EPiC Series in Computing

Volume 59, 2019, Pages 65–78

Proceedings of Pragmatics of SAT 2015 and 2018

Competitive Sorter-based Encoding of PB-Constraints into SAT

Michał Karpiński[1] and Marek Piotrów[1]

Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland
{karp,mpi}@cs.uni.wroc.pl

Abstract

A Pseudo-Boolean (PB) constraint is a linear inequality constraint over Boolean variables. A popular idea to solve PB-constraints is to transform them to CNFs (via BDDs, adders and sorting networks [5, 11]) and process them using – increasingly improving – state-of-the-art SAT-solvers. Recent research have favored the approach that uses Binary Decision Diagrams (BDDs), which is evidenced by several new constructions and optimizations [2, 21]. We show that encodings based on comparator networks can still be very competitive. We present a system description of a PB-solver based on MiniSat+ [11] which we extended by adding a new construction of selection network called 4-Way Merge Selection Network, with a few optimizations based on other solvers. Experiments show that on many instances of popular benchmarks our technique outperforms other state-of-the-art PB-solvers.

1 Introduction

One of the ways to solve hard decision problems is to reduce them to Boolean satisfiability (SAT) problem and use recently-developed SAT-solvers to find the solution. Using only clauses as a tool to describe such problems can be a big challenge on its own. We can improve this situation by using high-level constraints as a bridge between our problem and SAT.

A *Pseudo-Boolean constraint* (PB-constraint) is a linear inequality with integer coefficients, where variables are over Boolean domain. More formally, PB-constraints are of the form $a_1 l_1 + \cdots + a_n l_n \sim k$, where a_i's and k are integers, l_i's are Boolean literals (that is, variables or their negations) and $\sim$ is a relation from the set $\{<, \leq, =, \geq, >\}$. PB-constraints are more expressive and more compact than clauses to represent some Boolean formulas, especially for optimization problems. PB-constraints are used in many real-life applications, for example, in cumulative scheduling [22], logic synthesis [3] or verification [7].

There have been many approaches for handling PB-constraints in the past, for example, extending existing SAT-solvers to support PB-constraints natively or extending variables' domain to finite sets or unbounded integers. One of the most successful ideas was introduced by Eén and Sorensson [11], where they show how PB-constraints can be handled through translation to SAT without modifying the SAT procedure itself. They implemented techniques based on

binary adders, sorting networks and binary decision diagrams in a tool called MiniSat+. This tool has served as a base for many new solvers and has been extended to test new constructions and optimizations in the field of PB-solving. Similarly, we have developed a system based on MiniSat+ which encodes PB-constraints using the new sorter-based algorithm. We describe this system here and we show experimentally that it is competitive to other state-of-the-art solvers.

1.1 Related work

Recent development in PB-solvers show superiority of encodings based on *Binary Decision Diagrams* (BDDs). The main advantage of BDD-based encodings is that the resulting size of the formula is not dependent on the size of the coefficients of a PB-constraint. Abío et al. [2] show a construction of *Reduced Ordered BDDs* (ROBDDs), which produce arc-consistent, efficient encoding for PB-constraints. Sakai and Nabeshima [21] extend the ROBDD construction to support constraints in band form: $l \leq \langle \text{Linear term} \rangle \leq h$. They also propose an incremental SAT-solving strategy of binary/alternative search for minimizing values of a given goal function and their experiments show significant speed-up in SAT-solver runtime.

Another approach to handle large coefficients in PB-constraint is to decompose the constraint into a number of interconnected sorting networks, where each sorter represents an adder of digits in a mixed radix base – this was implemented in MiniSat+ [11]. This construction requires the good choice of a mixed radix base and the goal is to find a base which minimizes the size of the sorting networks. Codish et al. [8, 12] show that solving optimal base problem – the problem of finding an efficient representation for a given collection of positive integers – can lead to smaller encodings of PB-constraints.

A particular considered case of Pseudo-Boolean constraints is the one of *Cardinality Constraints* (CCs), where all coefficients a_i are equal to 1. In this context, multiple papers showed that encodings based on comparator networks give very good results. Codish and Zazon-Ivry [9] introduced pairwise selection networks that were later improved in [14]. Abío, Asín et al. [1, 4] defined encodings that implemented selection networks based on the odd-even sorting networks by Batcher [6]. Recently, we used generalized model of comparator network, where we not only use sorters of order 2, but also m-sorters, for arbitrary natural number m. We fixed $m = 4$ and proposed encoding based on 4-Odd-Even Selection Network [15]. In our construction we use the idea of the multiway merge sorting networks by Lee and Batcher [17] that generalizes the technique of odd-even sorting ones by merging simultaneously more than two subsequences. The new selection network merges 4 subsequences in that way. Based on this construction, we can encode more efficiently comparators in the combine phase of the network: instead of encoding each comparator separately by 3 clauses and 2 additional variables, we propose an encoding scheme that requires 5 clauses and 2 variables on average for each pair of comparators. The experiments show that this method is very good in practice for solving CCs. All constructions mentioned in this paragraph are arc-consistent [13].

1.2 Our contribution

To replicate the success of our algorithm [15] in the field of PB-solving, we implemented the 4-Odd-Even Selection Network in MiniSat+ and removed the 2-Odd-Even Sorting Network from the original implementation [11]. In [15] we show a top-down, divide-and-conquer algorithm for constructing 4-Odd-Even Selection Network. The difference in our new implementation is that we build our network in a bottom-up manner, which results in the easier and cleaner implementation.

We apply a number of optimization techniques in our solver, some based on the work of other researchers. In particular, we use optimal base searching algorithm based on the work of Codish et al. [8] and ROBDD structure [2] instead of BDDs for one of the encodings in MiniSat+. We also substitute sequential search of minimal value of the goal function in optimization problems with binary search similarly to Sakai and Nabeshima [21]. We use COMiniSatPS [19] by Chanseok Oh as the underlying SAT-solver, as it has been observed to perform better than the original MiniSat [10] for many instances.

We experimentally compare our solver with other state-of-the-art general constraints solvers like PBLib [20] and NaPS [21] to prove that our techniques are good in practice. Since more then a decade there have been organized a series of Pseudo-Boolean Evaluations [18] which aim to assess the state-of-the-art in the field of PB-solvers. We use the competition problems from the PB 2016 Competition as a benchmark for the solver proposed in this paper.

1.3 Structure of the paper

In Section 2 we briefly introduce how MiniSat+ uses sorting networks to translate PB-constraint into SAT and we describe how we have extended MiniSat+ using new selection algorithm and other optimizations. In Section 3 we present experimental results and we leave concluding remarks in Section 4.

2 System Description

The main tool in our solver is a comparator network. Traditionally comparator networks are presented as circuits that receive n inputs and permute them using comparators (2-sorters) connected by "wires". Each comparator has two inputs and two outputs. The "upper" output is the maximum of inputs, and "lower" one is the minimum. The standard definitions and properties of them can be found, for example, in [16]. The only difference is that we assume that the output of any sorting operation or comparator is in a non-increasing order.

2.1 4-Way Merge Selection Network

The MiniSat+ uses Batcher's original construction [6] – the 2-Odd-Even Sorting Network. What we propose to use is a *selection network*. A selection network of order (n, k) is a comparator network such that for any 0-1 input of length n outputs its k largest elements. Those k elements must also be sorted in order to easily assert the given constraint, by asserting only the k-th output.

The main building block of our encoding is a direct selection network, which is a certain generalization of a comparator. Encoding of the direct selection network of order (n, k) with inputs $\langle x_1, \ldots, x_n \rangle$ and outputs $\langle y_1, \ldots, y_k \rangle$ is the set of clauses $\{x_{i_1} \wedge \cdots \wedge x_{i_p} \Rightarrow y_p : 1 \leq p \leq k, 1 \leq i_1 < \cdots < i_p \leq n\}$. The direct n-sorter is a direct selector of order (n, n), therefore we need n auxiliary variables and $2^n - 1$ clauses to encode it. This shows that n should be small in order to avoid exponential blowup in the number of clauses.

It has already been observed that using selection networks instead of sorting networks is more efficient for the encoding of constraints [9]. This fact has been successfully used in encoding CCs. We now apply this technique to PB-constraints. Here we describe the algorithm for constructing a 4-Odd-Even Selection Network. Proof of correctness and analysis can be found in our previous paper [15]. We extended our construction in a way that we mixed them with the direct encoding for small values of parameters n and k – the technique which was first

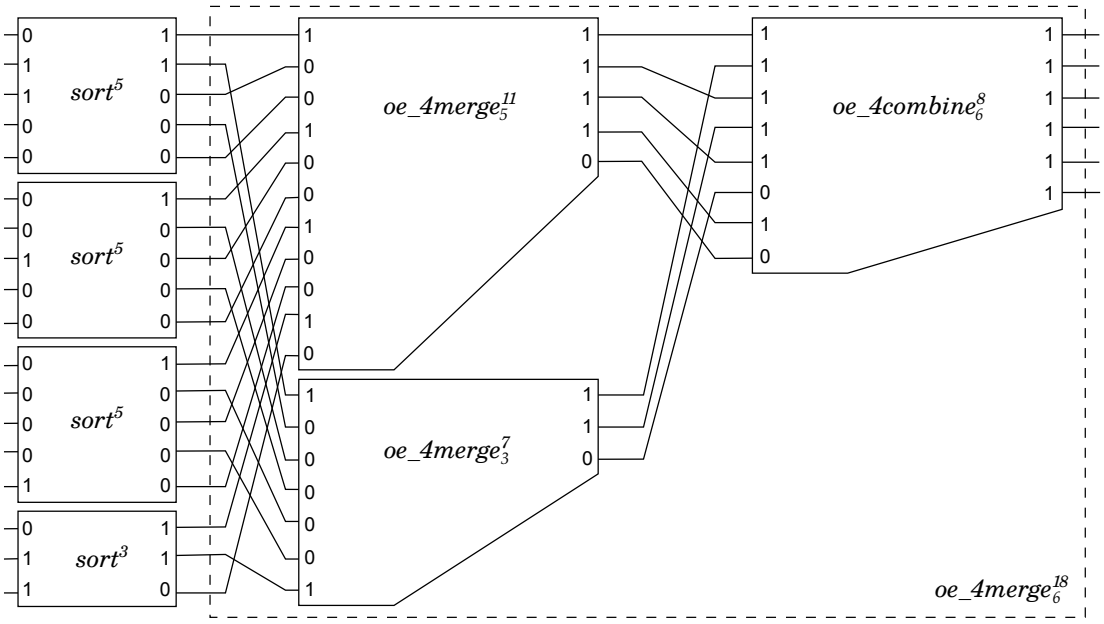


Figure 1: An example of 4-Odd-Even Selection Network, with $n = 18$ and $k = 6$.

implemented by Abío et al. [1]. Furthermore, in our previous paper we present our algorithm in a top-down, divide-and-conquer manner. Here we use a bottom-up, iterative approach. This leads to easier and cleaner implementation.

The procedure can be described as follows. Assume $k \leq n$ and that we have the sequence of Boolean literals $\bar{x}$ of length n and we want to select k largest, sorted elements, then:

- If $k = 0$, there is nothing to do.
- If $k = 1$, simply select the largest element from n inputs using a direct network.
- If $k > 1$, then split the input into subsequences of either the same literals (of length at least 2) or sorted 5 singletons (using a direct selection network of order $(5, \min(5, k))$). Next, sort subsequences by length, in a non-increasing order. In loop: merge each 4 (or less) consecutive subsequences into one (using 4-Odd-Even Merging Network as a sub-procedure) and select at most k largest items until one subsequence remains.

Example. See Figure 1 for a schema of our selection network (where $n = 18$ and $k = 6$) which selects 6 largest elements from the input 011000010000001011.

Code notation. Assume $\bar{x} = \langle x_1, \ldots, x_n \rangle$ and $\bar{y} = \langle y_1, \ldots, y_m \rangle$ are Boolean sequences. Then $\bar{x} :: \bar{y} = \langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$. We use also the following notation: $\bar{x}_{odd} = \langle x_1, x_3, \ldots \rangle$, $\bar{x}_{even} = \langle x_2, x_4, \ldots \rangle$, $\bar{x}_{a,\ldots,b} = \langle x_a, \ldots, x_b \rangle$, $1 \leq a \leq b \leq n$, and the *prefix/suffix* operators: $\mathrm{pref}(i, \bar{x}) = \bar{x}_{1,\ldots,i}$ and $\mathrm{suff}(i, \bar{x}) = \bar{x}_{i,\ldots,n}$, $1 \leq i \leq n$. The length of $\bar{x}$ is denoted by $|\bar{x}|$ and the number of occurrences of a given value b in $\bar{x}$ is denoted by $|\bar{x}|_b$. A sequence $\bar{x}$ is top k sorted, for $k \leq n$, if $\langle x_1, \ldots, x_k \rangle$ is sorted and $x_k \geq x_i$, for each $i > k$.

Algorithm 1 $oe\_4combine^s_k$

Input: A pair of sorted sequences $\langle \bar{x}, \bar{y} \rangle$, where $k \leq s = |\bar{x}| + |\bar{y}|$, $|\bar{y}| \leq \lfloor k/2 \rfloor$, $|\bar{x}| \leq \lfloor k/2 \rfloor + 2$ and $|\bar{y}|_1 \leq |\bar{x}|_1 \leq |\bar{y}|_1 + 4$.

1: Let $x(i)$ denote 0 if $i > |\bar{x}|$ or else x_i. Let $y(i)$ denote 1 if $i < 1$ or 0 if $i > |\bar{y}|$ or y_i, otherwise.
2: **for all** $j \in \{1, \ldots, |\bar{x}| + |\bar{y}|\}$ **do**
3: $\quad i = \lceil j/2 \rceil$
4: $\quad$ **if** j is even **then** $a_j \leftarrow \max(\max(x(i+2), y(i)), \min(x(i+1), y(i-1)))$
5: $\quad$ **else** $a_j \leftarrow \min(\max(x(i+1), y(i-1)), \min(x(i), y(i-2)))$
6: **return** $\bar{a}$

Ensure: The output is sorted and is a permutation of the inputs.

Algorithm 2 $oe\_4merge^s_k$

Input: A tuple of sorted sequences $\langle \bar{w}, \bar{x}, \bar{y}, \bar{z} \rangle$, where $1 \leq k \leq s = |\bar{w}| + |\bar{x}| + |\bar{y}| + |\bar{z}|$ and $k \geq |\bar{w}| \geq |\bar{x}| \geq |\bar{y}| \geq |\bar{z}|$.

1: **if** $|\bar{x}| = 0$ **then return** $\bar{w}$
2: **if** $|\bar{w}| = 1$ **then return** $select^s_k(\bar{w} :: \bar{x} :: \bar{y} :: \bar{z})$ ▷ Note that $s \leq 4$ in this case
3: $s_a = \lceil |\bar{w}|/2 \rceil + \lceil |\bar{x}|/2 \rceil + \lceil |\bar{y}|/2 \rceil + \lceil |\bar{z}|/2 \rceil$; $\quad k_a = \min(s_a, \lfloor k/2 \rfloor + 2)$;
4: $s_b = \lfloor |\bar{w}|/2 \rfloor + \lfloor |\bar{x}|/2 \rfloor + \lfloor |\bar{y}|/2 \rfloor + \lfloor |\bar{z}|/2 \rfloor$; $\quad k_b = \min(s_b, \lfloor k/2 \rfloor)$
5: $\bar{a} \leftarrow oe\_4merge^{s_a}_{k_a}(\bar{w}_{odd}, \bar{x}_{odd}, \bar{y}_{odd}, \bar{z}_{odd})$ ▷ Recursive calls.
6: $\bar{b} \leftarrow oe\_4merge^{s_b}_{k_b}(\bar{w}_{even}, \bar{x}_{even}, \bar{y}_{even}, \bar{z}_{even})$
7: **return** $oe\_4combine^{k_a+k_b}_k(\mathrm{pref}(k_a, \bar{a}), pref(k_b, \bar{b})) :: \mathrm{suff}(k_a + 1, \bar{a}) :: \mathrm{suff}(k_b + 1, \bar{b})$

Ensure: The output is top k sorted and is a permutation of the inputs.

The merging procedure uses a technique based on the odd-even merging, which Batcher and Lee called the *multiway merging* [17]. We implemented 4-way merger with 2 recursive sub-mergers. The pseudo code for this procedure is presented in Algorithm 2. The algorithm recursively merges subsequences with odd and even indices (lines 5–6) and then uses a *combine* operation to fix the order of elements (line 7). For base cases, since we assume that $|\bar{w}| \geq |\bar{x}| \geq |\bar{y}| \geq |\bar{z}|$, we need only to check – in line 1 – if $\bar{x}$ is empty (then only $\bar{w}$ is non-empty) or – in line 2 – if $\bar{w}$ contains only a single element – then the rest of the sequences contains at most one element and we can simply order them with a selector. In other cases we have $|\bar{w}| \geq 2$ and $|\bar{x}| \geq 1$, thus $|\bar{w}_{odd}| < |\bar{w}|$ and $|\bar{w}_{even}| < |\bar{w}|$, so the sizes of subproblems solved by recursive calls decrease.

Example. In Figure 1, in dashed lines, a schema of 4-Odd-Even merger is presented with $s = 18$, $k = 6$, $|\bar{w}| = |\bar{x}| = |\bar{y}| = 5$ and $|\bar{z}| = 3$. First, the inputs are split into two by odd and even indices, and the recursive calls are made. After that, a combine operation fixes the order of elements, to output the 6 largest ones.

In the case of 4-way merger, the combine operation by Batcher and Lee [17] uses two layers of comparators to fix the order of elements of two sorted sequences $\bar{x}$ and $\bar{y}$, as presented in Figure 2. The combine operation takes sequences $\langle x_0, x_1, \ldots \rangle$ and $\langle y_0, y_1, \ldots \rangle$ and performs a zip operation: $\langle x_0, y_0, x_1, y_1, x_2, y_2, \ldots \rangle$. Then, two layers of comparators are applied: $[y_i : x_{i+2}]$, for $i = 0, 1, \ldots$, resulting in $\langle x'_0, y'_0, x'_1, y'_1, \ldots \rangle$, and then $[y'_i : x'_{i+1}]$, for $i = 0, 1, \ldots$, to get $\langle x''_0, y''_0, x''_1, y''_1, \ldots \rangle$.

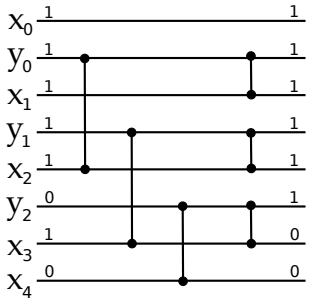


Figure 2: Comparators of $oe\_4combine_6^8$ from Figure 1.

If we were to directly encode each comparator separately in a combine operation we would need to use 3 clauses and 2 additional variables on each comparator. The novelty of our construction is that the encoding of a combine phase requires 5 clauses and 2 variables on average for each pair of comparators, using the following observations:

$$y'_i = \max(y_i, x_{i+2}) \equiv y_i \vee x_{i+2} \quad i = 0, 1, \ldots$$
$$x'_i = \min(y_{i-2}, x_i) \equiv y_{i-2} \wedge x_i \quad i = 2, 3, \ldots$$

and

$$\begin{aligned} y''_i &= \max(y'_i, x'_{i+1}) = y'_i \vee x'_{i+1} = y_i \vee x_{i+2} \vee (y_{i-1} \wedge x_{i+1}), \\ x''_i &= \min(y'_{i-1}, x'_i) = y'_{i-1} \wedge x'_i = (y_{i-1} \vee x_{i+1}) \wedge y_{i-2} \wedge x_i \\ &= (y_{i-1} \wedge y_{i-2} \wedge x_i) \vee (y_{i-2} \wedge x_i \wedge x_{i+1}) = (y_{i-1} \wedge x_i) \vee (y_{i-2} \wedge x_{i+1}) \end{aligned}$$

In the above calculations we use the fact that the input sequences are sorted, therefore $y_{i-1} \wedge y_{i-2} = y_{i-1}$ and $x_i \wedge x_{i+1} = x_{i+1}$. By the above observations, the two calculated values can be encoded using the following set of 5 clauses:

$$y_i \Rightarrow y''_i, x_{i+2} \Rightarrow y''_i, y_{i-1} \wedge x_{i+1} \Rightarrow y''_i, y_{i-1} \wedge x_i \Rightarrow x''_i, y_{i-2} \wedge x_{i+1} \Rightarrow x''_i$$

if 1's should be propagated from inputs to outputs, otherwise:

$$y''_i \Rightarrow y_{i-1} \vee x_{i+2}, y''_i \Rightarrow y_i \vee x_{i+1}, x''_i \Rightarrow x_i, x''_i \Rightarrow y_{i-2}, x''_i \Rightarrow y_{i-1} \vee x_{i+1}.$$

This saves one clause and two variables for each pair of comparators in the original combine operation, which scales to $\frac{1}{2}k$ clauses and k variables saved for each two layers of comparators associated with the use of a 4-way merger. The pseudo code for our combine procedure is presented in Algorithm 1.

2.2 Introduction to Mixed Radix Base Technique

To demonstrate how sorters can be used to translate PB-constraints, consider the following example from [11]:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 2y_1 + 3y_2 \geq 4$$

The sum of coefficients is 11. We build a sorting network of size 11, feeding y_1 into two of the inputs, y_2 into three of the inputs, and all the signals x_i into one input each. To assert the constraint, one just asserts the fourth output bit of the sorter.

The shortcoming of this approach is that the resulting size of a CNF after transformation of the sorting network can get exponential if the coefficients get bigger. Consider an example from [2]:

$$3x_1 + 2x_2 + 4x_3 \leq 5, \qquad 30001x_1 + 19999x_2 + 39998x_3 \leq 50007$$

Both constraints are equivalent. The Boolean function they represent can be expressed, for example, by the clauses $\bar{x}_1 \vee \bar{x}_3$ and $\bar{x}_2 \vee \bar{x}_3$. But clearly, a sorting network for the left constraint will be smaller.

To remedy this situation the authors of MiniSat+ propose a method to decompose the constraint into a number of interconnected sorting networks, where sorters play the role of adders on unary numbers in a *mixed radix representation*.

In the classic base r radix system, positive integers are represented as finite sequences of digits $\mathbf{d} = \langle d_0, \ldots, d_{m-1} \rangle$ where for each digit $0 \leq d_i < r$, and for the most significant digit, $d_{m-1} > 0$. The integer value associated with $\mathbf{d}$ is $v = d_0 + d_1 r + d_2 r^2 + \cdots + d_{m-1} r^{m-1}$. A mixed radix system is a generalization where a base $\mathbf{B}$ is a sequence of positive integers $\langle r_0, \ldots, r_{m-1} \rangle$. The integer value associated with $\mathbf{d}$ is $v = d_0 w_0 + d_1 w_1 + d_2 w_2 + \cdots + d_{m-1} w_{m-1}$ where $w_0 = 1$ and for $i \geq 0$, $w_{i+1} = w_i r_i$. For example, the number $\langle 2, 4, 10 \rangle_{\mathbf{B}}$ in base $\mathbf{B} = \langle 3, 5 \rangle$ is interpreted as $2 \times \mathbf{1} + 4 \times \mathbf{3} + 10 \times \mathbf{15} = 164$ (values of w_i's in boldface).

The decomposition of a PB-constraint into sorting networks is roughly as follows: first, find a "suitable" finite base $\mathbf{B}$ for the set of coefficients, for example, in MiniSat+ base is chosen so that the sum of all the digits of the coefficients written in that base, is as small as possible. Then for each element r_i of $\mathbf{B}$ construct a sorting network where the inputs to i-th sorter will be those digits $\mathbf{d}$ (from the coefficients) where d_i is non-zero, plus the potential carry bits from the $(i-1)$-th sorter. For examples of how this procedure works we refer to the papers [11] and [8].

In the following subsections we explain how we have extended the MiniSat+ solver to make the above process (and the entire PB-solving computation) more efficient.

2.3 Simplifying Inequality Assertions

We use the following optimization found in NaPS [21] for simplifying inequality assertions in a constraint. We introduce this concept with an example. In order to assert the constraint $a_1 l_1 + \cdots + a_n l_n \geq k$ the encoding compares the digits of the sum of the terms on the left side of the constraint with those from k (in some base $\mathbf{B}$) from the right side. Consider the following example:

$$5x_1 + 7x_2 \geq 9$$

Assume that the base is $\mathbf{B} = \langle 2, 2 \rangle$. Then $9 = \langle 1, 0, 2 \rangle_{\mathbf{B}}$, but if we add 7 to both sides of the inequality:

$$7 + 5x_1 + 7x_2 \geq 16$$

then those constraints are obviously equivalent and $16 = \langle 0, 0, 4 \rangle_{\mathbf{B}}$. Now in order to assert the inequality we only need to assert a single output variable of the encoding of the sum of LHS coefficients (using a singleton clause). The constant 7 on the LHS has a very small impact on

the size of LHS encoding. This simplification allows for the reduction of the number of clauses in the resulting CNF encoding, as well as allows better propagation.

2.4 Optimal Base Problem

We have mentioned that MiniSat+ searches for a mixed radix base such that the sum of all the digits of the coefficients written in that base, is as small as possible. In their paper [11] authors mention in the footnote that:

> *The best candidate is found by a brute-force search trying all prime numbers* < 20*. This is an ad-hoc solution that should be improved in the future. Finding the optimal base is a challenging optimization problem in its own right.*

Codish et al. [8] present an algorithm which scales to find an optimal base consisting of elements with values up to $1,000,000$ and they consider several measures of optimality for finding the base. They show experimentally that in many cases finding a better base leads also to better SAT solving time. We use their algorithm in our solver, but we restrict the domain of the base to prime numbers less than 50, as preliminary experiments show that numbers in the base are usually small.

2.5 Minimization Strategy

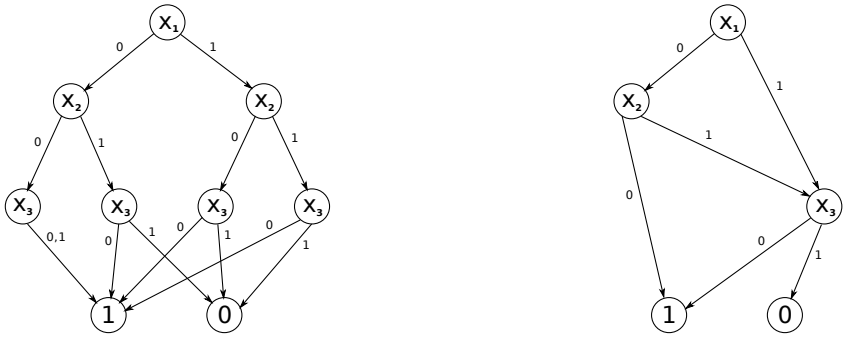
The key to efficiently solve Pseudo Boolean optimization problems is the repeated use of a SAT-solver. Assume we have a minimization problem with an objective function $f(x)$. First, without considering f, we run the solver on a set of constraints to get an initial solution $f(x_0) = k$. Then we add the constraint $f(x) < k$ and run the solver again. If the problem is unsatisfiable, k is the optimal solution. If not, the process is repeated with the new (smaller) candidate solution k'. The minimization strategy is about the choice of k'. If we choose k' as reported by the SAT-solver, then we are using the so called *sequential* strategy – this is implemented in MiniSat+.

Sakai and Nabeshima [21] propose the *binary* strategy for the choice of new k'. Let k be the best known goal value and l be the greatest known lower bound, which is initially the sum of negative coefficients of f. After each iteration, new constraint $p \Rightarrow f(x) < \lfloor (k(q-1)+l)/q \rfloor$ is added, where p is a fresh variable (assumption) and q is a constant (we set $q = 3$ as default value). Depending on the new SAT-solver answer, $\lfloor (k(q-1)+l)/q \rfloor$ becomes the new upper or lower bound (in this case p is set to 0), and the process begin anew.

In our implementation we use binary strategy until the difference between the upper and lower bounds of the goal value is less than 96, then we switch to the sequential strategy. We do this in order to avoid a situation when a lot of computation is needed when searching for UNSAT answers, which could arise if only binary strategy was used. This was also observed in [21] and the authors have used it as a default strategy. Moreover, they propose to alternate between binary and sequential strategy depending on the SAT-solver answer in a given iteration.

2.6 ROBDDs Instead of BDDs

One of the encodings of MiniSat+ is based on *Binary Decision Diagrams* (BDDs). We have improved the implementation of this encoding by using the more recent *Reduced Ordered BDD* (ROBDD) construction [2]. Now ROBDD is used to create a DAG representation of a constraint. One of the advantages of ROBBDs is that we can reuse nodes in the ROBDD structure, which



(a) BDD (b) ROBDD

Figure 3: Construction of a BDD (left) and a ROBDD (right) for $2x_1 + 3x_2 + 5x_3 \leq 6$.

results in a smaller encoding. This concept is illustrated in Figure 3, which shows an example from [2] of BDD and ROBDD for the PB-constraint $2x_1 + 3x_2 + 5x_3 \leq 6$. Two reductions are applied (until fix-point) for obtaining ROBDD: removing nodes with identical children and merging isomorphic subtrees. See [2] for a more detailed example.

2.7 Merging Carry Bits

In the construction of interconnected sorters in MiniSat+ carry bits from one sorter are being fed to the next sorter. In the footnote in [11] it is mentioned that:

> *Implementation note: The sorter can be improved by using the fact that the carries are already sorted.*

We go with this suggestion and use our merging network to merge the carry bits with a sorted digits representation instead of simply forwarding the carry bits to the inputs of the next selection network.

2.8 SAT-solver

The underlying SAT-solver of MiniSat+ is MiniSat [10] created by Niklas Eén and Niklas Sörensson. It has served as an extension to many new solvers, but it is now quite outdated. We have integrated a solver called COMiniSatPS by Chanseok Oh [19], which have collectively won six medals in SAT Competition 2014 and Configurable SAT Solver Challenge 2014. Moreover, the modification of this solver called MapleCOMSPS won the Main Track category of SAT Competition 2016[1].

3 Experimental Evaluation

Our extension of MiniSat+ based on the features explained in this paper, which we call KP-MiniSat+, is available online[2]. It should be linked with a slightly modified COMiniSatPS,

[1] See http://baldur.iti.kit.edu/sat-competition-2016/
[2] See https://github.com/karpiu/kp-minisatp

solver	DEC-SMALLINT-LIN			OPT-BIGINT-LIN			OPT-SMALLINT-LIN		
	Sat	UnSat	cpu	Opt	UnSat	cpu	Opt	UnSat	cpu
KP-MS+	432	1049	647041	359	72	1135925	808	86	1289042
NaPS	348	1035	816725	314	69	1314536	799	81	1330536
PBLib	349	922	1104508	–	–	–	691	56	1611247
MS+	395	951	895774	149	71	1647958	715	73	1515166
MS+COM	428	1027	703269	174	71	1609433	734	71	1491269

Table 1: Number of solved instances of PB-competition benchmarks.

also available online[3]. Detailed results of the experimental evaluation are also available online[4].

The set of instances we use is from the Pseudo-Boolean Competition 2016[5]. We use instances with linear, Pseudo-Boolean constraints that encode either decision or optimization problems. To this end, three categories from the competition have been selected:

- **DEC-SMALLINT-LIN** - 1783 instances of decision problems with small coefficients in the constraints (no constraint with sum of coefficients greater than 2^{20}). No objective function to optimize. The solver must simply find a solution.
- **OPT-BIGINT-LIN** - 1109 instances of optimization problems with big coefficients in the constraints (at least one constraint with a sum of coefficients greater than 2^{20}). An objective function is present. The solver must find a solution with the best possible value of the objective function.
- **OPT-SMALLINT-LIN** - 1600 instances of optimization problems. Similar to OPT-BIGINT-LIN but with small coefficients (as in DEC-SMALLINT-LIN) in the constraints.

We compare our solver (abbreviated to **KP-MS+**) with two state-of-the-art general purpose constraint solvers. First is the pbSolver from PBLib ver. 1.2.1, by Tobias Philipp and Peter Steinke [20] (abbreviated to **PBLib** in the results). This solver implements a plethora of encodings for three types of constraints: at-most-one, at-most-k (cardinality constraints) and Pseudo-Boolean constraints. The PBLib automatically normalizes the input constraints and decides which encoder provides the most effective translation. We have launched the program `./BasicPBSolver/pbsolver` of PBLib on each instance with the default parameters.

The second solver is NaPS ver. 1.02b by Masahiko Sakai and Hidetomo Nabeshima [21] which implements improved ROBDD structure for encoding constraints in band form, as well as other optimizations. This solver is also built on the top of MiniSat+. NaPS won two of the optimization categories in the Pseudo-Boolean Competition 2016: OPT-BIGINT-LIN and OPT-SMALLINT-LIN. We have launched the main program of NaPS on each instance, with parameters `-a -s -nm`.

We also compare our solver with the original MiniSat+ in two different versions, one using the original MiniSat SAT-solver and the other using the COMiniSatPS (the same as used by us). We label these **MS+** and **MS+COM** in the results. We prepared results for **MS+COM** in order to show that the advantage of using our solver does not come simply from changing the underlying SAT-solver.

[3]See https://github.com/marekpiotrow/cominisatps
[4]See http://www.ii.uni.wroc.pl/%7ekarp/pos/2018.html
[5]See http://www.cril.univ-artois.fr/PB16/

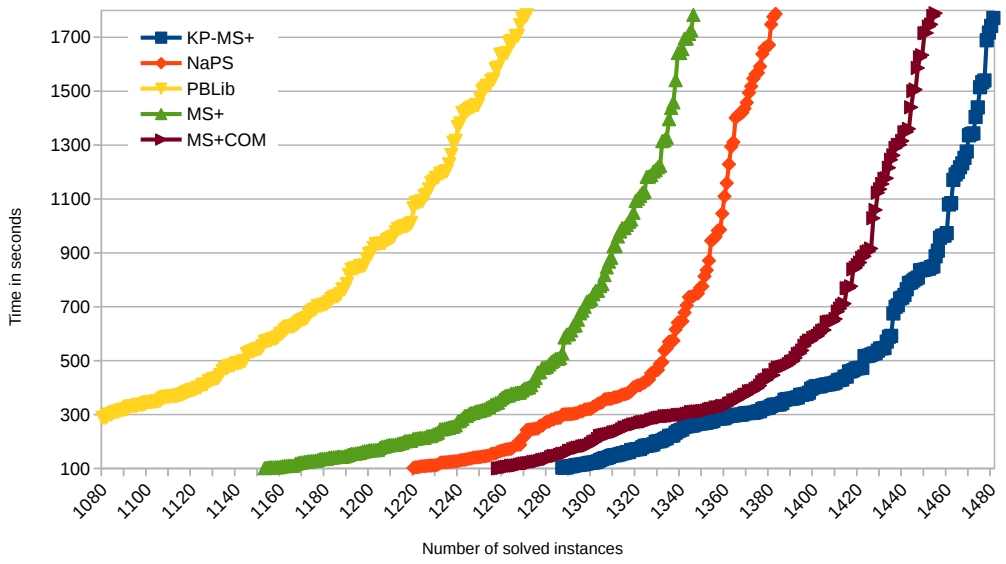


Figure 4: Cactus plot for DEC-SMALLINT-LIN division.

We have launched our solver on each instance, with parameters `-a -s -cs -nm`, where `-cs` means that in experiments the solver used just one encoding technique, the 4-Odd-Even Selection Networks combined with a direct encoding of small subnetworks.

All experiments were carried out on the machines with Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz. Timeout limit is set to 1800 seconds and memory limit is 15 GB, which are enforced with the following commands: `ulimit -Sv 15000000` and `timeout -k 20 1809 <solver> <parameters> <instance>`.

We would like to note that we also wanted to include in this evaluation the winner of DEC-SMALLINT-LIN category, which is the solver based on the *cutting planes* technique, but we refrained from that for the following reason. We have not found the source code of this solver and the only working version found in the author's website[6] is a binary file without any documentation. Because of this, we were unable to get any meaningful results of running forementioned program on optimization instances.

In Table 1 we present the number of solved instanced for each competition category. **Sat**, **UnSat** and **Opt** have the usual meaning, while **cpu** is the total solving time of the solver over all instances of a given category. Results clearly favor our solver. We observe significant improvement in the number of solved instances in comparison to NaPS in categories DEC-SMALLINT-LIN and OPT-BIGINT-LIN. The difference in the number of solved instances in the OPT-SMALLINT-LIN category is not so significant. Solver PBLib had the worst performance in this evaluation. Notice that the results of PBLib for OPT-BIGINT-LIN division is not available. This is because PBLib is using 64-bit integers in calculations, thus could not be launched with all OPT-BIGINT-LIN instances.

Figures 4, 5 and 6 show cactus plots of the results, which indicate the number of solved instances within the time. We see clear advantage of our solver over the competition in the DEC-SMALLINT-LIN and OPT-BIGINT-LIN categories.

[6] See http://www.csc.kth.se/%7eelffers/

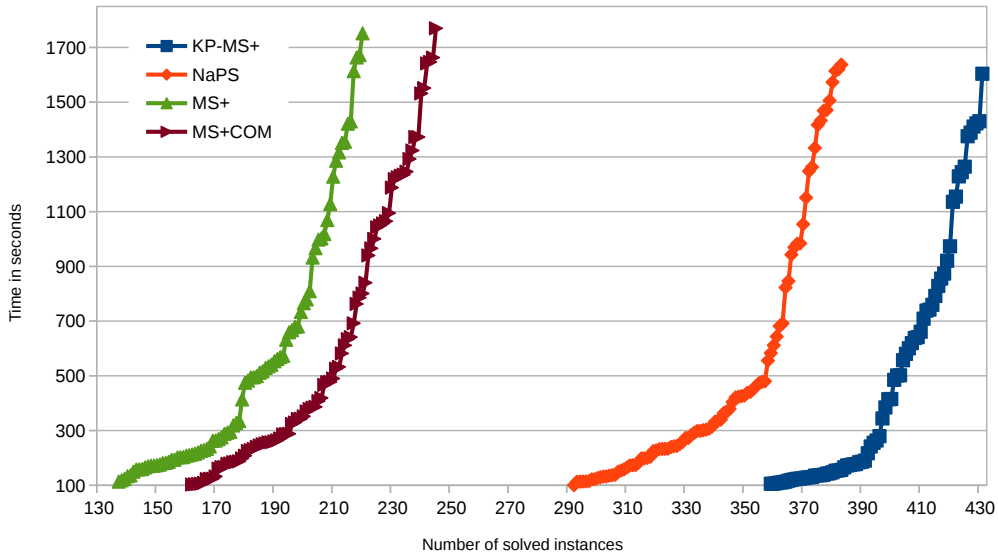


Figure 5: Cactus plot for OPT-BIGINT-LIN division.

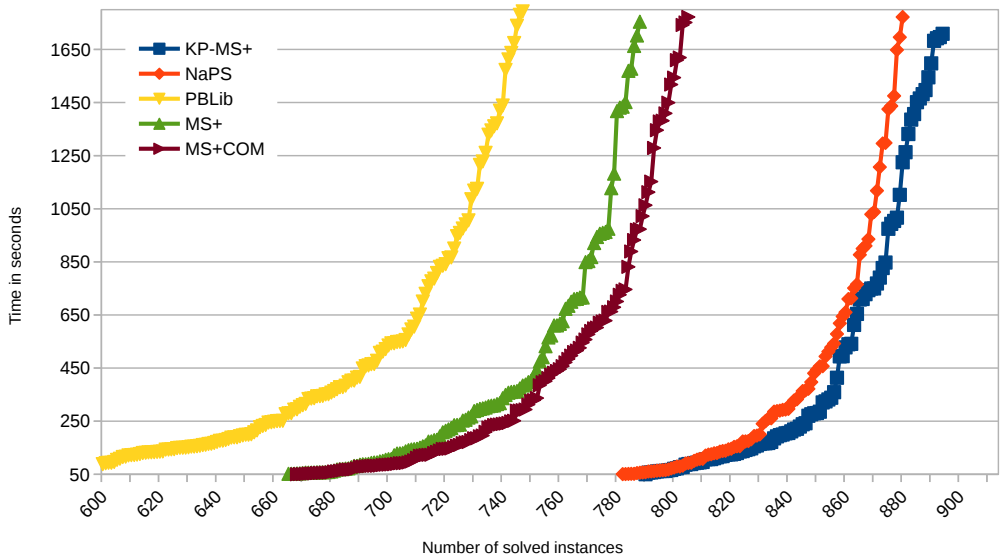


Figure 6: Cactus plot for OPT-SMALLINT-LIN division.

4 Conclusions

In this paper we proposed a new method of encoding PB-constraints into SAT based on sorters. We have extended the MiniSat+ with the 4-way merge selection algorithm and showed that this method is competitive to other state-of-the-art solutions. Our algorithm is short and easy

to implement. Our implementation is modular, therefore it can be easily extracted and applied in other solvers.

References

[1] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 80–96. Springer, 2013.

[2] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, 45:443–480, 2012.

[3] Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Generic ilp versus specialized 0-1 ilp: An update. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 450–457. ACM, 2002.

[4] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[5] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo-boolean constraints to sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.

[6] Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314. ACM, 1968.

[7] Randal E Bryant, Shuvendu K Lahiri, and Sanjit A Seshia. Deciding clu logic formulas via boolean and pseudo-boolean encodings. In *In Proc. Intl. Workshop on Constraints in Formal Verification (CFV'02*. Citeseer, 2002.

[8] Michael Codish, Yoav Fekete, Carsten Fuhs, and Peter Schneider-Kamp. Optimal base encodings for pseudo-boolean constraints. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 189–204. Springer, 2011.

[9] Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 154–172. Springer, 2010.

[10] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[11] Niklas Eén and Niklas Sorensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[12] Yoav Fekete and Michael Codish. Simplifying pseudo-boolean constraints in residual number systems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 351–366. Springer, 2014.

[13] Michał Karpiński. Encoding cardinality constraints using standard encoding of generalized selection networks preserves arc-consistency. *Theoretical Computer Science*, 707:77–81, 2018.

[14] Michał Karpiński and Marek Piotrów. Smaller selection networks for cardinality constraints encoding. In *International Conference on Principles and Practice of Constraint Programming*, pages 210–225. Springer, 2015.

[15] Michał Karpiński and Marek Piotrów. Encoding cardinality constraints using generalized selection networks. *arXiv preprint arXiv:1704.04389*, 2017.

[16] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[17] De-Lei Lee and Kenneth E. Batcher. A multiway merge sorting network. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):211–215, 1995.

[18] Vasco M Manquinho and Olivier Roussel. The first evaluation of pseudo-boolean solvers (pb'05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2:103–143, 2006.

[19] Chanseok Oh. *Improving sat solvers by exploiting empirical characteristics of cdcl*. PhD thesis, New York University, 2016.

[20] Tobias Philipp and Peter Steinke. Pblib–a library for encoding pseudo-boolean constraints into cnf. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 9–16. Springer, 2015.

[21] Masahiko Sakai and Hidetomo Nabeshima. Construction of an robdd for a pb-constraint in band form and related techniques for pb-solvers. *IEICE TRANSACTIONS on Information and Systems*, 98(6):1121–1127, 2015.

[22] Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Why cumulative decomposition is not as bad as it sounds. In *International Conference on Principles and Practice of Constraint Programming*, pages 746–761. Springer, 2009.