPEG decoder, the reference set is obtained by CPLEX, while for the JPEG decoder it is obtained by running SPEA2r with $T = 10000$.

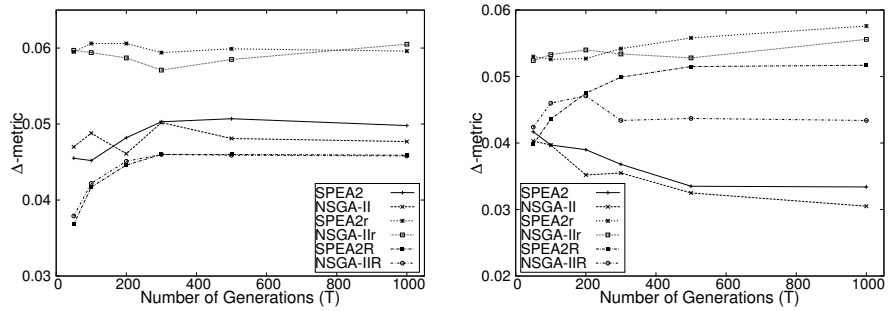


Figure 8: Convergence analyses of the Δ-metric for the M-JPEG encoder (left) and the JPEG decoder (right).

the CPLEX Mixed Integer Optimizer [1]. The outcome of numerous runs with different weights has resulted in 18 Pareto-optimal points which are plotted in Figure 6. The JPEG decoder is not solved by this exact method due to its size, instead the result obtained by running SPEA2 for $T = 10,000$ is taken as the reference set.

The following table summarizes the experimental setup.

MOEA	repair strategy	crossover	mutation
SPEA2	*no-repair*	one-point	indep. bit
SPEA2r	*moderate-repair*	one-point	indep. bit
SPEA2R	*intensive-repair*	one-point	indep. bit
NSGA-II	*no-repair*	one-point	indep. bit
NSGA-IIr	*moderate-repair*	one-point	indep. bit
NSGA-IIR	*intensive-repair*	one-point	indep. bit

In the experiments we have performed 30 runs (varying the random generator seed from 1 to 30) for each setup. An MOEA with some chosen

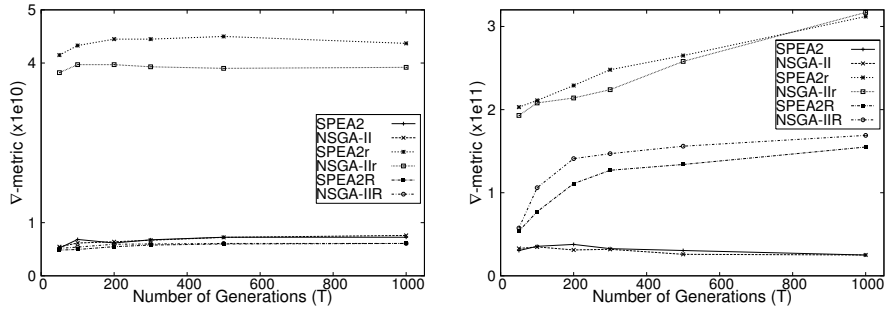


Figure 9: Convergence analyses of the ∇-metric for the M-JPEG encoder (left) and the JPEG decoder (right).

T makes up a setup: SPEA2 with $T = 50$ or NSGA-IIR with $T = 500$ are two examples. As a result we have obtained 30 nondominated sets for each setup. All experiments have been performed on an Intel Pentium M PC with 1.7 GHz CPU and 512 MB RAM running Linux OS.

5.1 MOEA Performance Comparisons

Table 3 in the appendix presents averages and standard deviations of the three performance metrics for each experimental setup with respect to 30 runs. The results for the same number of generations are grouped and compared. The best values obtained for all metrics are given in bold. To visualize the metrics convergence, we have plotted average metrics values against the numbers of generations in Figures 7, 8, and 9. We have the following conclusions:

- In terms of all three metrics, SPEA2 and NSGA-II score very close numbers and overall can be considered evenly matched. The same is true between SPEA2r and NSGA-IIr, and also for SPEA2R and NSGA-IIR. However with respect to run-times, NSGA-II, NSGA-IIr and NSGA-IIR outperform SPEA2, SPEA2r and SPEA2R by only demanding on average 44% of their rivals' run-times. The latter is also demonstrated in Figure 10 where we plot D-metric values with respect to wall clock time. Therefore, the finer-grained computationally-expensive fitness assignment in SPEA2 (also in SPEA2r and SPEA2R) does not seem to pay off in general.

- In terms of accuracy (D-metric), SPEA2R and NSGA-IIR clearly outperform SPEA2r and NSGA-IIr. The worst performance is obtained when no repair is used, as clearly observed in Figure 7, SPEA2 and NSGA-II fail to converge to the Pareto front. Therefore, constraint handling is of crucial importance.

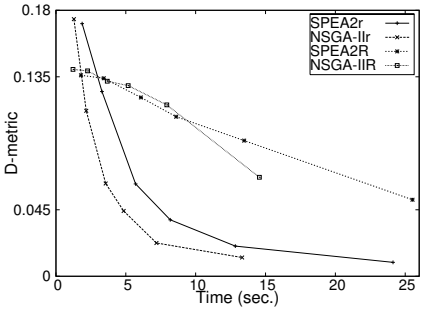


Figure 10: Convergence with respect to time.

- From D-metric plots in Figure 7 and Figure 10 we observe that convergence to the Pareto front accelerates with higher usage of repair. In this respect we observe an exponential convergence curve for SPEA2R and NSGA-IIR, while the convergence of SPEA2r and NSGA-IIr approximates a linear curve. However, as the number of generations increase, the difference in terms of D-metric between SPEA2R (NSGA-IIR) and SPEA2r (NSGA-IIr) diminishes.

- In terms of uniformity, all algorithms perform indifferently. Although they start from a good initial value, they all fail to converge towards the optimal value zero. It is also very difficult to come to any conclusion about their behaviors from Figure 8, e.g. it is unclear whether repair has any positive or negative effect on the Δ-metric. Overall, all algorithms can be considered as good in terms of uniformity, as they all score below 0.06 which is reasonably close to the optimal value.

- SPEA2r and NSGA-IIr clearly outperform other variants in terms of the extent metric. The reason behind this may be the higher explorative capacity of SPEA2r and NSGA-IIr, as they can locate diverse feasible regions by mutating the invalid individuals. In this metric, SPEA2R and NSGA-IIR come second. Also from the ∇-metric plot for JPEG decoder problem in Figure 9, we observe convergence behavior for SPEA2r, NSGA-IIr, SPEA2R, NSGA-IIR, but not for SPEA2 and NSGA-II. Therefore, repair is essential for good extension but there seems to be no linear relation between the two.

- In the M-JPEG encoder case study, we compare nondominated sets generated by the MOEAs against the exact Pareto set (obtained by the lexicographic weighted Tchebycheff method in CPLEX). As the numbers are very close to the ideal value 0 for $T \geq 300$, especially for SPEA2R and NSGA-IIR, we consider them as highly promising optimizers. Convergence to Pareto front is also achieved with SPEA2r and NSGA-IIr, but this takes considerably larger amount of time.

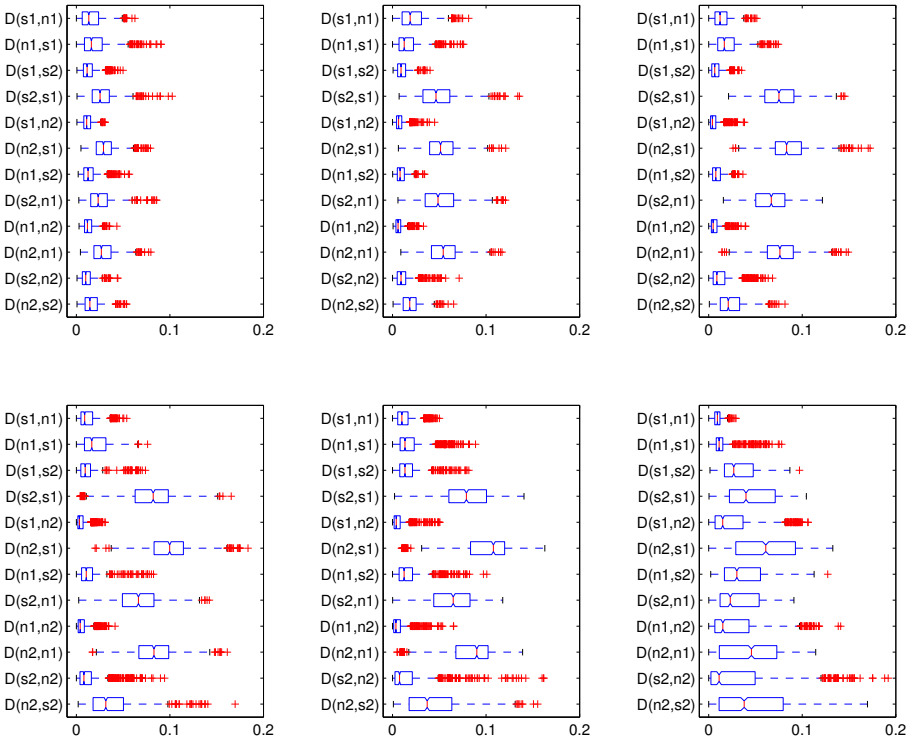


Figure 11: Notched boxplots showing the distribution of D-metric values for reciprocal comparison of MOEAs. Each nondominated set of a MOEA is compared with respect to each nondominated set from the other, and the resulting 900 comparisons is plotted. The plots are given for the JPEG decoder case study in ascending number of generations (50, 100, 200, 300, 500, and 1000) in the order from left to right and top to bottom. Hence, the top leftmost plot corresponds to MOEA comparisons with $N = 50$, while the bottom rightmost is for comparisons with $N = 1000$. The abbreviations s1, s2, n1, n2 are used in places of SPEA2R, SPEA2r, NSGA-IIR, NSGA-IIr, respectively. Comparisons regarding SPEA2 and NSGA-II are not given as they fail to converge (Section 5.1). Since D-metric is non-symmetric, i.e. $D(Z_1, Z_2) \neq D(Z_2, Z_1)$, both comparisons are performed. Note that smaller values are always better.

- In Figure 11 we perform reciprocal comparison of nondominated sets at numbers of generations 50, 100, 200, 300, 500, and 1000 for the JPEG decoder case study. To perform one comparison at a certain number of generations, 30 nondominated sets from an MOEA is compared one by one with the 30 nondominated sets from the other. The resulting distribution of 900 D-metric comparisons is given as a single

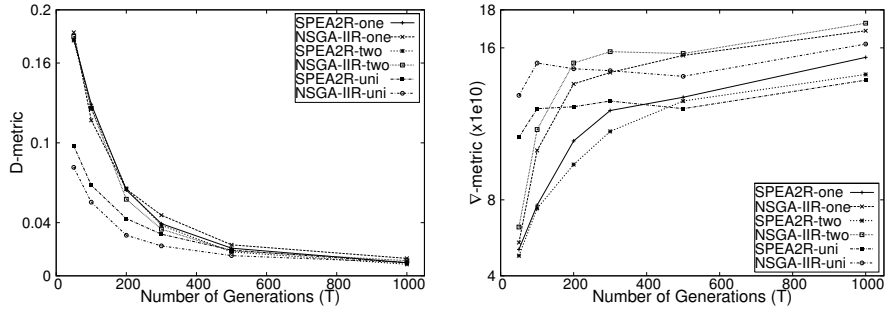


Figure 12: Crossover analysis for the JPEG encoder case study.

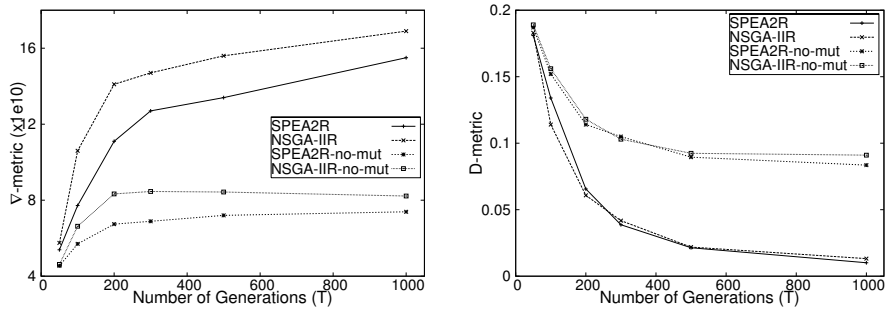


Figure 13: Using no mutation results in poor ∇-metric values in the JPEG encoder case study. This is an expected result as the mutation operator is responsible for exploration of the search space.

notched boxplot in Figure 11. The comparisons unveil that SPEA2R and NSGA-IIR beat SPEA2r and NSGA-IIr in all comparisons; and SPEA2R and NSGA-IIR can be considered as equally matched. The same is true between SPEA2r and NSGA-IIr. However, as the number of generations increase, the difference in D-metric between SPEA2R (NSGA-IIR) and SPEA2r (NSGA-IIr) diminishes as a result of the belated convergence of SPEA2r and NSGA-IIr. This difference in convergence speed is also apparent in Figure 7, where we observe an exponential convergence curve for SPEA2R and NSGA-IIR in contrast to a linear curve for SPEA2r and NSGA-IIr.

5.2 Effect of Crossover and Mutation

In this section we have performed two independent experiments with the JPEG decoder case study in order to analyze the effect of crossover and mutation operators on different MOEA performance criteria. The purpose of the first experiment is to examine the correlation between crossover type

Table 2: Three solutions chosen for simulation.

Solution	Max. Processing Time	Power Cons.	Arch. Cost
cplex1	129338.0	1166.2	160.0
cplex2	162203.7	966.9	130.0
ad-hoc	167567.3	1268.0	170.0

and convergence to the Pareto front. In this respect, besides the default one-point crossover operator, we have implemented two-point and uniform crossover operators (see Section 4.3.3 for how they work). When we look at the resulting D-metric plots in Figure 12 we observe a better performance with uniform crossover in the early generations; however after $T = 500$, all crossover operators exhibit very close performance. With respect to extent (∇-metrics in Figure 12) two point crossover shows the worst performance, while once again one-point and uniform crossover operators match each other. The relatively fast convergence but coincidentally poor coverage of the search space in the case of the two-point crossover implies that the operator is biased more towards exploitation than exploration. One-point and uniform crossover operators seem to find a better balance of the two in this case.

In the second experiment we analyze the effect of the mutation operator on MOEA performance. To realize this we have taken our original experimental setup (see Section 5) and repeated the experiments without the mutation operator. The resulting D-metric and ∇-metric plots are given in Figure 13. With respect to both metrics omitting the mutation operator has resulted in very poor performance. MOEAs without mutation seem to converge towards the local optima and fail to collect variant solutions. Both observations imply that insufficient exploration has been realized in the search. These implications are in accordance with the widely accepted role of mutation as providing a reasonable level of population diversity in the standard EAs [36]. This experiment suggests that the explorative role of mutation is of high importance for MOEAs as well.

5.3 Simulation Results

In this section, we use the Sesame framework in order to evaluate three selected solutions of the M-JPEG encoder problem by means of simulation. Two of the solutions are taken from the Pareto-optimal set (referred here as cplex1 and cplex2), while the third solution is an ad-hoc solution (referred as ad-hoc) which is very similar to those proposed and studied in [29], [33]. It is clear from their objective function values in Table 2 that Pareto-optimal cplex1 and cplex2 outperform the ad-hoc solution in all objectives. The outcome of simulation experiments are also in accordance with optimization results, as the results in Figure 14 reveal that similar performance can be

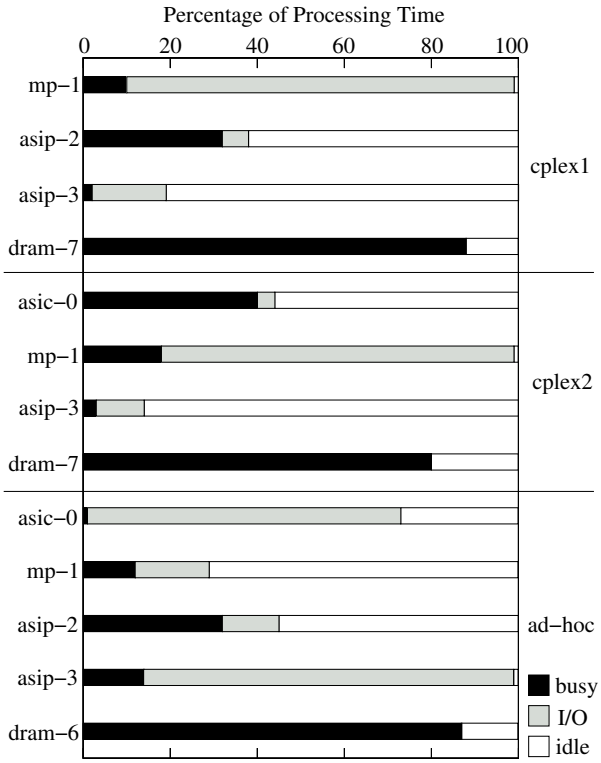


Figure 14: Simulation results showing the utilization of architecture components in all three solutions. The throughput values are 52.2, 47.8 and 51.3 frames/sec for the cplex1, cplex2 and ad-hoc solutions, respectively.

achieved using less processing cores (cplex1 and cplex2 use three while ad-hoc uses four processors), which in turn results in less power consumption and cheaper implementation.

6 Conclusion

In this paper, we studied a multiobjective design problem from the multiprocessor system-on-chip domain: mapping process networks onto heterogeneous multiprocessor architectures. The mathematical model for the problem takes into account three objectives, namely the maximum processing time, power consumption, and cost of the architecture, and is formulated as a nonlinear mixed integer programming. We have used an exact (lexicographic weighted Tchebycheff) method and two state-of-the-art MOEAs (SPEA2 [47] and NSGA-II [14]) in order to locate the Pareto-optimal solutions. To apply the exact method, we first linearized the mathematical model by adding additional binary decision variables and new constraints.

Three new performance metrics have been defined to measure three attributes (accuracy, uniformity and extent) of nondominated sets. These metrics are subsequently used to compare SPEA2 and NSGA-II with each other and also with the Pareto-optimal set. The two elitist MOEAs mainly differ in their fitness assignment schemes. SPEA2 uses a finer-grained and computationally more expensive scheme with respect to its rival NSGA-II. Performing two case studies, we have shown that SPEA2 is not superior than NSGA-II in any of the three defined metrics. Therefore regarding the MMPN problem, the computationally more expensive fitness assignment scheme of SPEA2 does not seem to pay off, as NSGA-II is on average 2.2 times faster. Comparing the two MOEAs with the exact set in the M-JPEG encoder case study, we have shown that both SPEA2 and NSGA-II find solution sets very close to the Pareto-optimal set.

Constraint violations have been tackled by three repair strategies differing in terms of repair intensity, and the outcome of each strategy is evaluated with respect to the defined metrics. The main result is that using sufficient amount of repair is necessary for good convergence, but allowing some infeasibility may help the MOEA to explore new feasible regions over infeasible solutions. Thus, a balance should be established in terms of repair frequency.

Additionally, one-point, two-point, and uniform crossover operators have been comparatively evaluated in terms of accuracy and extent. To summarize, one-point and uniform crossover operators seem to find a good balance of exploitation vs. exploration, while two-point crossover is more biased towards exploitation. With respect to mutation usage, the experiments reveal that mutation retains its importance for exploration.

We have also compared and simulated two Pareto-optimal solutions and one ad-hoc solution from previous studies [29], [33]. The results indicate that multiobjective search of the design space improves all three objectives, i.e. a cheaper implementation using less power but still performing the same in terms of system throughput can be achieved.

References

[1] ILOG CPLEX, http://www.ilog.com/products/cplex.

[2] G. Alpaydin, S. Balkir, and G. Dundar. An evolutionary approach to automatic synthesis of high-performance analog integrated circuits. *IEEE Transactions on Evolutionary Computation*, 7(3):240–252, 2003.

[3] G. Ascia, V. Catania, and M. Palesi. A GA-based design space exploration framework for parameterized system-on-a-chip platforms. *IEEE Transactions on Evolutionary Computation*, 8(4):329–346, 2004.

[4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[5] D. Beasley, D. Bull, and R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[6] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA – a platform and programming language independent interface for search algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508. Springer-Verlag, 2003.

[7] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, 1998.

[8] M. S. Bright and T. Arslan. Synthesis of low-power DSP systems using a genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 5(1):27–40, 2001.

[9] C. A. Coello Coello. Guest editorial: Special issue on evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 7(2):97–99, 2003.

[10] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[11] J. E. Coffland and A. D. Pimentel. A software framework for efficient system-level performance evaluation of embedded systems. In *Proc. of the ACM Symposium on Applied Computing*, pages 666–671, Mar. 2003. `http://sesamesim.sourceforge.net/`.

[12] E. A. de Kock. Multiprocessor mapping of process networks: A JPEG decoding case study. In *Proc. of Int. Symposium on System Synthesis*, pages 68–73, Oct. 2002.

[13] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[14] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 849–858. Springer, Berlin, 2000.

[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*, pages 920–935, Oct. 1998.

[16] M. Ehrgott. *Multicriteria Optimization*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Heidelberg, 2000.

[17] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. In *Proc. of the Int. Conference on Hardware/Software Codesign and System Synthesis*, pages 182–187, Oct. 2003.

[18] C. M. Fonseca and P. J. Flemming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proc. of the Int. Conference on Genetic Algorithms*, pages 416–423, 1993.

[19] J. C. Gallagher, S. Vigraham, and G. Kramer. A family of compact genetic algorithms for intrinsic evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 8(2):111–126, 2004.

[20] T. Givargis, F. Vahid, , and J. Henkel. Instruction-based system-level power evaluation of system-on-a-chip peripheral cores. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(6):856–863, 2002.

[21] T. Givargis and F. Vahid. Platune: A tuning framework for system-on-a-chip platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1317–1327, 2002.

[22] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):416–422, 2002.

[23] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[24] M. T. Jensen. Reducing the run-time complexity of multi-objective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, 2003.

[25] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress*, pages 471–475, 1974.

[26] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 19(12), 2000.

[27] J. Knowles and D. Corne. On metrics for comparing non-dominated sets. In *Proc. of the Congress on Evolutionary Computation*, pages 711–716, May 2002.

[28] M. Laumanns, E. Zitzler, and L. Thiele. A unified model for multi-objective evolutionary algorithms with elitism. In *Proc. of the Congress on Evolutionary Computation*, pages 46–53, 2000.

[29] P. Lieverse, T. Stefanov, P. van der Wolf, and E. F. Deprettere. System level design with Spade: an M-JPEG case study. In *Proc. of the Int. Conference on Computer Aided Design*, pages 31–38, Nov. 2001.

[30] H. Lu and G. G. Yen. Rank-density-based multiobjective genetic algorithm and benchmark test function study. *IEEE Transactions on Evolutionary Computation*, 7(4):325–343, 2003.

[31] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded systems architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 2006. to be published.

[32] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, and E. F. Deprettere. Exploring embedded-systems architectures with Artemis. *IEEE Computer*, pages 57–63, Nov. 2001.

[33] A. D. Pimentel, S. Polstra, F. Terpstra, A. W. van Halderen, J. E. Coffland, and L. O. Hertzberger. Towards efficient design space exploration of heterogeneous embedded media systems. In E. Deprettere, J. Teich, and S. Vassiliadis, editors, *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation*, volume 2268 of *LNCS*, pages 57–73. Springer-Verlag, 2002.

[34] G. Rudolph. Evolutionary search under partially ordered fitness sets. In M. F. Sebaaly, editor, *Proc. of the Int. NAISO Congress on Information Science Innovations*, pages 818–822, Millet, Canada, 2001. ICSC Academic Press.

[35] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proc. of the Int. Conference on Genetic Algorithms*, pages 93–100, 1985.

[36] W. Spears. Crossover or mutation? In *Proc. of the Workshop on Foundations of Genetic Algorithms*, pages 221–237, July 1992.

[37] N. Srinivas and K. Deb. Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[38] A. Stammermann, L. Kruse, W. Nebel, A. Pratsch, E. Schmidt, M. Schulte, and A. Schulz. System level optimization and design space exploration for low power. In *Proc. of the Int. Symposium on Systems Synthesis*, pages 142–146, Oct. 2001.

[39] R. E. Steuer. An overview in graphs of multiple objective programming. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proc. of the Int. Conference on Evolutionary Multi-Criterion Optimization*, volume 1993 of *LNCS*, pages 41–51. Springer-Verlag, 2001.

[40] R. E. Steuer and E. U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 1983.

[41] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. of the Int. Conference on Genetic Algorithms*, pages 2–9, 1989.

[42] R. Szymanek, F. Catthoor, and K. Kuchcinski. Time-energy design space exploration for multi-layer memory architectures. In *Proc. of the Design, Automation and Test in Europe*, pages 10318–10323, Feb. 2004.

[43] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. of the Design Automation Conference*, pages 880–885, June 2002.

[44] G. G. Yen and L. Haiming. Dynamic multiobjective evolutionary algorithm: Adaptive cell-based rank and density estimation. *IEEE Transactions on Evolutionary Computation*, 7(3):253–274, 2003.

[45] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.

[46] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[47] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100. CIMNE, Barcelona, Spain, 2002.

A Performance Metrics

Table 3 presents the mean values and the standard deviations for the three metrics obtained in the M-JPEG encoder and JPEG decoder case studies. Best values are shown in bold.

Table 3: Performance comparison of the MOEAs for the M-JPEG encoder and JPEG decoder applications. Best values are in bold.

		M-JPEG encoder						JPEG decoder					
MOEA	T	D-metric		Δ-metric		∇-metric		D-metric		Δ-metric		∇-metric	
		avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.
SPEA2	50	9.07e-2	2.77e-2	4.55e-2	1.12e-2	5.22e9	2.58e9	2.86e-1	1.60e-2	4.17e-2	8.19e-3	3.04e10	1.18e10
NSGA-II		9.18e-2	2.25e-2	4.70e-2	1.35e-2	5.47e9	3.01e9	2.85e-1	2.18e-2	4.03e-2	7.60e-3	3.29e10	9.54e9
SPEA2r		**9.94e-3**	8.93e-3	5.95e-2	1.15e-2	**4.15e10**	8.65e9	**1.36e-1**	2.54e-2	5.30e-2	8.10e-3	**2.03e11**	3.75e10
NSGA-IIr		1.30e-2	1.28e-2	5.97e-2	1.05e-2	3.82e10	9.55e9	1.40e-1	1.62e-2	5.24e-2	5.58e-3	1.93e11	2.33e10
SPEA2R		1.93e-2	2.27e-2	**3.68e-2**	7.27e-3	4.84e9	2.37e9	1.71e-1	1.74e-2	**3.98e-2**	5.12e-3	5.39e10	1.41e10
NSGA-IIR		2.00e-2	2.26e-2	3.79e-2	6.27e-3	5.17e9	1.63e9	1.74e-1	1.57e-2	4.24e-2	5.51e-3	5.75e10	2.02e10
SPEA2	100	8.74e-2	2.54e-2	4.52e-2	1.23e-2	6.84e9	4.15e9	2.78e-1	1.80e-2	**3.97e-2**	7.47e-3	3.55e10	1.02e10
NSGA-II		8.10e-2	1.81e-2	4.88e-2	9.81e-3	6.17e9	2.17e9	2.82e-1	1.87e-2	**3.97e-2**	7.83e-3	3.48e10	1.16e10
SPEA2r		6.50e-3	1.01e-2	6.06e-2	1.19e-2	**4.33e10**	9.71e9	1.34e-1	2.47e-2	5.26e-2	6.95e-3	**2.11e11**	4.00e10
NSGA-IIr		**5.41e-3**	8.41e-3	5.94e-2	8.44e-3	3.97e10	8.59e9	1.39e-1	1.94e-2	5.33e-2	5.51e-3	2.08e11	3.64e10
SPEA2R		5.90e-3	1.15e-2	**4.17e-2**	6.30e-3	4.95e9	1.78e9	1.25e-1	2.02e-2	4.36e-2	4.53e-3	7.72e10	2.09e10
NSGA-IIR		7.32e-3	1.64e-2	4.22e-2	5.26e-3	5.40e9	1.56e9	**1.12e-1**	2.20e-2	4.60e-2	4.68e-3	1.06e11	3.25e10
SPEA2	200	8.65e-2	2.12e-2	4.82e-2	1.05e-2	6.18e9	1.86e9	2.73e-1	2.27e-2	3.90e-2	8.15e-3	3.78e10	1.44e10
NSGA-II		8.01e-2	1.68e-2	4.61e-2	1.16e-2	6.40e9	1.65e9	2.74e-1	2.18e-3	**3.52e-2**	6.18e-3	3.11e10	9.54e9
SPEA2r		5.55e-3	1.06e-2	6.06e-2	1.16e-2	**4.45e10**	9.57e9	1.21e-1	2.30e-2	5.27e-2	7.18e-3	**2.29e11**	4.38e10
NSGA-IIr		4.01e-3	8.00e-3	5.87e-2	8.68e-3	3.97e10	9.61e9	1.32e-1	2.31e-2	5.40e-2	5.92e-3	2.14e11	4.02e10
SPEA2R		9.85e-4	2.29e-3	**4.46e-2**	3.45e-3	5.48e9	1.49e9	**6.25e-2**	1.79e-2	4.75e-2	5.23e-3	1.11e11	2.91e10
NSGA-IIR		**5.50e-4**	2.08e-3	4.51e-2	3.30e-3	5.94e9	8.62e8	6.28e-2	1.71e-2	4.71e-2	4.03e-3	1.41e11	3.57e10

Table 3: Performance comparison of the MOEAs for the M-JPEG encoder and JPEG decoder applications. Best values are in bold. (*continued*)

		M-JPEG encoder						JPEG decoder					
MOEA	T	D-metric		Δ-metric		∇-metric		D-metric		Δ-metric		∇-metric	
		avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.
SPEA2	300	8.26e-2	1.98e-2	5.03e-2	1.30e-2	6.77e9	2.05e9	2.76e-1	2.21e-2	3.68e-2	8.56e-3	3.27e10	1.15e10
NSGA-II		8.64e-2	2.15e-2	5.02e-2	1.41e-2	6.69e9	1.70e9	2.87e-1	1.96e-2	**3.55e-2**	5.66e-3	3.19e10	8.27e9
SPEA2r		4.20e-3	8.58e-3	5.94e-2	1.02e-2	**4.45e10**	9.91e9	1.08e-1	3.19e-2	5.42e-2	7.35e-3	**2.48e11**	4.82e10
NSGA-IIr		3.67e-3	8.34e-3	5.71e-2	8.77e-3	3.93e10	9.73e9	1.29e-1	2.71e-2	5.34e-2	6.68e-3	2.24e11	5.16e10
SPEA2R		3.49e-5	1.91e-4	**4.60e-2**	1.16e-3	5.78e9	9.80e8	**3.82e-2**	1.61e-2	4.99e-2	4.72e-3	1.27e11	2.38e10
NSGA-IIR		**1.37e-5**	7.52e-5	**4.60e-2**	1.17e-3	5.99e9	5.92e8	4.42e-2	2.02e-2	4.34e-2	5.66e-3	1.47e11	2.65e10
SPEA2	500	8.63e-2	2.31e-2	5.07e-2	1.46e-2	7.25e9	3.16e9	2.81e-1	2.22e-2	3.35e-2	7.34e-3	3.05e10	8.73e9
NSGA-II		8.58e-2	1.81e-2	4.81e-2	1.31e-2	7.23e9	1.47e9	2.91e-1	2.01e-2	**3.25e-2**	6.79e-3	2.59e10	1.04e10
SPEA2r		2.55e-3	7.19e-3	5.99e-2	1.02e-2	**4.50e10**	1.15e10	9.18e-2	3.32e-2	5.58e-2	8.17e-3	**2.65e11**	4.30e10
NSGA-IIr		2.52e-3	4.81e-3	5.85e-2	7.79e-3	3.90e10	9.53e9	1.16e-1	3.30e-2	5.28e-2	8.29e-3	2.58e11	7.30e10
SPEA2R		3.49e-5	1.91e-4	4.60e-2	1.12e-3	5.99e9	5.79e8	**2.04e-2**	1.36e-2	5.15e-2	4.75e-3	1.34e11	1.97e10
NSGA-IIR		**0.00e0**	0.00e0	**4.59e-2**	1.20e-3	6.09e9	4.27e7	2.25e-2	1.67e-2	4.37e-2	5.01e-3	1.56e11	2.61e10
SPEA2	1000	9.10e-2	1.82e-2	4.98e-2	1.22e-2	7.26e9	4.43e9	2.86e-1	2.00e-2	3.34e-2	6.50e-3	2.50e10	8.76e9
NSGA-II		7.74e-2	1.44e-2	4.77e-2	9.89e-3	7.59e9	4.44e9	2.94e-1	2.04e-2	**3.05e-2**	6.86e-3	2.51e10	9.26e9
SPEA2r		2.02e-4	6.77e-4	5.96e-2	1.04e-2	**4.37e10**	1.17e10	5.18e-2	3.08e-2	5.76e-2	7.44e-3	3.12e11	6.76e10
NSGA-IIr		2.09e-3	4.90e-3	6.05e-2	9.55e-3	3.92e10	9.68e9	6.70e-2	4.04e-2	5.56e-2	1.25e-2	**3.17e11**	8.82e10
SPEA2R		3.38e-5	1.85e-4	4.59e-2	1.13e-3	6.09e9	3.55e7	**9.43e-3**	3.83e-3	5.17e-2	4.63e-3	1.55e11	1.97e10
NSGA-IIR		**0.00e0**	0.00e0	**4.58e-2**	9.56e-4	6.09e9	3.55e7	1.27e-2	9.33e-3	4.34e-2	5.07e-3	1.69e11	1.84e10