Solving Multi-Objective Pseudo-Boolean Problems*

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich

Hardware-Software-Co-Design
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz, glass, haubelt, teich}@cs.fau.de

Abstract. Integer Linear Programs are widely used in areas such as routing problems, scheduling analysis and optimization, logic synthesis, and partitioning problems. As many of these problems have a Boolean nature, i.e., the variables are restricted to 0 and 1, so called *Pseudo-Boolean solvers* have been proposed. They are mostly based on SAT solvers which took continuous improvements over the past years. However, Pseudo-Boolean solvers are only able to optimize a single linear function while fulfilling several constraints. Unfortunately many real-world optimization problems have multiple objective functions which are often conflicting and have to be optimized simultaneously, resulting in general in a set of optimal solutions. As a consequence, a single-objective Pseudo-Boolean solver will not be able to find this set of optimal solutions. As a remedy, we propose three different algorithms for solving multi-objective Pseudo-Boolean problems. Our experimental results will show the applicability of these algorithms on the basis of several test cases.

1 Introduction

Solving *0-1 Integer Linear Programs* (0-1 ILP) came to the field of vision over the past years. This problem class is a special case of *Integer Linear Programs* (ILP) and is also termed as *Pseudo-Boolean* (PB) [1]. In particular a Pseudo-Boolean problem is an optimization problem with a linear objective function and a set of linear constraints in which the coefficients are integers and the variables are restricted to 0 and 1. Despite the restriction of the variables to Boolean values the expressiveness is equal to ILPs which can be formulated as Pseudo-Boolean problems by using a binary encoding.

The Boolean nature of Pseudo-Boolean problems is connecting these strongly to the *Satisfiability problem* (SAT) in conjunctive normal form [2]. The Satisfiability problem can easily be converted to a Pseudo-Boolean problem with an empty objective function in which for each clause a greater-zero constraint is added. The 0-1 Integer Linear Programming is, in fact, one of KARP'S 21 NP-complete problems [3]. On the other hand, converting efficiently PB constraints into clauses is a non-trivial problem that can result in an exponential number of clauses.

There are several PB solvers that are borrowing techniques from state-of-the-art SAT solvers which became essential in the field of *Electronic Design Automation* [4]. These specialized PB solvers are based on the DPLL backtracking algorithm [5] and

* Supported in part by the German Science Foundation (DFG), SFB 694

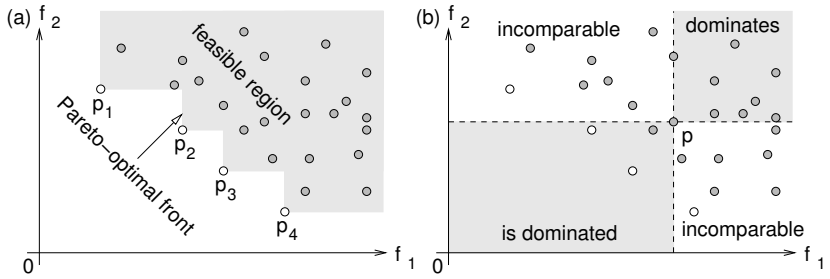

Fig. 1. Objective space showing (a) a Pareto-optimal front of solutions and (b) a solution p and the areas with dominating, non-dominating and incomparable solutions.

benefit from the improvements on the field of SAT-solving of the recent years like the non-chronological backtracking [6], watched literals [7], or an efficient conflict learning scheme [8]. As a matter of fact it is validated that specialized PB solvers are superior to generic ILPs, mostly if the underlying problem has a Boolean nature [9].

PB solvers have their applications among many real-world applications like routing problems, scheduling analysis and optimization, logic and system level synthesis, and partitioning problems. Some of these applications like the problem of system level synthesis [10] can contain more than one objective function, e.g., if the system is optimized by its power consumption, area usage, and the monetary costs. In the case of multi-objective optimization the goal is not to find optimal solutions corresponding to each objective function, but to find the set of optimal solutions the so called Pareto-optimal solutions. A solution is called Pareto-optimal if there exists no other solution that is better or equal in all objectives and at least better in one objective, i.e., no other solution *dominates* the Pareto-optimal one. As the search space in Pseudo-Boolean problems is finite the number of Pareto-optimal solutions is also finite. Figure 1(a) illustrates the Pareto-optimal solutions of a problem with two objective functions. A PB solver optimizes at most one objective function and will not find these Pareto-optimal solutions as preference-based approaches do not find the trade-off solutions. In the case of system level synthesis a designer is interested in the full set of Pareto-optimal solutions containing the trade-off solutions to make an appropriate choice for one implementation.

This paper is dedicated to the multi-objective Pseudo-Boolean problem in which we propose three different algorithms for solving multi-objective Pseudo-Boolean problems and compare them on the basis of several test cases. The first algorithm is an iterative search with a common PB solver by restricting the search space by upper bounds. The second algorithm extends a DPLL backtracking algorithm such that it sifts through the valid search space and at the same time prunes evidently not optimal solutions. The third algorithm is using a translation into the Satisfiability problem such that a common SAT solver finds one solution that fulfills the constraints. To ensure a convergence to the Pareto-optimal solutions, the found and dominated solutions are excluded from the ongoing search by appending additional clauses.

The rest of the paper is organized as follows: Section 2 gives a short introduction to the functionality of modern PB solvers, and Section 3 will formally state the problem

this paper is dedicated to. In Section 4 the three algorithms for solving multi-objective Pseudo-Boolean problem are presented. The comparison of the algorithms on the basis of several experimental results is made in Section 5, before we conclude the paper in Section 6.

2 Specialized PB Solvers

Mathematically a Pseudo-Boolean problem is defined as[1]

$$min\{c^T x \mid Ax \leq b\}$$

with $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m,n}$, $b \in \mathbb{Z}^m$, and $x \in \{0,1\}^n$. The objective function is given as the linear function $c^T x$, whereas the constraints that are linear equalities and inequalities are summarized in $Ax \leq b$. The goal is to find one optimal solution x for which the objective function is minimal.

Specialized PB solvers are based on a backtracking search algorithm similar to modern SAT solvers. The algorithm starts by searching for a solution that fulfills all constraints. If there exists a solution x the objective function is calculated and a '$<$' constraints is added with left hand side the objective function and right hand side the calculated objective value. This procedure is carried out iteratively and ensures the convergence to the optimal value. If the constraints are not satisfiable the last found solution is the optimal solution.

These specialized PB solvers are divided into two categories: First are enhanced SAT solvers that beside clauses in the conjunction normal form also support natively PB constraints, e.g., PBS [9], PUEBLO [11], or GALENA [12]. The second category are PB solvers which translate the PB constraints into clauses such that a common SAT solver is used to find a solution. By introducing additional variables an exponential number of resulting clauses is prevented. The proceeding as described in [13] is two-staged and is implemented in the PB solver MINISAT+ [13, 14]. Each PB constraint is first converted into a hardware circuit by using BDDs, Adders, or Sorters. The resulting hardware circuits are then converted linearly into a set of clauses by using the TSEITIN-transformation [15] that introduces additional variables.

3 Problem Formulation

Extending a 0-1 ILP for multiple objective functions results in a new problem class. Mathematically we define a multi-objective Pseudo-Boolean problem as

$$min\{C^T x \mid Ax \leq b\}$$

with $C \in \mathbb{Z}^{n,z}$, $A \in \mathbb{Z}^{m,n}$, $b \in \mathbb{Z}^m$, and $x \in \{0,1\}^n$. We are considering an optimization problem with z objective functions which, without loss of generality, all have to be minimized. The objective vectors are calculated by $f(x) = C^T x$ with $C = (c_1, ..., c_z)$ where a single objective function is $f_i(x) = c_i^T x$ with $i \in \{1, ..., z\}$.

[1] Maximization problems can be converted to a minimization by negating the objective function.

The optimization for a multi-objective problem is not a search for a single optimal objective value but instead for the set of Pareto-optimal solutions $X_p \subseteq X_f$ or the Pareto-optimal front $Y_p = \{f(x)|x \in X_p\}$, respectively. The valid search space X_f is containing all solutions that are fulfilling the constraints $Ax \leq b$, we are also speaking of *feasible* solutions. A solution $x_p \in X_p$ is said to be Pareto-optimal if its objective vector $f(x_p)$ is not dominated by any other objective vector $f(x)$ with $x \in X_f$, cf. Definition 1.

Relating to Definition 1 the terms for Pareto dominance are applied to the objective vector or the solution vectors, respectively. The solution vectors are termed by Definition 1 with respect to their calculated objective vectors, e.g., x dominates $\tilde{x}$ if and only if $f(x) \succ f(\tilde{x})$.

Definition 1 (Pareto dominance (cf. [16])). *For any two objective vectors a and b,*

$$\begin{aligned} a \succ\succ b &\quad (a \textit{ strictly dominates } b) \textit{ if } \forall i : a_i < b_i \\ a \succ b &\quad (a \textit{ dominates } b) \textit{ if } \forall i : a_i \leq b_i \wedge \exists j : a_j < b_j \\ a \succeq b &\quad (a \textit{ weakly dominates } b) \textit{ if } \forall i : a_i \leq b_i \\ a \parallel b &\quad (a \textit{ is incomparable to } b) \textit{ if } \exists i, j : a_i > b_i \wedge a_j < b_j. \end{aligned}$$

In general, a common iteratively working PB solver as described in Section 2 will not be able to find the set of Pareto-optimal solutions of the multi-objective Pseudo-Boolean problem. By adding a PB constraint each time a solution is found these PB solvers are restricting the search space such that weakly dominated solutions are not found in the ongoing search. Using this approach for a multi-objective problem it would be necessary to add a PB constraint for each objective. These constraints would have to be joined by a logical OR as an improvement in only one dimension is mandatory to find the next not dominated solution. It is obvious that a common PB solver can not be simply adapted to solve multi-objective Pseudo-Boolean problems as all constraints are joined by a logical AND unless additional variables are added for each iteration.

4 Algorithms

4.1 Algorithm 1

The first algorithm, given in Algorithm 1, enables the usage of a common PB solver to find all Pareto-optimal solutions iteratively. In order to find these solutions the upper bounds for the objective functions are set adequately as constraints for the PB solver.

The algorithm fills the *archive* A with the non-dominated solutions. These are solutions that are not dominated by any other solution that was found during the ongoing search. The property of non-dominance proves that there is no solution inside the archive that is better or equal in all objectives compared to another solution in the archive, cf. Definition 1 on *weak domination*. The set D contains the *domains* where a domain is a vector of the upper bounds for the objective functions. The algorithm starts with an empty archive (line 1) and the set of domains containing an initial vector of the length z, where the initial values are set to ∞ (line2) corresponding the whole search space.

Algorithm 1: Algorithm for multi-objective optimization of Pseudo-Boolean problems based on the iterative usage of a common PB solver.

```
1  A = {}
2  D = {(max_1, ..., max_z)}
3  while |D| > 0 do
4      choose h ∈ D
5      min{0 | Ax ≤ b ∧ C^T x ≤ h}
6      if UNSATISFIABLE then
7          D = D\h
8      else
9          y = C^T x
10         A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
11         foreach e ∈ D ∧ y ⪰ e do
12             D = D\e
13             for i ∈ {1, ..., z} do
14                 D = D ∪ (e_1, ..., y_i − 1, ..., e_z)
15             end
16         end
17         foreach e, ẽ ∈ D ∧ e ≢ ẽ ∧ e ⪰ ẽ do
18             D = D\e
19         end
20     end
21 end
```

While the set of domains is not empty (line 3) one domain is chosen randomly (line 4) and a solution that both fulfills the constraints and is located inside the selected domain is searched with a common PB solver (line 5). At the same time the objective function is empty. If the PB solver does not find a solution, the current domain is removed from the set of domains since there is no feasible solution inside (line 6,7). In case that a solution is found in the domain, the objective vector is calculated (line 9) and the archive is updated such that it just contains non-dominated solutions (line 10).

Each domain that contains the found solution needs to be split (line 11). This is done by removing that domain and adding new domains whereas an improvement in at least one dimension has to be achieved (line 12-15). Concluding, the set D is cleaned up in which sub-domains of other domains are removed (line 17-19).

The advantage of this methodology is the independence of the used PB solver. Any common PB solver can be extended to a multi-objective PB solver by this algorithm. Moreover, this straightforward method is not restricted to Pseudo-Boolean problems, by using an ILP solver it is possible to solve also multi-objective Integer Linear Programs.

4.2 Algorithm 2

The second method is an extension of the DPLL backtracking algorithm. More precisely, it is a modification of the DPLL algorithm as it is used in specialized PB solvers. Thereby, it is not important which category of specialized PB solver is used, the ones

that natively support PB constraint or the other category that translates the PB constraints completely into clauses. The DPLL algorithm is used to stay in the valid search space X_f and obtain only feasible solutions. For the first category of PB solvers, the PB constraints are given directly, while for the second category the PB constraints are converted into clauses. The complete algorithm is given in Algorithm 2.

Algorithm 2: Algorithm for multi-objective optimization of Pseudo-Boolean problems based on a DPLL backtracking algorithm.

```
1  A = {}
2  while true do
3      branch()
4      status = deduce()
5      if status ==CONFLICT then
6          blevel = analyze_conflict()
7          if blevel < 0 then
8              break
9          else
10             backtrack(blevel)
11         end
12     else if status ==SATISFIABLE ∧ ∀a ∈ A : ¬(a ≽ C^T x) then
13         y = C^T x
14         A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
15     end
16     if ∃ a ∈ A : C^T a ≽ (c_1^T x_1, ..., c_z^T x_z)^T then
17         blevel = 'level of the most recent decision tried not both ways'
18         if blevel < 0 then
19             break
20         else
21             backtrack(blevel)
22         end
23     end
24 end
```

The archive A is holding the set of non-dominated solutions (line 1). The archive is filled and updated throughout the backtracking process until the algorithm aborts and the archive contains the optimal non-dominated solutions, which are the Pareto-optimal solutions.

In a nutshell, line 3 to 11 is identical to a DPLL backtracking algorithm, it ensures that the search process stays in the valid search space X_f: The operation $branch()$ chooses an unassigned variable and assigns it a value. The rules which decide which variable is chosen and value is assigned is called *decision strategy*. The operation $deduce()$ recognizes if any variable assignment is required to keep the constraints satisfiable or a *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 so called *implications*. If an implication of the same variable occurs to 0 and 1, a conflict is recognized and analyzed in $analyze\_conflict()$ such that a backtracking is triggered. If the backtrack level is less than 0, the first decision was already tested in both ways 0 and 1 and the algorithm is aborted.

In case that all variables have an assignment ($decide()$ returns *SATISFIABLE*) and the current solution is not weakly dominated by any solution inside the archive (line 12), it is added to this archive. At the same time all solutions inside the archive which are weakly dominated by the new solution are removed (line 14).

A backtracking is also triggered if a partial solution is recognized to be weakly 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 weak domination with the archive (line 16). 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 $(c_1^T x_1, ..., c_z^T x_z)^T$. Therefore, the vector x_i contains the values of the assigned variables and for unassigned variables a 0 (1) is used if the corresponding coefficient of the vector c_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 17-22).

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 based on distribution functions. For each dimension a distribution function $F_i : \mathbb{N} \rightarrow [0, 1]$ is approximated by the absolute values of the vector containing the coefficients c_i, thus also a normalization of the coefficients is achieved. We will use a uniform distribution between 0 and the highest value of each dimensions coefficient. For instance, it is also possible to sample the values to a normal or any other distribution. With the given distributions a specific value for each variable can be calculated as follows:

$$\forall i = 1, ..., n : \sum_{j=1}^{z} F_i(|C_{ij}|) sign(C_{ij})$$

According to the rules of the single objective problem the decision strategy uses these values as follows: Variables with a high absolute value calculated by this formula are prioritized in the decision strategy. For a positive (negative) value, the decision takes place to 0 (1). This decision strategy will only work properly if the coefficients are distributed with an adequate variance.

In future work we will extend the algorithm by a dynamic variable order, random restarts, and more precise, but on the other hand slower, lower bound estimation strategies [17].

4.3 Algorithm 3

The third algorithm is an extension of a common iteratively working PB solver. As mentioned before the PB constraints in a PB solver are usually joined by a logical AND. Instead, the constrained objective function in multi-objective problems have to be joined by logical ORs. To overcome this restriction we will modify the category of specialized PB solvers which translate the PB constraints into clauses and use a common SAT solver to converge to the optimal value. This category of PB solvers is working two-staged as described in [13] and implemented in the PB solver MINISAT+. In the first step each PB constraint is translated into a hardware circuit. In the second step the hardware circuits are translated into a set of clauses. It it obvious that the clauses that are added to a common SAT solver can not be joined by a logical OR as a SAT solver is expecting a conjunctive normal form. As the constrained objective functions need to be joined by a logical OR each time a solution is found, we connect the hardware circuits by ORs and then translate the full circuit to clauses. The complete algorithm is given in Algorithm 3.

Algorithm 3: Algorithm for multi-objective optimization of Pseudo-Boolean problems based on the translation into the Satisfiability problem.

```
1  A = {}
2  SATSolver.addClauses(toClauses(toCircuit('Ax ≤ b')))
3  while SATSolver.solve()==SATISFIABLE do
4  |   x =SATSolver.x()
5  |   y = C^T x
6  |   A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
7  |   h = false
8  |   foreach i ∈ {1, ..., z} do
9  |   |   h = h ∨ toCircuit('c_i^T x < y_i')
10 |   end
11 |   SATSolver.addClauses(toClauses(h));
12 end
```

Like in the other algorithms the archive A is holding the set of non-dominated solutions (line 1). It is updated throughout the search process and contains the Pareto-optimal solutions when the algorithm terminates.

Primarily, the PB constraints are translated into clauses (line 2). The translation is two-staged: Each PB constraint is translated into a hardware circuit and afterwards the hardware circuits are translated into clauses. The translation of a PB constraint into a hardware circuit is done by using BDDs, Adders, or Sorters, while the translation of one hardware circuit into a set of clauses is done by using the TSEITIN-transformation,

which prevents an exponential number of clauses by introducing additional variables. For a further explanation we strongly recommend [13].

The SAT solver is used iteratively just like in specialized PB solvers to search for non-dominated solutions (line 3). If a non-dominated solution is found, it is added to the archive (line 4-6). Additionally, before the next start of the SAT solver all by this current solution weak dominated solutions have to be excluded from the further search process to guarantee a convergence to the Pareto-optimal solutions. Following Definition 1 of weak dominance this exclusion is done by the formula

$$\overline{(f_1(x) \geq y_1 \wedge ... \wedge f_z(x) \geq y_z)}$$

for the current found solution y. By using DEMORGAN'S law this can also be interpreted as

$$(f_1(x) < y_1 \vee ... \vee f_z(x) < y_z),$$

which are PB constraints connected by logical ORs. Therefore, the two-staged translation is split as following: The PB constraints are translated into hardware circuits and these circuits are connected by an OR-gate to one hardware circuit (line 7-10). This hardware circuit is then translated into a set of clauses which are added to the SAT solver (line 11).

This approach is showing the high versatility of the category of PB solvers which are translating the PB constraints into SAT. The success of this algorithm depends strongly on the translation of the PB constraints and objective functions into clauses, and the performance of the used SAT solver.

5 Experimental Results

For the experimental results the three algorithms were implemented on the basis of the PB solver MINISAT+ [13] or SAT solver MINISAT [14], respectively, which participated very successful in the past SAT Competition [18] and PB Evaluation [19]. Through using the same PB solver as the basis for all three algorithms, we have the chance for a fair comparison on the basis of runtime. To compare the algorithms, we will use several modifications of the famous queens puzzle and synthetic industrial multi-objective problems from the field of system level synthesis [10]. All test cases were carried out on an Intel Pentium 4 3.20 GHz machine with 1 GB RAM. For each handmade test case 10 instances were created and a representative average was calculated. The timeout bound was set to 1800 seconds.

5.1 Queens Puzzle

Problem Statement The common queens puzzle is about putting eight queens on a chessboard such that no pair of queens attacks one another. Though a single solution can be obtained by a construction scheme, finding one solution that fulfills the conditions is a non-trivial problem. The problem can easily be converted into a Satisfiability or Pseudo-Boolean problem, respectively, by introducing one variable for each field defining if a queen is located on it or not, and adding the conditions such that in each

row and column of the chessboard has to be exactly one queen and in each diagonal at most one queen. Moreover, the queens puzzle can be extended to an $n \times n$ chessboard in which n queens have to be put on this chessboard. The advantage of these handmade problems is that the scaling of the problem size is done by a single variable n instead of many variables, as is the case, e.g., in graph problems where nodes and edges are varied.

To achieve representative test cases we will use the n queens puzzle with appropriate objective functions. The objective functions should have the property to be applicable independently on any number of dimensions. For instance the minimal vertex cover problem can not be scaled to a multi-objective problem. Therefore, we will focus on two optimization classes: the weighted costs optimization and the minimal token optimization. These optimization problems can be extended to any number of dimensions.

Weighted Costs Optimization Each field of the chessboard gets a cost that is an integer value. The overall costs are a sum of the costs of the fields where a queen is located on, which equals a linear function that has to be minimized. In our examples we will create the single costs randomly as an integer from a uniform distribution between a lower and an upper bound given as integers. The weighted costs optimization problem is denoted as $w(l, h)$ with the lower bound l and the upper bound h. We will analyze two problem classes namely $w(1, 100)$ which we will refer to as *strongly weighted* and $w(0, 1)$ as *weakly weighted*.

Minimal Token Optimization For each token one variable is introduced. One token exists several times and is distributed randomly on the chessboard. If one field of the chessboard is taken by a queen, the tokens of the field are used which is realized by an implication[2]. The goal is the minimization of the used tokens which is the sum of the variables of the tokens or a linear function with the coefficients 1, respectively. If there are n tokens and one token exists m times and is randomly distributed on the chessboard, we will denote this optimization problem as $t(n, m)$. Instead of some minimization problems where the coefficients are uniquely 1 like the minimal vertex cover, minimal dominating set, or set covering, this problem can be extended to several dimensions by using disjunctive sets of tokens for each dimension.

Analysis of Experimental Results Several combinations of test cases were carried out and summarized in Table 1. We varied the size, the number of objective functions and the sort of objective function. As one can expect, the problem size and number of objectives affect the runtime of all algorithms. In the strongly weighted problems Algorithm 2 is superior to the other algorithms because an appropriate decision strategy could be calculated. In the weakly weighted problems the decision strategy for Algorithm 2 can not be calculated clearly, which leads to a decline of the runtime. In fact, no algorithm is clearly the best in that problem class. In the minimal token problems Algorithm 3

[2] The field x labeled by a token t leads to an implication $(x \rightarrow t)$, a clause $(\overline{x} \vee t)$, or a PB constraint $x - t \leq 0$, respectively.

Problem	Size	Objective functions	Runtime [s]		
			Algorithm 1	Algorithm 2	Algorithm 3
Queens	10×10	w(1,100)	1.95 (0.49)	**0.28 (0.07)**	2.10 (0.28)
Queens	12×12	w(1,100)	37.9 (7.80)	**5.00 (2.07)**	51.9 (25.5)
Queens	14×14	w(1,100)	1800 (0)	**210 (113)**	1229 (441)
Queens	10×10	w(1,100),w(1,100)	11.3 (4.28)	**0.38 (0.03)**	8.36 (1.14)
Queens	12×12	w(1,100),w(1,100)	699 (176)	**16.7 (2.66)**	227 (25.9)
Queens	14×14	w(1,100),w(1,100)	1800 (0)	**1443 (241)**	1800 (0)
Queens	10×10	w(1,100),w(1,100),w(1,100)	92.3 (22.1)	**0.39 (0.02)**	22.7 (4.18)
Queens	12×12	w(1,100),w(1,100),w(1,100)	1800 (0)	**17.9 (2.76)**	468 (44.8)
Queens	14×14	w(1,100),w(1,100),w(1,100)	1800 (0)	1800 (0)	1800 (0)
Queens	12×12	w(0,1)	0.39 (0.07)	**0.08 (0.01)**	0.09 (0.01)
Queens	14×14	w(0,1)	0.68 (0.13)	**0.12 (0.01)**	0.13 (0.01)
Queens	16×16	w(0,1)	1.05 (0.20)	**0.17 (0.01)**	0.19 (0.01)
Queens	12×12	w(0,1),w(0,1)	1.97 (0.81)	0.79 (0.60)	**0.72 (0.37)**
Queens	14×14	w(0,1),w(0,1)	5.80 (3.71)	8.48 (10.7)	**3.47 (3.87)**
Queens	16×16	w(0,1),w(0,1)	**24.6 (28.3)**	62.5 (124)	35.1 (83.6)
Queens	12×12	w(0,1),w(0,1),w(0,1)	14.5 (7.11)	**6.20 (1.70)**	7.18 (3.45)
Queens	14×14	w(0,1),w(0,1),w(0,1)	**131 (135)**	258 (199)	169 (189)
Queens	16×16	w(0,1),w(0,1),w(0,1)	**1197 (629)**	1800 (0)	1224 (716)
Queens	8×8	t(24,8)	0.23 (0.11)	0.23 (0.8)	**0.05 (0.01)**
Queens	10×10	t(30,10)	0.58 (0.13)	11.5 (5.46)	**0.24 (0.04)**
Queens	12×12	t(36,12)	6.10 (2.59)	1800 (0)	**4.20 (1.18)**
Queens	8×8	t(24,8),t(24,8)	0.87 (0.23)	88.0 (20.8)	**0.08 (0.01)**
Queens	10×10	t(30,10),t(30,10)	3.38 (0.80)	1800 (0)	**0.44 (0.03)**
Queens	12×12	t(36,12),t(36,12)	81.7 (15.1)	1800 (0)	**16.1 (1.66)**
Queens	8×8	t(24,8),t(24,8),t(24,8)	1.98 (0.61)	1800 (0)	**0.14 (0.01)**
Queens	10×10	t(30,10),t(30,10),t(30,10)	15.8 (3.35)	1800 (0)	**0.77 (0.03)**
Queens	12×12	t(36,12),t(36,12),t(36,12)	570 (149)	1800 (0)	**22.3 (2.46)**
Queens	10×10	w(1,100),w(0,1)	3.96 (1.51)	**0.35 (0.05)**	3.52 (0.66)
Queens	12×12	w(1,100),w(0,1)	126 (45.7)	**13.9 (2.56)**	97.5 (20.3)
Queens	10×10	w(1,100),t(30,10)	6.45 (1.96)	1800 (0)	**3.52 (0.70)**
Queens	12×12	w(1,100),t(30,10)	284 (79.4)	1800 (0)	**119 (10.1)**
Queens	10×10	w(0,1),t(30,10)	1.72 (0.66)	1800 (0)	**0.33 (0.03)**
Queens	12×12	w(0,1),t(30,10)	26.2 (8.72)	1800 (0)	**10.2 (1.69)**

Table 1. Results on several queen puzzle problems. Given is the size of the chessboard and the used objective functions. The runtime of the algorithms were calculated as an average each with 10 instances in which the variance is given in the brackets.

Problem	Processes	Resources	$\lvert X_f \rvert$	Runtime [s]		
				Algorithm 1	Algorithm 2	Algorithm 3
System level Synthesis	56	25	small	226 (332)	**0.20 (0.08)**	4.11 (3.21)
System level Synthesis	56	25	medium	1121 (721)	**0.57 (0.37)**	29.2 (17.8)
System level Synthesis	56	25	big	1800 (0)	**0.94 (1.17)**	55.6 (64.2)
System level Synthesis	101	50	small	1800 (0)	**845 (673)**	1226 (626)
System level Synthesis	101	50	medium	1800 (0)	1800 (0)	1800 (0)
System level Synthesis	101	50	big	1800 (0)	1800 (0)	1800 (0)
H.264 Video Decoder	68	15	-	5.55	**0.05**	1.65

Table 2. Results on several system level synthesis problems. Given is the number of processes and resources and the number of feasible solutions or size of the valid search space X_f, respectively. The runtime of the algorithms were calculated as an average each with 10 instances in which the variance is given in the brackets.

is superior to the other algorithms. Algorithm 2 is not able to calculate a proper decision strategy as all coefficients are 1 and the algorithm decays to a simple enumeration. Therefore, in all combined problems where at least one objective function is a minimal token optimization Algorithm 2 fails and the best results are provided by Algorithm 3.

5.2 System Level Synthesis

Problem Statement The task of system level synthesis is to bind a set of communicating processes on a set of interconnected resources and generate feasible *implementations* w.r.t. to a correct communication on the given architecture. Corresponding to the objectives, the goal of design space exploration is to find all optimal implementations which satisfy the specification. For a further explanation we refer to [10]. Searching a single feasible implementation can be formulated as a Satisfiability problem [20]. If the objective functions are linear or linearizable, the resulting problem is a multi-objective Pseudo-Boolean problem.

The first test case group consists of graphs with 56 processes and 25 resource nodes. For each process the number of mapping edges varies from 3 to 6. This leads to approximately 2^{117} possible solutions. For the second test case group the number of processes was increased to 101 and resources to 50. The mapping edges per process vary from 4 to 8. That leads to about 2^{256} possible solutions. Additionally, the number of feasible solutions was varied from *small* over *medium* to *big*. This is done by specifying the possibility of a connection between two resources. The optimized objectives were the power consumption and the area usage.

Analysis of Experimental Results The results of the test cases are given in Table 2. As these problems are handling with weighted costs, Algorithm 2 is superior to the other algorithms. The interesting fact is that for a growing number of feasible implementations the complexity is also growing. As typical problems known from real-world applications show hard constrained search spaces containing only a small fraction of feasible solutions [21], this turns out to be advantageous.

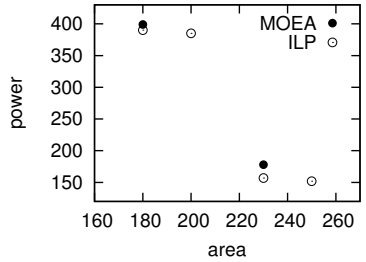

Fig. 2. Pareto-optimal front for the Pareto-optimal solutions and the best non-dominated solutions from a MOEA of the H.264 Video Decoder example. The values area and power are given in abstract units.

H.264 Video Decoder Concluding we will use an industrial system level synthesis example namely a H.264 Video Decoder. The specification graph contains 68 processes, 15 resources and 276 mapping edges what leads to approximately 2^{136} possible solutions. The proposed algorithms are solving this problem easily, the runtimes are stated in Table 2. Moreover, a multi-objective Evolutionary Algorithm (MOEA) that is usually used to solve these problems will not find the Pareto-optimal solutions even after one hour whereas Algorithm 2 needs just a fraction of one second. The Pareto-optimal solutions and the best solutions of the MOEA after one hour of exploration are illustrated in Figure 2.

6 Conclusions

In this paper we have proposed three algorithms for solving multi-objective Pseudo-Boolean problems. Comparing the algorithms on several examples shows that none of these algorithms is generally superior to the others. Instead of that, the success of the methodologies depends on the given problem and the implementation of the algorithm. The algorithms are, in fact, modifications and extensions of common SAT- and PB solvers, respectively. Therefore, improvements on the field of PB-solving and SAT-solving will consequently also lead to a speed up of the proposed algorithms.

Moreover, we have shown that a typical industrial system level synthesis problem, can be solved in a reasonable amount of time. The example of an H.264 Video Decoder was solved in less than a second what, in this particular case, makes the multi-objective Pseudo-Boolean solvers outstanding in comparison to common multi-objective heuristics like Evolutionary Algorithms.

References

1. Barth, P.: A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (1995)

2. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC '71: Proceedings of the third annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1971) 151–158
3. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., eds.: Complexity of Computer Computations. Plenum Press (1972) 85–103
4. Marques-Silva, J.P., Sakallah, K.A.: Boolean satisfiability in electronic design automation. In: DAC '00: Proceedings of the 37th conference on Design automation, New York, NY, USA, ACM Press (2000) 675–680
5. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7) (1962) 394–397
6. Marques-Silva, J.P., Sakallah, K.A.: Grasp - a new search algorithm for satisfiability. In: ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, Washington, DC, USA, IEEE Computer Society (1996) 220–227
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: DAC '01: Proceedings of the 38th conference on Design automation, New York, NY, USA, ACM Press (2001) 530–535
8. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: ICCAD '01: Proc. of the 2001 IEEE/ACM international conference on Computer-aided design, Piscataway, NJ, USA, IEEE Press (2001) 279–285
9. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Generic ilp versus specialized 0-1 ilp: an update. In: ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, New York, NY, USA, ACM Press (2002) 450–457
10. Teich, J., Blickle, T., Thiele, L.: An evolutionary approach to system-level synthesis. In: CODES '97: Proceedings of the 5th International Workshop on Hardware/Software Co-Design, Washington, DC, USA, IEEE Computer Society (1997) 167
11. Sheini, H.M., Sakallah, K.A.: Pueblo: A modern pseudo-boolean sat solver. In: DATE '05: Proc. of the conf. on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2005) 684–685
12. Chai, D., Kuehlmann, A.: A fast pseudo-boolean constraint solver. In: DAC '03: Proc. of the 40th conference on Design automation, New York, NY, USA, ACM Press (2003) 830–835
13. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Moelding and Computation **2** (2006) 1–25
14. Eén, N., Sörensson, N.: An extensible sat-solver. In: Conference on Theory and Application of Satisfiability Testing SAT. (2003) 502–518
15. Tseitin, G.: On the Complexity of Derivations in Propositional Calculus. Studies in Contr. Math. and Math. Logic (1968)
16. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evolutionary Computation **7**(2) (2003) 117–132
17. Manquinho, V.M., Marques-Silva, J.: Effective lower bounding techniques for pseudo-boolean optimization. In: DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2005) 660–665
18. Berre, D.L., Simon, L.: Sat competition 2005. Website (2005) Available online at http://www.satcompetition.org/2005/.
19. Manquinho, V., Roussel, O.: Pseudo boolean evaluation 2006. Website (2006) Available online at http://www.cril.univ-artois.fr/PB06/.
20. Haubelt, C., Teich, J., Feldmann, R., Monien, B.: SAT-Based Techniques in System Design. In Wehn, N., Verkest, D., eds.: Proceedings of Design, Automation and Test in Europe, Munich, Germany, IEEE Computer Society (2003) 1168–1169
21. Deb, K., ed. In: Optimization for Engineering Design. Prentice-Hall of India Pvt.Ltd (1995)