# Efficient Symbolic Multi–Objective Design Space Exploration

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich
Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz, glass, haubelt, teich}@cs.fau.de

**Abstract — Nowadays many design space exploration tools are based on Multi–Objective Evolutionary Algorithms (MOEAs). Beside the advantages of MOEAs, there is one important drawback as MOEAs might fail in design spaces containing only a few feasible solutions or as they are often afflicted with premature convergence, i.e., the same design points are revisited again and again. Exact methods, especially Pseudo Boolean solvers (PB solvers) seem to be a solution. However, as typical design spaces are multi–objective, there is a need for *multi–objective PB solvers*. In this paper, we will formalize the problem of design space exploration as multi–objective 0–1 ILP. We will propose (1) a heuristic approach based on PB solvers and (2) a complete multi–objective PB solver based on a backtracking algorithm that incorporates the non–dominance relation from multi–objective optimization and is restricted to linear objective functions. First results from applying our novel multi–objective PB solver to synthetic problems will show its effectiveness in small sized design spaces as well as in large design spaces only containing a few feasible solutions. For non–linear and large problems, the proposed heuristic approach is outperforming common MOEA approaches. Finally, a real world example from the automotive area will emphasize the efficiency of the proposed algorithms.**

## I. Introduction and Related Work

The automatic optimization of embedded systems with respect to several and often conflicting objectives, also known as *design space exploration*, is still a challenging task in electronic system level design. This task is twofold [1]: (1) each design point has to be *evaluated* regarding all objective functions and (2) a strategy for *covering* the search space is needed. State–of–the–art approaches in automatic design space exploration are based on *Multi–Objective Evolutionary Algorithms* (MOEAs), e.g., [2, 3, 4]. MOEAs have several advantages over other optimization strategies, e.g., they do not make any assumption on the objective functions. However, MOEAs also have some severe limitations: Firstly, MOEAs often fail in search spaces containing only a few feasible solutions. Secondly, after a convergence phase, MOEAs are not able to explore the search space even more. Instead, MOEAs revisit already known solutions again and again.

A remedy to these drawbacks might be symbolic representations [5] or Integer Linear Programming (ILP) in automatic design space exploration [6]. The basic design space exploration problem can be reduced to an ILP problem with binary variables (0–1 ILP). A 0–1 ILP is stated as

$$min\{f(x)|Ax \leq b\} \tag{1}$$

where the linear objective function $f(x) = c^T x$ is optimized considering a set of linear constraints[1] $Ax \leq b$ with $b \in \mathbb{Z}^k$, $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{k,n}$, and $x \in \{0,1\}^n$. However, these exact methods are afflicted by exponential runtimes making them prohibitive in the presence of real–world applications. Anyhow, the recent enhancements in the *Boolean Satisfiability* [7, 8, 9] were transfered into specialized programs, the *Pseudo Boolean (PB) solvers* [10, 11, 12], that are efficiently solving 0–1 ILPs. Although a general ILP solver will solve these problems, too, PB solvers are superior regarding the runtime [13].

Common PB solvers have some limitations, too: Firstly, they can optimize just a single objective function. Secondly, the objective function has to be linear. In this paper, we will close this gap by formalizing and solving the design space exploration problem as a multi–objective 0–1 ILP. The main contribution of this paper are two approaches solving these multi–objective 0–1 ILPs: (1) a sophisticated heuristic approach for problems with also non–linear objective functions and (2) a novel complete multi–objective PB solver for problems with only linear objective functions. The effectiveness will be shown by comparing our results with results from a design space exploration performed with a state–of–the–art MOEA approach.

The remainder of the paper is organized as follows: Section II formally defines the problem of design space exploration. In Section III, we propose the formalization of the design space exploration problem as a multi–objective 0–1 ILP and present a novel heuristic approach based on common PB solvers as well as our multi–objective PB solver based on a specialized backtracking algorithm. In Section IV, we will present first results from applying our methodologies to synthetic and real–world problem instances.

[1] Greater–relations and equalities are obtained by $Ax \geq b \Leftrightarrow -Ax \leq -b$ and $Ax = b \Leftrightarrow Ax \geq b \wedge Ax \leq b$

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

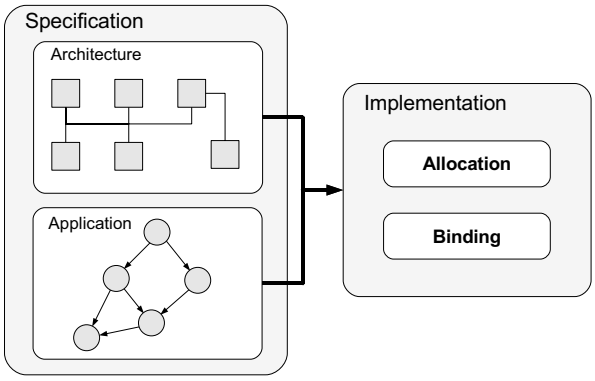


Fig. 1. The design space exploration problem.

## II. Problem Statement

In this paper, we consider the problem of design space exploration at system level. The task of design space exploration is to find the set of optimal *feasible implementations* for a given *specification*. The design flow is illustrated in Figure 1. The specification consists of the *architecture*, the *application*, and the relation between these two views:

- The architecture is modeled by a graph $g_{\mathsf{a}}(V_{\mathsf{a}}, E_{\mathsf{a}})$ and represents possible interconnected hardware resources. The vertices $r_1, ..., r_{|V_{\mathsf{a}}|} \in V_{\mathsf{a}}$ are the single resources, the edges $E_{\mathsf{a}}$ are possible communication links.
- The application is modeled by a problem graph $g_{\mathsf{p}}(V_{\mathsf{p}}, E_{\mathsf{p}})$ that describes the behavior of the system. The vertices $p_1, ..., p_{|V_{\mathsf{p}}|} \in V_{\mathsf{p}}$ are tasks whereas the directed edges $E_{\mathsf{p}}$ are data dependencies. Data–dependent tasks have to be executed on the same or adjacent resources to ensure a correct communication.
- The set of *mapping edges* $E_{\mathsf{m}}$ are indicating whether a specific task can be executed on a hardware resource. Each mapping edge $e_1, ..., e_{|E_{\mathsf{m}}|}$ is a directed edge from a task to a resource.

An implementation or *solution* of the problem is deduced from the specification and consists of two main parts:

- The *allocation* $\alpha$ is the set of hardware resources like, e.g., processors, IP cores, or buses. The allocation is a subset of $V_{\mathsf{a}}$ and represents these resource that are actually used.
- The *binding* $\beta$ determines on which allocated resource each task is executed. For each task from the problem graph exactly one mapping has to be in use. The binding is a subset of $E_{\mathsf{m}}$.

Due to data dependencies, a binding can be infeasible. A binding is called feasible if it guarantees that data communications imposed by the problem graph could by established by the allocated resources. Furthermore, a *feasible allocation* is an allocation $\alpha$ that allows at least one feasible binding $\beta$. In order to restrict the combinatorial search space, it is useful to determine the set of feasible allocations and bindings. With this knowledge, we define an implementation as a pair $(\alpha, \beta)$.

**Definition 1 (Design Space Exploration)** *The task of* design space exploration *can be formulated as the following multi–objective optimization problem:*

minimize $f(\alpha, \beta)$,
subject to:
$\alpha$ is a feasible allocation,
$\beta$ is a feasible binding,

The constraints on $\alpha$ and $\beta$ define the set of feasible implementations. The objective function $f$ consists of $m \in \mathbb{N}$ functions where in the following, without loss of generality, only minimization problems are assumed.

In single–objective optimization, the feasible set of implementations is totally ordered, whereas in multi–objective optimization problems, the feasible set is only partially ordered and, thus, there is generally not only one global optimum, but a set of *Pareto solutions*. A Pareto–optimal solution is not worse or equal in all objectives than any other feasible solution in the design space [14].

## III. Symbolic Design Space Exploration

### A. 0–1 ILP Model

In order to use a 0–1 ILP model for the design space exploration problem, cf. Equation 1, an implementation has to be encoded into a binary vector $x = (\boldsymbol{r_1}, ..., \boldsymbol{r_{|V_{\mathsf{a}}|}}, \boldsymbol{e_1}, ..., \boldsymbol{e_{|E_{\mathsf{m}}|}})$. Here, for each resource and mapping edge a binary variable is introduced. For a variable $\boldsymbol{r_i}$ a 1 indicates that the resource $r_i \in V_{\mathsf{a}}$ is part of the allocation $\alpha$ and for a variable $\boldsymbol{e_i}$ the value 1 means that the mapping edge $e_i \in E_{\mathsf{m}}$ is in use and part of the binding $\beta$. Correspondingly, the values 0 indicate that the resource or mapping edge, respectively, are not allocated or used.

If the objective functions are linear, the costs are calculated as a sum of the costs of each used mapping edge and each allocated resource. In this case, the $m$–dimensional objective function of the 0–1 ILP is $f(x) = C^T x$ with $C \in \mathbb{Z}^{n,m}$. The single costs function for the objective $i$ is named $c_i$.

**Objective Function 1** *The overall costs are the sum of the single costs of each used mapping edge and each allocated resource.*

$$\forall i \in \{1, ..., m\} : \quad f_i(x) = \sum_{e \in E_{\mathsf{m}}} c_i(e) \cdot \boldsymbol{e} + \sum_{r \in V_{\mathsf{a}}} c_i(r) \cdot \boldsymbol{r}$$

One vector $x \in X = \{0, 1\}^n$ represent one implementation. Furthermore, the subset $X_f \subseteq X$ contains all feasible implementations. Therefore, the constraints $Ax \leq b$ of the 0–1 ILP have to formulated such that they are satisfied if and only if the implementation $x$ is feasible or $x \in X_f$, respectively. With the

following constraints, one solution of $Ax \leq b$ equals a feasible implementation.

**Constraint 1** *Each used mapping edge has to end at an allocated resource.*

$$\forall e = (p, r) \in E_{\mathsf{m}} : \quad \boldsymbol{r} - \boldsymbol{e} \geq 0$$

**Constraint 2** *For each task $p \in V_{\mathsf{p}}$ exactly one mapping edge has to be activated.*

$$\forall p \in V_{\mathsf{p}} : \quad \sum_{e=(p,r)\in E_{\mathsf{m}}} \boldsymbol{e} = 1$$

**Constraint 3** *Data–dependent tasks have to be mapped to the same or to an adjacent resource.*

$$\forall e = (p, r) \in E_{\mathsf{m}} \wedge (p, \tilde{p}) \in E_{\mathsf{p}} : \quad -\boldsymbol{e} + \sum_{\substack{\tilde{e}=(\tilde{p},\tilde{r})\in E_{\mathsf{m}}: \\ r=\tilde{r}\vee(r,\tilde{r})\in E_{\mathsf{a}}}} \tilde{\boldsymbol{e}} \geq 0$$

A scheme for the composition of additional system–constraints as well as the conversion for non–linear constraints into linear constraints is stated in [11].

### *B. Heuristic Multi–Objective PB Solver* — HPB

A common PB solver is only able to solve single–objective problems. Hence, the PB solver cannot find all Pareto–optimal solutions of general multi–objective problems. Instead, common MOEA approaches are applicable, but, since they are directly varying the implementation, they tend to find many infeasible implementations. Repairing strategies and punishing functions are used to force the MOEA to stay in the feasible search space. But particularly, if the problem has just a few feasible solutions, the MOEA is more focused on searching a feasible implementation than optimizing the objectives.

As a remedy, we will propose a sophisticated heuristic strategy that incorporates a common DPLL [15] based PB solver (cf. Algorithm 1). This heuristic optimizes multi–objective 0–1 ILPs with also non–linear objective functions and design space exploration problems, respectively. In particular, the PB solver is used to force a common heuristic approach, like a MOEA, to stay in the feasible search space allowing to focus merely on the optimization of the objectives:

With the defined constraints from Section A and without any objective function, the PB solver will find one specific feasible implementation for a given design space exploration problem. In fact, the branching strategy of the search process of the PB solver has a huge impact on which implementation is found. Our branching strategy is guided by two vectors $\rho \in \mathbb{R}^n$ and $\sigma \in \{0, 1\}^n$. The optimization algorithm now varies the vectors $\rho$ and $\sigma$ of the branching of the PB solver. Instead of varying the implementation as common design space exploration approaches, this will produce only feasible implementations. Hence, this ensures a good convergence to the optimal solutions.

Here, we will use an MOEA for the variation of $\rho$ and $\sigma$. The fitness estimation of an *individual* $(\rho, \sigma)$ is done by a two–step procedure:

1. Decoding: Using Algorithm 1 and the branching strategy $(\rho, \sigma)$ a feasible implementation $x$ is found.

2. Evaluating: The found implementation $x$ is evaluated by the objective function $f(x)$. At this, the objective function can obviously be also non–linear.

Now, the MOEA tries to improve the values $\rho$ and $\sigma$ with respect to the objective values. In the MOEA, the PB solver is subsequently restarted with varied branching strategies and at the same time feasible implementations $x$ are found and improved.

In the following, the used DPLL based backtracking algorithm is explained.

**Algorithm 1** Search algorithm for a single feasible solution based on a DPLL backtracking algorithm, cf. [8].

```
1: while true do
2:     branch(ρ, σ)
3:     if CONFLICT then
4:         backtrack()
5:     else if SATISFIED then
6:         return x
7:     end if
8: end while
```

This algorithm efficiently searches for an implementation that fulfills all constraints, cf. [10, 12]:

Starting with completely unassigned variables, the operation $branch(\rho, \sigma)$ chooses an unassigned variable and assigns it a value. The rules which variable is chosen and value is assigned is called *decision strategy*. Our decision strategy is simply guided by the vectors $\rho \in \mathbb{R}^n$ and $\sigma \in \{0, 1\}^n$. Unassigned variables $x_i$ with the highest value $\rho_i$ are prioritized and are set to the value $\sigma_i$.

The branch operation recognizes if any variable assignment is required to keep the constraints satisfiable or a *conflict* (*CONFLICT*) occurred. One has to keep in mind that every single constraint has to be satisfied in order to find a feasible solution. Therefore, one decision can cause several necessary assignments, the *implications*. If an implication of the same variable occurs to 0 and 1, a conflict is recognized and analyzed such that a backtracking is triggered, i.e., variable assignments are reverted. The backtracking is able to recognize unsatisfiable problems if the first decision was already tested in both ways 0 and 1 and the algorithm is aborted since there exists not a single feasible solution. If there exists at least one feasible solution, the algorithm proceeds until all variables have an assignment (*SATISFIED*) and one feasible solution is found, respectively.

For a detailed theoretical analysis of the HPB confer [16].

### C. Complete Multi–Objective PB Solver — CPB

If the objective functions are linear ($f(x) = C^T x$) it is possible to solve these problems with a complete multi–objective PB solver. Here, we will provide an approach as an extension of the DPLL backtracking algorithm [15]. Note that instead of an MOEA approach, this algorithm is able to find all Pareto–optimal solutions and prove their optimality as well.

It is obvious that a common PB solver cannot be simply adapted to solve multi–objective PB problems as all constraints are joined by a logical *AND*. Instead, we will propose a modification of the search algorithm as it is used in DPLL based PB solvers. The DPLL algorithm is used to stay in the feasible search space $X_f$ and obtain only feasible solutions. At the same time, the search space is pruned such that the found solutions are optimized throughout the search process. Hence, each solution corresponds a feasible implementation. This depth–first search is given in Algorithm 2.

**Algorithm 2** Algorithm for multi–objective optimization of 0–1 ILPs based on a DPLL backtracking algorithm.

```
1:  A = {}
2:  while true do
3:      branch(ρ, σ)
4:      if CONFLICT then
5:          backtrack()
6:      else if SATISFIED ∧ ∀a ∈ A : ¬(a ⪰ C^T x) then
7:          y = C^T x
8:          A = x ∪ {a | a ∈ A ∧ ¬(y ⪰ C^T a)}
9:      end if
10:     if ∃ a ∈ A : C^T a ⪰ (f_1(x̃), ..., f_m(x̃))^T then
11:         backtrack() // 'to the most recent decision tried not both ways'
12:     end if
13: end while
```

The archive $A$ is holding the set of non–dominated solutions (line 1). A dominated solution is worse or equal in all objectives compared to a second solution, i.e., $a$ dominates $b$ ($a \succeq b$) if $\forall i \in \{1, ..., m\} : f_i(a) \leq f_i(b)$. The archive is filled and updated throughout the backtracking process until the algorithm aborts. Now, the archive contains the optimal non–dominated solutions or the Pareto–optimal implementations, respectively.

Line 3 to 5 is identical to Algorithm 1. It ensures that the search process stays in the valid search space $X_f$. In case that all variables have an assignment (*SATISFIED*) and the current solution is not dominated by any solution inside the archive (line 6), it is added to the archive. At the same time all solutions inside the archive that are dominated by the new solution are removed (line 8).

For this algorithm, it is crucial that a backtracking is also triggered if a partial solution is recognized to be dominated by some solutions in the archive independently of its completion. This operation prunes the search space and prevents that the algorithm equals an enumeration of the feasible solutions in $X_f$. Hence, a lower bound for each objective function has to be calculated and compared for domination with the archive (line 10). The lower bounds for the objective functions for a partial solution are calculated separately in each dimension, i.e., a lower bound vector is calculated by $(f_1(\tilde{x}), ..., f_m(\tilde{x}))^T$. Therefore, the vector $\tilde{x}$ contains the values of the assigned variables and for unassigned variables a 0 (1) is used if the corresponding coefficient $C_{ij}$ of the objective function $f_i$ is positive (negative). The backtracking will take place to the level of the most recent decision that was not tried in both ways 0 and 1 unless this level is lower than 0 what causes an abort of the algorithm (line 11).

The used decision strategy is crucial for the success of this algorithm. It is obvious that with good solutions early in the search process and an accurate lower bound calculation, large parts of the search space can be pruned. A good approach is a decision strategy that is guided by the coefficients of the objective functions: Focusing on a single–objective problem, variables with a big corresponding coefficient should be favored by the decision strategy to increase the accuracy of the calculated lower bound. This takes place as only variables with small coefficients will be unassigned later in the search process. Moreover, it is desirable to obtain good solutions early in the search process and, as a minimization problem is given, the favored decision phase for a variable with a positive (negative) coefficient should be 0 (1). For multi–objective problems, a more sophisticated decision strategy is needed because variables have different effects in different objective functions. We will propose a static decision strategy for the vectors $\rho$ and $\sigma$ based on distribution functions:

$$\forall i \in \{1, ..., n\} \; : \; \rho_i = |v_i| \qquad \sigma_i = \lceil sign(v_i) \rceil \qquad \text{with } v_i = \sum_{j=1}^{m} F_j(|C_{ij}|) sign(C_{ij})$$

For each dimension a distribution function $F_j : \mathbb{N} \rightarrow [0, 1]$ is approximated by the absolute values of all coefficients $C_{ij}$. We will use a uniform distribution between 0 and the highest value of each dimensions coefficient.

For a comparison of different complete multi–objective PB solvers cf. [17].

## IV. Experimental Results

In the following, the proposed complete multi–objective PB solver is termed *CPB*, the heuristic PB solver *HPB*. The introduced methodologies based on 0–1 ILPs are compared with a common method for design space exploration based on MOEAs that is termed *EA*. The *EA* is described in [18] and is based on a data structure with priority lists that allows a local repair strategy. Moreover, infeasible solution are deteriorated by a punishing function. The used MOEA for the common method *EA* and the heuristic strategy *HPB* is the *elitist SPEA2* algorithm [19]. The population size is 100 with 25 parents and offspring for each generation. The mutation rate was set to $p = \frac{1}{|V_a|+|E_m|}$. The crossover of the real number values is based on the *Simulated Binary Crossover* operator [20] followed by a mutation by adding a number from the the natural

| test case | $\lvert V_\mathsf{p}\rvert$ | $\lvert V_\mathsf{a}\rvert$ | $\lvert E_\mathsf{m}\rvert$ |
|---|---|---|---|
| *f40/m40* | 40 | 20 | 160 |
| *f60/m60* | 60 | 30 | 270 |
| *f80/m80* | 80 | 40 | 400 |
| *f100/m100* | 100 | 50 | 500 |
| *f120/m120* | 120 | 60 | 660 |

TABLE I
TEST CASES WITH THE NUMBER OF PROCESSES $|V_\mathsf{p}|$, THE NUMBER OF RESOURCES $|V_\mathsf{a}|$, AND THE NUMBER OF MAPPING EDGES $|E_\mathsf{m}|$.

distribution $\mathcal{N}(0, p)$. All experiments were carried out on a Intel Pentium 4 3.20 GHz machine with 1 GB RAM.

In order to evaluate the quality of the methods, we use the $\epsilon$–dominance [14] criterion which is a measurement used to specify the convergence of multi–objective optimization methods to the front of Pareto–optimal solutions. The $\epsilon$–dominance calculates the relation of a set of solutions $A$ to the set of the Pareto–optimal solutions $B$.

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

Thus, the $\epsilon$–dominance is the smallest value $\epsilon$ that a set of Pareto–optimal solutions has to be scaled with in order to be dominated by the set $A$. If the Pareto–optimal solutions could not be estimated, the set of the non–dominated solutions over all runs on a problem instance was used as reference set $B$. The scaling is normalized in a way that the value of $\mathcal{D}_\epsilon(A, B)$ lies between 1 and 2. In the following, we will use the reciprocal value such that a high value is aspired with 1 being the optimal value.

Synthetic test cases with different sizes were generated and are stated in Table I. For all problem sizes, test cases with only a few as well as many feasible implementations were achieved by varying the number of possible interconnections $E_\mathsf{a}$ in the architecture graph. For each test case, 10 instances were generated and the methods were started 10 times such that a meaningful average value was calculated. The objective functions are the area and power–consumption that are both linear such that all three methods could be used.

Figure 2 shows the results. For the heuristic methods *HPB* and *EA* the runtime is 120 seconds. The *CPB* method is superior on small test cases and also if the search space has just a few feasible implementations. The *CPB* delivers all optimal implementations for *f40* (0.2 sec.), *m40* (0.6 sec.), *f60* (3.9 sec.), *m60* (4.8 sec.), and *f80* (36.6 sec.) in a short time. At the same time the method proves optimality of these solutions. On all other test cases the *CPB* was interrupted after 120 seconds. Our heuristic *HPB* outperforms the common method *EA* on all test cases and compared to the *CPB*, it is near–optimal on the small test cases. The results are also indicators for the behavior on problems with non–linear objective functions for the *HPB* and *EA* methods since both methods are driven by an MOEA that does not make any assumption on the objective functions.

Finally, we studied an industrial example from the automotive area to compare our new methodologies with common

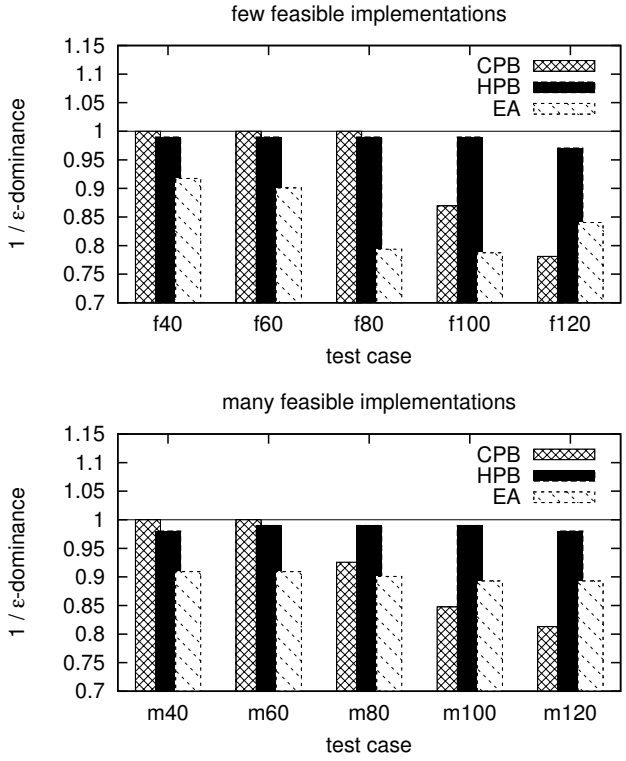


Fig. 2. Quality estimation of the found implementations for the synthetic test cases based on the reciprocal $\epsilon$–dominance criterion. A high value is aspired, the optimal value is 1. All methods were interrupted after 120 seconds.

strategies. The *adaptive light control* (ALC) is a large automotive design problem and consists of 234 process, 1103 resources and 1851 mappings edges. This leads to approximately $2^{375}$ possible bindings. All algorithms were interrupted after 600 seconds.

Firstly, the optimized objectives are area and power–consumption. Again, linear models were used for these two objectives such that the *CPB* could be used. Figure 3 shows the convergence of the used methods. For the heuristic strategies, the average over 10 runs was calculated. The quality of the implementation of the *CPB* and *HPB* methods are nearly equal after 600 seconds and superior to the common method *EA*. Secondly, a third objective, the reliability of the system, was introduced, cf. [21]. This objective is not linear or linearizable. That means, the *CPB* method is not applicable and only the heuristic strategies *HPB* and *EA* are compared. As expected, Figure 3 shows that our heuristic approach *HPB* is superior to the *EA* analogous to the linear objective test cases.

## V. CONCLUSIONS

In this paper, we have formalized the design space exploration problem as a multi–objective 0–1 ILP. In order to solve this multi–objective optimization problem, a novel multi–objective PB solver and a heuristic based on a PB solver were proposed. Synthetic as well as real world problems have been

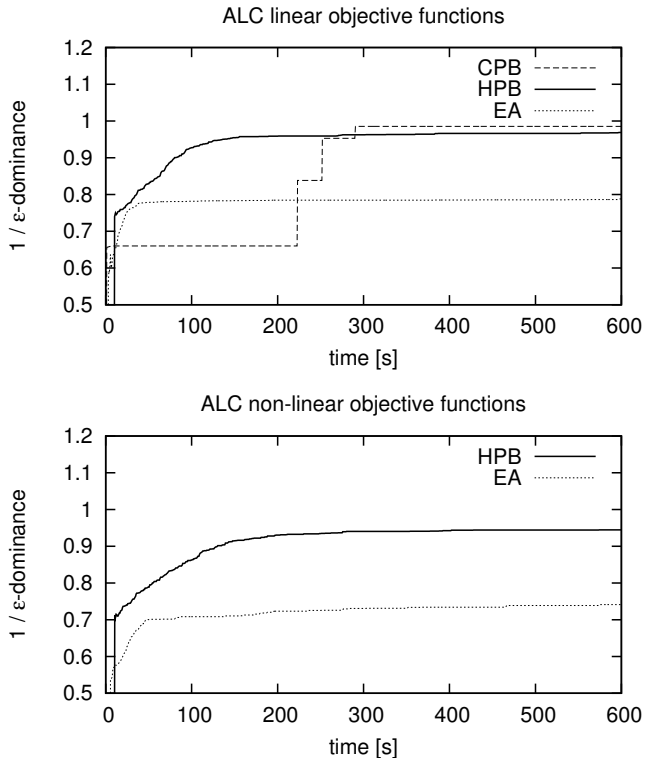


Fig. 3. Quality estimation of the found implementations for the adaptive light control case study based on the reciprocal $\epsilon$–dominance criterion. A high value is aspired, the optimal value is 1.

tested, showing the effectiveness of our proposed design space exploration. For small problems with linear objective functions, the multi–objective PB solver is able to deliver the optimal implementations in a short time whereas it is also proved that these solutions are optimal. For large problems and those with non–linear objective functions our heuristic approach is applicable and outperforms common MOEA search strategies on all test cases.

## REFERENCES

[1] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration, the VLSI Journal, Elsevier*, vol. 38, no. 2, pp. 131–183, December 2004.

[2] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 358–374, 2006.

[3] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems," in *Proceedings of ASAP '04*, 2004, pp. 28–40.

[4] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "Design Space Exploration of Network Processor Architectures," *Network Processor Design: Issues and Practices*, vol. 1, pp. 55–89, Oct. 2002.

[5] S. Neema, "System Level Synthesis of Adaptive Computing Systems," Ph.D. dissertation, Vanderbilt University, Nashville, Tennessee, May 2001.

[6] R. Niemann and P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 165–193, 1997.

[7] N. Eén and N. Sörensson, "An extensible sat-solver." in *Proceedings of SAT '03*, 2003, pp. 502–518.

[8] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *Proceedings of DAC '01*, 2001, pp. 530–535.

[9] J. P. M. Silva and K. A. Sakallah, "Grasp - a new search algorithm for satisfiability," in *Proceedings of ICCAD '96*, 1996, pp. 220–227.

[10] D. Chai and A. Kuehlmann, "A fast pseudo-boolean constraint solver," in *Proceedings of DAC '03*, 2003, pp. 830–835.

[11] N. Eén and N. Sörensson, "Translating Pseudo-Boolean Constraints into SAT," *Journal on Satisfiability, Boolean Moelding and Computation*, vol. 2, pp. 1–25, 2006.

[12] H. M. Sheini and K. A. Sakallah, "Pueblo: A modern pseudo-boolean sat solver," in *Proceedings of DATE '05*, 2005, pp. 684–685.

[13] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Generic ilp versus specialized 0-1 ilp: an update," in *Proceedings of ICCAD '02*, 2002, pp. 450–457.

[14] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review." *IEEE Trans. Evol. Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[15] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Comm. of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[16] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems," in *Proceedings CEC '07*, 2007, pp. 935–942.

[17] ——, "Solving Multiobjective Pseudo-Boolean Problems," in *Proceedings of SAT '07*, 2007, pp. 56–69.

[18] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using Evolutionary Algorithms," *J. Design Automation for Embedded Systems*, vol. 3, no. 1, pp. 23–58, 1998.

[19] 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*, 2002, pp. 19–26.

[20] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.

[21] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-Aware System Synthesis," in *Proceedings of DATE '07*, 2007, pp. 409–414.