Cardinality Networks and their Applications

Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell*

Abstract. We introduce *Cardinality Networks*, a new CNF encoding of cardinality constraints. It improves upon the previously existing encodings such as the sorting networks of [ES06] in that it requires much less clauses and auxiliary variables, while arc consistency is still preserved: e.g., for a constraint $x_1 + \ldots + x_n \leq k$, as soon as k variables among the x_i's become true, unit propagation sets all other x_i's to false. Our encoding also still admits *incremental strengthening:* this constraint for any smaller k is obtained without adding any new clauses, by setting a single variable to false.
Here we give precise recursive definitions of the clause sets that are needed and give detailed proofs of the required properties. We demonstrate the practical impact of this new encoding by careful experiments comparing it with previous encodings on real-world instances.

1 Introduction

Compared with other systematic constraint solving techniques, SAT solvers have many advantages for non-expert users as extremely efficient off-the-shelf black boxes that moreover require no tuning regarding variable (or value) selection heuristics. Therefore quite some work has been devoted to finding good propositional encodings for many kinds of constraints.

A particularly important class of constraints are the *cardinality constraints*, i.e., constraints of the form $x_1 + \ldots + x_n \ \# \ k$ where k is a natural number and $\# \ \in \{<, \leq, =, \geq, >\}$.

Cardinality constraints appear in many practical problem contexts, such as timetabling, scheduling, or pseudo-boolean constraint solving. For instance, given an input formula F over n variables $x_1, \ldots, x_n$, one may be interested in finding a model of F in which at most k variables are set to true. For this, one can add the clauses encoding the constraint $x_1 + \ldots + x_n \leq k$. Going beyond, for instance for the *min-ones* problem for F, that is, finding a model with the minimal number of true variables, one can *incrementally strengthen* the constraint for successively lower k until it becomes unsatisfiable. In fact, cardinality constraints frequently occur in other optimization problems too. For example, the Max-SAT problem consists of, given a set of clauses $S = \{C_1, \ldots, C_n\}$, finding an assignment A that satisfies the maximal number of clauses in S. One way of doing this is to add a fresh indicator variable x_i to each clause, getting $\{C_1 \vee x_1, \ldots, C_n \vee x_n\}$

* Technical Univ. of Catalonia, Barcelona. All authors partially supported by Spanish Min. of Educ. and Science through the LogicTools-2 project (TIN2007-68093-C02-01). The first author is also partially supported by FPI grant TIN2004-03382.

and incrementally strengthening the constraint $x_1 + \ldots + x_n \leq k$. In general, it is typical to see situations where n is much larger than k.

This kind of applications of cardinality constraints has been very elegantly handled in MiniSAT and its extension to pseudo-boolean constraints [ES06]. There, one encoding for cardinality constraints is based on *sorting networks* with inputs $x_1, \ldots, x_n$ and output $y_1, \ldots, y_n$, such that if exactly k input variables are true, then $y_1, \ldots, y_k$ will become true and $y_{k+1}, \ldots, y_n$ will be false. For enforcing the constraint $x_1 + \ldots + x_n \leq k$, it then suffices to set y_{k+1} to false, and incrementally strengthening the constraint can be done by setting to false y_p's with successively smaller p.

In [ES06] it is also proved that for the CNF encoding of sorting networks unit propagation preserves arc consistency. For instance, for a constraint of the form $x_1 + \ldots + x_n \leq k$, as soon as k variables among the x_i's become true, unit propagation sets all other x_i's to false. The proof of arc consistency given in [ES06] relies on general properties of sorting networks.

Here we give recursive definitions for this kind of networks that, given sequences of input variables, return a sequence of output variables and a set of clauses. The required arc-consistency properties under unit propagation can be directly proved by induction from these definitions. Our starting point will be a deconstruction of the odd-even merge sorting networks of [Bat68], focussing on their specific use for encoding cardinality constraints in SAT.

For this purpose, and for allowing the reader to become familiar with the notations and methodology of this paper, in Section 3 we first define *Half Merging Networks* and *Half Sorting Networks*, which require only half as many clauses as their standard versions while preserving all desired properties.

As said, in many applications, it is typical to find cardinality constraints $x_1 + \ldots + x_n \ \# \ k$ where n is much larger than k. This motivated us to look for encodings that exploit this fact. In Section 4 we introduce *Cardinality Networks* which require $O(n \log^2 k)$ clauses instead of $O(n \log^2 n)$ as in previous approaches. In addition, Cardinality Networks also leverage the advantages from the use of Half Merging and Half Sorting Networks. All definitions, properties and proofs in this section and in Section 3 are for cardinality constraints of the form $x_1 + \ldots + x_n \leq k$. Therefore, in Section 5 we extend them to the other cases such as $\geq$ and $=$, and to *range constraints* of the form $k \leq x_1 + \ldots + x_n \leq k'$.

In Section 6 we demonstrate the practical impact of this new encoding by careful experiments comparing it with previous encodings on real-world instances and we conclude in Section 7. Because of space limitation, not all results are proved in the paper[1].

2 Preliminaries

Let P be a fixed finite set of propositional variables. If $p \in P$, then p and $\overline{p}$ are *literals* of P. The *negation* of a literal l, written $\overline{l}$, denotes $\overline{p}$ if l is p, and p if l is

[1] An extended version of the paper can be found at `www.lsi.upc.edu/~rasin/cardinality-extended.ps`.

$\overline{p}$. A *clause* is a disjunction of literals $l_1 \vee \ldots \vee l_n$. A CNF *formula* is a conjunction of one or more clauses $C_1 \wedge \ldots \wedge C_n$. When it leads to no ambiguities, we will sometimes consider such a formula as the set of its clauses.

A (partial truth) *assignment* M is a set of literals such that $\{p, \overline{p}\} \subseteq M$ for no p. A literal l is *true* in M if $l \in M$, is *false* in M if $\overline{l} \in M$, and is *undefined* in M otherwise. A clause C is true in M if at least one of its literals is true in M. A formula F is true in M if all its clauses are true in M. In that case, M is a *model* of F. The systems that decide whether a formula has a model or not are called *SAT solvers*.

Most state-of-the-art SAT solvers are based on extensions of the DPLL algorithm [DP60,DLL62]. The main inference rule in DPLL is known as *unit propagation*. Given a set of clauses S and an empty assignment M, clauses are sought in which all literals are false but one, say l, which is undefined (initially only clauses of size one satisfy this condition). This literal l is then added to M and the process is iterated until reaching a fix point. If U is the set of all literals that have been added to the assignment in this process, we will denote this fact by $S \models_{up} U$.

In this paper we will work with *cardinality constraints* $a_1 + \ldots + a_n \ \# \ k$, where $\# \ \in \{\leq, \geq, =\}$, the a_i's are propositional variables and k is a natural number. An assignment M *satisfies* such a constraint if at most ($\leq$), at least($\geq$) or exactly ($=$) k literals in $\{a_1, \ldots, a_n\}$ are true in M. The aim of this paper is, given a set of cardinality constraints C, to obtain a CNF formula S such that looking for assignments satisfying C is equivalent to looking for models of S. Moreover this S should be as small as possible and, whenever a concrete value for a variable in a constraint can be inferred, this should be detected by unit propagation on S.

In what follows, we consider *variable sequences*, or simply *sequences*, which are ordered lists of distinct propositional variables, written $\langle x_1 \ldots x_n \rangle$, and denoted by capital letters $A, B, C, \ldots$ Unless stated otherwise, these lists always have length $n = 2^m$, for some $m \geq 0$. When necessary these lists will be seen as sets, so that we can consider subsets of their variables.

Sometimes new *fresh* variables, that is, distinct new variables, will be introduced, These will always be denoted by the (possibly subscripted or primed) letters c, d, e.

3 Half Merging and Half Sorting Networks

In this section we introduce *Half Merging Networks* and *Half Sorting Networks*, which are like the *Sorting Networks* based on odd-even merges of [Bat68,ES06], but only need half of the clauses. The definitions and properties that are given will be used later on and allow the reader to become familiar with our notations and methodology. We remind that all the definitions in this section and in Section 4 are designed to be used in constraints of the form $x_1 + \ldots + x_n \leq k$, and that we implicitly assume that all sequences have size 2^m for some $m \geq 0$.

3.1 Half Merging Networks

Given two sequences A and B of length n, the *Half Merging Network of A and B*, denoted $HMerge(A, B)$, is a pair (C, S), where C is a sequence of length $2n$ and S is a set of clauses, defined as follows.

For sequences of length 1 we define:

$$HMerge(\ \langle a\rangle,\ \langle b\rangle\ )\ =\ (\ \langle\ c_1\ c_2\ \rangle,\ \{\ \overline{a}\vee\overline{b}\vee c_2,\ \ \overline{a}\vee c_1,\ \ \overline{b}\vee c_1\ \ \}\ )$$

For sequences of length $n > 1$ we define:

$$HMerge(\ \langle a_1 \dots a_n\rangle,\ \langle b_1 \dots b_n\rangle\ )\ =\ (\ \langle\ d_1\, c_2\ \dots\ c_{2n-1}\, e_n\ \rangle,\ \ S_{odd} \cup S_{even} \cup S'\ )$$

recursively in terms of the odd and the even subsequences:

$$HMerge(\ \langle a_1\, a_3 \dots a_{n-1}\rangle,\ \langle b_1\, b_3 \dots b_{n-1}\rangle\ )\ =\ (\ \langle d_1\ \dots\ d_n\rangle,\ S_{odd}\ \ \ ),$$
$$HMerge(\ \langle a_2\, a_4 \dots a_n\ \ \ \rangle,\ \langle b_2\, b_4 \dots b_n\ \ \ \rangle\ )\ =\ (\ \langle e_1\ \dots\ e_n\rangle,\ S_{even}\ ),$$

where the clause set S' is: $\bigcup_{i=1}^{n-1}\{\ \overline{d}_{i+1}\vee\overline{e}_i\vee c_{2i+1},\ \ \overline{d}_{i+1}\vee c_{2i},\ \ \overline{e}_i\vee c_{2i}\ \}$.

Example 1. Intuitively, a (Half) Merging Network merges two sequences of input variables $\langle a_1 \dots a_n\rangle$ and $\langle b_1 \dots b_n\rangle$ that are already sorted into a single sorted output sequence $\langle c_1 \dots c_{2n}\rangle$, and the required unit propagation is that if $a_1 \dots a_p$ and $b_1 \dots b_q$ are true, then the first $p + q$ output variables will become true (Lemma 1 below), and (roughly speaking) if in addition c_{p+q+1} is set to false, then also a_{p+1} and b_{q+1} will become false (Lemma 2).

Let us take $HMerge(\ \langle a_1\ a_2\rangle,\ \langle b_1\ b_2\rangle\ )$, which is $(\ \langle d_1\ c_2\ c_3\ e_2\rangle, S)$ with S being the set of clauses:

$$\begin{array}{lll} \overline{a}_1\vee\overline{b}_1\vee d_2 & \overline{a}_2\vee\overline{b}_2\vee e_2 & \overline{d}_2\vee\overline{e}_1\vee c_3 \\ \overline{a}_1\vee d_1 & \overline{a}_2\vee e_1 & \overline{d}_2\vee c_1 \\ \overline{b}_1\vee d_1 & \overline{b}_2\vee e_1 & \overline{e}_1\vee c_2 \end{array}$$

The partial assignments $(a_1, a_2) = (1, 0)$ and $(b_1, b_2) = (0, 0)$ cause S to unit propagate the first output (d_1), but not the second one (c_2). If we add another 1 to the input, for example $(a_1, a_2) = (1, 1)$, then both d_1 and c_2 get propagated, but not c_3. For propagating c_3 we need to add another input 1, e.g, setting $(b_1, b_2) = (1, 0)$, but $(b_1, b_2) = (0, 1)$ would not do it, since this propagation only works if all ones appear as a prefix in the input sequences, which will always be the case in our uses of *HMerge*. With inputs $(a_1, a_2) = (1, 0)$ and $(b_1, b_2) = (1, 0)$, and setting c_3 to false, unit propagation will set a_2 and b_2 to false. Similar properties about propagation of ones and zeros will hold in all the constructions in this paper and will be precisely stated in each case. □

Lemma 1. *If* $HMerge(\ \langle a_1 \dots a_n\rangle,\ \langle b_1 \dots b_n\rangle\ )\ =\ (\ \langle c_1\ \dots\ c_{2n}\rangle,\ \ S\ )$ *and* $p, q \in \mathbb{N}$ *with* $1 \le p, q \le n$, *then* $S \cup \{a_1\ \dots\ a_p\ b_1 \dots b_q\} \models_{up}\ c_1, \dots, c_{p+q}$.

Lemma 2. *Let* $HMerge(\ \langle a_1 \dots a_n\rangle,\ \langle b_1 \dots b_n\rangle\ )$ *be* $(\ \langle c_1\ \dots\ c_{2n}\rangle,\ \ S\ )$, *and* $p, q \in \mathbb{N}$ *with* $p, q \le n$.

If $p < n$ *and* $q < n$ *then* $S \cup \{a_1, \dots, a_p, b_1, \dots, b_q, \overline{c}_{p+q+1}\} \models_{up}\ \overline{a}_{p+1},\ \overline{b}_{q+1}$.
If $p = n$ *and* $q < n$ *then* $S \cup \{a_1, \dots, a_p, b_1, \dots, b_q, \overline{c}_{p+q+1}\} \models_{up}\ \overline{b}_{q+1}$.
If $p < n$ *and* $q = n$ *then* $S \cup \{a_1, \dots, a_p, b_1, \dots, b_q, \overline{c}_{p+q+1}\} \models_{up}\ \overline{a}_{p+1}$.

Lemma 3. *Given A and B sequences of length n, the Half Merging Network $HMerge(A, B)$ contains $O(n \log n)$ clauses with $O(n \log n)$ auxiliary variables.*

3.2 Half Sorting Networks

Given a sequence A of length $2n$, the *Half Sorting Network of A*, denoted $HSort(A)$, is a pair (C, S), where C is a sequence of length $2n$ and S is a set of clauses, defined as follows.

For sequences of length 2 we define:

$$HSort(\ \langle a\ b \rangle\ )\ =\ HMerge(\ \langle a \rangle,\ \langle b \rangle\ )$$

For sequences of length $2n > 2$ we define:

$$HSort(\ \langle a_1\ \ldots\ a_{2n} \rangle\ )\ =\ (\ \langle\ c_1\ \ldots\ c_{2n}\ \rangle,\ \ S_D \cup S_{D'} \cup S_M\ )$$

recursively in terms of two subsequences of size n:

$HSort(\ \langle\ a_1\ \ldots\ a_n\ \rangle\ )\ =\ (\ \langle d_1 \ldots d_n \rangle,\ S_D\ ),$
$HSort(\ \langle\ a_{n+1} \ldots a_{2n} \rangle\ )\ =\ (\ \langle d'_1 \ldots d'_n \rangle,\ S_{D'}\ ),$

and the merge of them

$HMerge(\ \langle d_1 \ldots d_n\ \rangle,\ \langle d'_1 \ldots d'_n\ \rangle\ )\ =\ (\ \langle c_1\ \ldots\ c_{2n} \rangle,\ S_M\ ),$

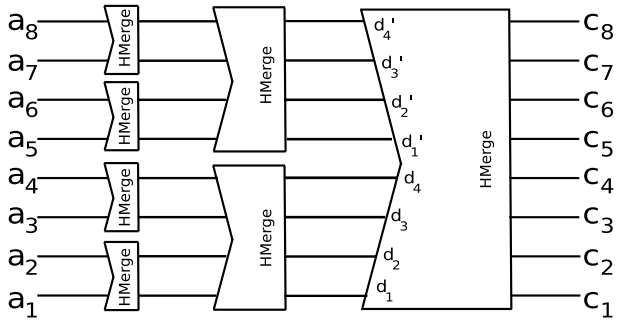

Fig. 1. *HSort* with input $\langle a_1 \ldots a_8 \rangle$ and output $\langle c_1 \ldots c_8 \rangle$

Lemma 4. *Given a sequence A of length n, the Half Sorting Network $HSort(A)$ contains $O(n \log^2 n)$ clauses with $O(n \log^2 n)$ auxiliary variables.*

Similar properties to the ones of Half Merging Networks also hold here, but without the requirement that the input ones are at prefixes: (i) if *any* p input variables are set to true, the first p output variables are unit propagated (Lemma 5), and (ii) if in addition the $p+1$-th output is set to false, the remaining input variables are set to false (Lemma 6), hence not allowing more than p input variables to be true.

Lemma 5. *Let HSort*(A) *be* $(\langle c_1 \dots c_{2n} \rangle, S)$ *and let* $A' \subseteq A$ *with* $|A'| = p$. *Then,*

$$S \cup A' \models_{up} c_1, \dots, c_p$$

Lemma 6. *Let HSort*(A) *be* $(\langle c_1 \dots c_{2n} \rangle, S)$ *and let* $A' \subsetneq A$ *with* $|A'| = p < 2n$. *Then,*

$$S \cup A' \cup \overline{c}_{p+1} \models_{up} \overline{a}_j \text{ for all } a_j \in (A - A')$$

4 Cardinality Networks

Here we exploit the fact that in cardinality constraints $x_1 + \ldots + x_n \leq k$ it is frequently the case that n is much larger than k. We introduce *Cardinality Networks* which require $O(n \log^2 k)$ clauses instead of $O(n \log^2 n)$ as in previous approaches. A main ingredient for Cardinality Networks are the *Simplified Merging Networks*, which we introduce first.

4.1 Simplified Merging Networks

If we are only interested in the (maximal) $n+1$ bits of the output (instead of the $2n$ original ones), Half Merging Networks can be further simplified. Given two sequences A and B of length n, the *Simplified Merging Network of A and B*, denoted *SMerge*(A, B), is a pair (C, S), where C is a sequence of length $n+1$ and S is a set of clauses, defined as follows. For $n = 1$, we have

$$\mathit{SMerge}(\langle a \rangle, \langle b \rangle) = (\langle c_1, c_2 \rangle, \{\overline{a} \vee \overline{b} \vee c_2, \overline{a} \vee c_1, \overline{b} \vee c_1\})$$

The case $n > 1$ is defined

$$\mathit{SMerge}(\langle a_1 \dots a_n \rangle, \langle b_1 \dots b_n \rangle) = (\langle d_1\, c_2 \ \dots\ c_{n+1} \rangle,\ S_{odd} \cup S_{even} \cup S')$$

recursively in terms of the odd and the even subsequences,

$$\mathit{SMerge}(\langle a_1\, a_3 \dots a_{n-1} \rangle, \langle b_1\, b_3 \dots b_{n-1} \rangle) = (\langle d_1 \ \dots\ d_{\frac{n}{2}+1} \rangle, S_{odd})$$
$$\mathit{SMerge}(\langle a_2\, a_4 \dots a_n \ \rangle, \langle b_2\, b_4 \dots b_n \ \rangle) = (\langle e_1 \ \dots\ e_{\frac{n}{2}+1} \rangle, S_{even})$$

where the clause set S' is:

$\bigcup_{i=1}^{\frac{n}{2}} \{\overline{d}_{i+1} \vee \overline{e}_i \vee c_{2i+1},\ \overline{d}_{i+1} \vee c_{2i},\ \overline{e}_i \vee c_{2i}\}$.

Remark: We have defined Simplified Merging Networks with $n+1$ outputs because this $n+1$-th bit is needed for the odd recursive case: $d_{\frac{n}{2}+1}$ is used in the clause set S'. But output $e_{\frac{n}{2}+1}$ from the even subcase is not used, and the $n+1$-th bit is not used either in the Cardinality Networks defined below. This fact can be exploited for a slightly further optimization in our encodings by using Simplified Merging Networks with n outputs for these subcases, but for clarity of explanation we have chosen not to do so here.

We now precisely state the propagation properties of Simplified Merging Networks. Lemma 7 is the equivalent of Lemma 1, proving that $p+q$ inputs ones properly placed (e.g. as prefixes in the input sequences), unit propagate the first $p+q$ outputs. After that, Lemma 8, the equivalent of Lemma 2, proves how zeros can be propagated from outputs to inputs.

Lemma 7. *If* $SMerge(\ \langle a_1 \dots a_n \rangle,\ \langle b_1 \dots b_n \rangle\ )\ =\ (\ \langle c_1\ \dots\ c_{n+1} \rangle,\ S\ )$ *and* $p, q \in \mathbb{N}$ *with* $1 \leq p+q \leq n+1$*, then* $S \cup \{a_1, \dots, a_p, b_1, \dots, b_q\} \models_{up} c_{p+q}$.

Proof. (By induction on n). If $n = 1$, we have

$$SMerge(\ \langle a \rangle, \langle b \rangle)\ =\ (\ \langle c_1, c_2 \rangle,\ \{\ \overline{a} \vee c_1, \overline{b} \vee c_1, \overline{a} \vee \overline{b} \vee c_2\ \}\ ).$$

If $p = 0$, $q = 1$ then setting b clearly propagates c_1. Similarly, if $p = 1$, $q = 0$, setting a propagates c_1. Otherwise, $p = 1$, $q = 1$, and a and b propagate c_2.

For the induction step ($n > 1$) we consider four different cases, depending on whether p and q are odd or even:

<u>*CASE 1*</u>: p is odd and q even. (Let $p = 2p' + 1$ and $q = 2q'$).

Let us focus on the odd part of *SMerge*:

$$SMerge(\ \langle a_1\, a_3 \dots a_{n-1} \rangle,\ \langle b_1\, b_3 \dots b_{n-1} \rangle\ )\ =\ (\ \langle d_1\ \dots\ d_{\frac{n}{2}+1} \rangle,\ S_{odd}\ ).$$

In $\langle a_1\, a_2 \dots a_p \rangle$ there are $p'+1$ odd indices, namely $\{1, 3, \dots, 2p'+1\}$. Similarly, in $\langle b_1\, b_2 \dots b_q \rangle$ there are q' odd indices, namely $\{1, 3, \dots, 2q'-1\}$. Hence, by IH we have $S_{odd} \cup \{a_1, \dots, a_{2p'+1}, b_1, \dots, b_{2q'-1}, \} \models_{up} d_{p'+q'+1}$ (note that $1 \leq (p'+1) + q' \leq \frac{n}{2} + 1$).

Now, let us take the even part of *SMerge*:

$$SMerge(\ \langle a_2\, a_4 \dots a_n \rangle,\ \langle b_2\, b_4 \dots b_n \rangle\ )\ =\ (\ \langle e_1\ \dots\ e_{\frac{n}{2}+1} \rangle,\ S_{even}\ )$$

In $\langle a_2\, a_4 \dots a_p \rangle$ there are p' even indices, namely $\{2, 4, \dots, 2p'\}$. Similarly, in $\langle b_2\, b_4 \dots b_q \rangle$ there are q' even indices, namely $\{2, 4, \dots, 2q'\}$. Hence, by IH we have $S_{even} \cup \{a_2, \dots, a_{2p'}, b_2, \dots, b_{2q'}, \} \models_{up} e_{p'+q'}$ (note that $1 \leq p' + q' \leq \frac{n}{2} + 1$).

Finally, since $1 \leq p' + q' \leq \frac{n}{2}$ the clause $\overline{d}_{p'+q'+1} \vee \overline{e}_{p'+q'} \vee c_{2p'+2q'+1}$ belongs to S, and hence literal $c_{2p'+2q'+1}$ can be unit propagated, as we wanted to prove.

<u>*CASE 2*</u>: p is even and q odd. (Symmetric to the previous one).

<u>*CASE 3*</u>: p and q are odd. (Let $p = 2p' + 1$ and $q = 2q' + 1$).

We will now use only the odd part of *SMerge*:

$$SMerge(\ \langle a_1\, a_3 \dots a_{n-1} \rangle,\ \langle b_1\, b_3 \dots b_{n-1} \rangle\ )\ =\ (\ \langle d_1\ \dots\ d_{\frac{n}{2}+1} \rangle,\ S_{odd}\ ).$$

In $\langle a_1\, a_2 \dots a_p \rangle$ there are $p'+1$ odd indices, namely $\{1, 3, \dots, 2p'+1\}$. Similarly, in $\langle b_1\, b_2 \dots b_q \rangle$ there are $q'+1$ odd indices, namely $\{1, 3, \dots, 2q'+1\}$. Hence, by IH we have $S_{odd} \cup \{a_1, \dots, a_{2p'+1}, b_1, \dots, b_{2q'+1}, \} \models_{up} d_{p'+q'+2}$ (note that, using that n is even, one can see that $1 \leq (p'+1) + (q'+1) \leq \frac{n}{2} + 1$).

Now, since $1 \leq p' + q' + 1 \leq \frac{n}{2}$, the clause $\overline{d}_{p'+q'+2} \vee c_{2p'+2q'+2}$ belongs to S, the literal $c_{2p'+2q'+2}$ can be unit propagated.

<u>*CASE 4*</u>: p and q are even. (Let $p = 2p'$ and $q = 2q'$).

We will now only use the even part of *SMerge*:

$$SMerge(\ \langle a_2\, a_4 \dots a_n \rangle,\ \langle b_2\, b_4 \dots b_n \rangle\ )\ =\ (\ \langle e_1\ \dots\ e_{\frac{n}{2}+1} \rangle,\ S_{even}\ )$$

In $\langle a_2\, a_4 \dots a_p \rangle$ there are p' even indices, namely $\{2, 4, \dots, 2p'\}$. Similarly, in $\langle b_2\, b_4 \dots b_q \rangle$ there are q' even indices, namely $\{2, 4, \dots, 2q'\}$. Hence, by IH we

have $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \} \models_{up} e_{p'+q'}$ (note that $1 \leq p' + q' \leq \frac{n}{2} + 1$).

Now, using that n is even, one can see that $1 \leq p' + q' \leq \frac{n}{2}$ and hence the clause $\overline{e}_{p'+q'} \vee c_{2p'+2q'}$ belongs to S, allowing one to propagate the literal $c_{2p'+2q'}$. □

Lemma 8. *Let SMerge*($\langle a_1 \ldots a_n \rangle$, $\langle b_1 \ldots b_n \rangle$) *be* ($\langle c_1 \ldots c_{n+1} \rangle$, S), *and* $p, q \in \mathbb{N}$ *with* $p + q \leq n$.

If $p < n$ *and* $q < n$ *then* $S \cup \{a_1, \ldots, a_p, b_1, \ldots, b_q, \overline{c}_{p+q+1}\} \models_{up} \overline{a}_{p+1}, \overline{b}_{q+1}$.
If $p = n$ *and* $q = 0$ *then* $S \cup \{a_1, \ldots, a_n, \overline{c}_{n+1}\} \models_{up} \overline{b}_1$.
If $p = 0$ *and* $q = n$ *then* $S \cup \{b_1, \ldots, b_n, \overline{c}_{n+1}\} \models_{up} \overline{a}_1$.

Lemma 9. *Given A and B sequences of length n, the Simplified Merging Network SMerge*(A, B) *contains* $O(n \log n)$ *clauses with* $O(n \log n)$ *auxiliary variables.*

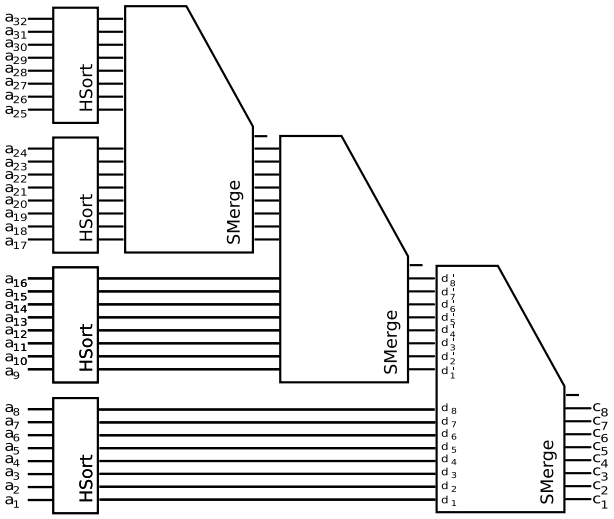

Fig. 2. Representation of *Card*$(\langle a_1 \ldots a_{32} \rangle, 8)$ with output $\langle c_1 \ldots c_8 \rangle$.

4.2 K-Cardinality Networks

Given a sequence A of length $n = m \times k$ with $k = 2^r$ and $m \in \mathbb{N}$, the *k-Cardinality Network of A*, denoted *Card*(A, k), is a pair (C, S), where C is a sequence of length k and S is a set of clauses, defined as follows.

For sequences of length k, we define:

$$\mathit{Card}(\ \langle a_1\ \ldots\ a_k\rangle,\ k\ )\ =\ \mathit{HSort}(\ \langle a_1\ \ldots\ a_k\rangle\ )$$

For sequences of length $n > k$ we define:

$$\mathit{Card}(\ \langle\, a_1\ \ldots\ a_n\,\rangle\,, k\ )\ =\ (\ \langle\, c_1\ \ldots\ c_k\,\rangle,\ \ S_D \cup S_{D'} \cup S_M\ )$$

recursively in terms of subsequences of sizes k and $n-k$:

$\mathit{Card}(\ \langle\, a_1\ \ldots\ a_k\ \rangle,\ k\ )\ =\ (\ \langle d_1\ \ldots\ d_k\rangle,\ S_D\ ),$
$\mathit{Card}(\ \langle a_{k+1} \ldots a_n\rangle,\ k\ )\ =\ (\ \langle d'_1\ \ldots\ d'_k\rangle,\ S_{D'}\ ),$

and a simplified merge of them (note that its last output is not used)

$\mathit{SMerge}(\ \langle d_1 \ldots d_k\ \rangle,\ \langle d'_1 \ldots d'_k\ \rangle\ )\ =\ (\ \langle c_1\ \ldots\ c_{k+1}\rangle,\ S_M\ ),$

Lemma 10. *Given a sequence A of length $n = m \times k$, the k-Cardinality Network $\mathit{Card}(A, k)$ contains $O(n \log^2 k)$ clauses with $O(n \log^2 k)$ auxiliary variables.*

Again, the usual properties of how zeros and ones are unit propagated follow. Their proofs are analogous to the ones of Lemma 5 and Lemma 6.

Lemma 11. *If $\mathit{Card}(\ A,\ k\ ) = (\langle\, c_1 \ldots c_k\,\rangle,\ S\ )$ and $A' \subseteq A$ with $|A'| = p \leq k$, then*

$$S\ \cup\ A'\ \models_{up}\ \ c_1, \ldots, c_p$$

Proof. Sequence A will be of the form $\langle a_1 \ldots a_n\rangle$ with $n = m \times k$. We will prove the lemma by induction on m.

If $m = 1$, we have $\mathit{Card}(\ \langle\, a_1,\ \ldots\ a_k\,\rangle, k)\ =\ \mathit{HSort}(\ \langle a_1,\ \ldots,\ a_k\rangle\ )$. Using lemma 5 we conclude that $\{c_1, \ldots, c_p\}$ are unit propagated.

For the induction step ($m > 1$) we have:

$$\begin{array}{lll}
\mathit{Card}(\ \langle\, a_1\ \ldots\ a_n\,\rangle\,, k\ ) & = & (\ \langle\, c_1\ \ldots\ c_k\,\rangle,\ \ S_D \cup S_{D'} \cup S_M\ ), \text{ with} \\
\quad \mathit{Card}(\ \langle\, a_1\ \ldots\ a_k\ \rangle,\ k\ ) & = & (\ \langle d_1\ \ldots\ d_k\rangle,\ S_D\ ), \\
\quad \mathit{Card}(\ \langle a_{k+1} \ldots a_n\rangle,\ k\ ) & = & (\ \langle d'_1\ \ldots\ d'_k\rangle,\ S_{D'}\ ) \text{ and} \\
\quad \mathit{SMerge}(\ \langle d_1 \ldots d_k\ \rangle,\ \langle d'_1 \ldots d'_k\ \rangle,\ ) & = & (\ \langle c_1\ \ldots\ c_{k+1}\rangle,\ S_M\ )
\end{array}$$

If we now consider the sets $A_D = A' \cap \{a_1, \ldots, a_k\}$, with size $|A_D| = p_D$, and $A_{D'} = A' \cap \{a_{k+1}, \ldots, a_n\}$, with $|A_{D'}| = p_{D'}$, by IH we have $A_D \cup S_D \models_{up} d_1, \ldots, d_{p_D}$ and $A_{D'} \cup S_{D'} \models_{up}\ d'_1, \ldots, d'_{p_{D'}}$. Now, by lemma 7 we know that $S_M \cup \{d_1, \ldots, d_{p_D}, d'_1, \ldots, d'_{p_{D'}}\} \models_{up}\ c_1, \ldots, c_{p_D+p_{D'}}$, which, since $p = p_D + p_{D'}$, concludes the proof. □

Theorem 1. *If $\mathit{Card}(\ \langle a_1 \ldots a_n\rangle,\ k\ ) = (\langle\, c_1 \ldots c_k\,\rangle,\ S\ )$ and $A' \subsetneq A$ with size $|A'| = p < k$, then*

$$S\ \cup\ A'\ \cup\ \overline{c}_{p+1} \models_{up}\ \overline{a}_j \text{ for all } a_j \in (A \setminus A')$$

Proof. We have that $n = m \times k$, and we will prove the lemma by induction on m.

If $m = 1$, we have $\mathit{Card}(\ \langle a_1, \ldots\ a_k\ \rangle, k)\ =\ \mathit{HSort}(\ \langle a_1, \ldots, a_k\rangle\ )$ and in this case the theorem amounts to Lemma 6.

For the induction step ($m > 1$) we have:

$$\begin{array}{rl}
\mathit{Card}(\ \langle\ a_1\ \ldots\ a_n\ \rangle\ , k\ ) & =\ (\ \langle\ c_1\ \ldots\ c_k\ \rangle,\ \ S_D \cup S_{D'} \cup S_M\ ), \text{ with} \\
\quad \mathit{Card}(\ \langle\ a_1\ \ldots\ a_k\ \ \rangle,\ k\ ) & =\ (\ \langle d_1\ \ldots\ d_k\rangle,\ S_D\ ), \\
\quad \mathit{Card}(\ \langle a_{k+1} \ldots a_n\rangle,\ k\ ) & =\ (\ \langle d'_1\ \ldots\ d'_k\rangle,\ S_{D'}\ ) \text{ and} \\
\quad \mathit{SMerge}(\ \langle d_1 \ldots d_k\ \rangle,\ \langle d'_1 \ldots d'_k\ \rangle,\ ) & =\ (\ \langle c_1\ \ldots\ c_{k+1}\rangle,\ S_M\ )
\end{array}$$

If we now consider the sets $A_D = A' \cap \{a_1, \ldots, a_k\}$, with size $|A_D| = p_D$, and $A_{D'} = A' \cap \{a_{k+1}, \ldots, a_n\}$, with $|A_{D'}| = p_{D'}$, by Lemma 11 we know that $A_D \cup S_D \models_{up} \ d_1, \ldots, d_{p_D}$ and $A_{D'} \cup S_{D'} \models_{up} \ d'_1, \ldots, d'_{p_{D'}}$. Due to these propagated literals and knowing that $p = p_D + p_{D'} \leq k$ and both $p_D < k$ and $p_{D'} < k$, we obtain $S_M \cup \{d_1, \ldots, d_{p_D}, d'_1, \ldots, d'_{p_{D'}}, \overline{c}_{p+1}\} \models_{up} \overline{d}_{p_D+1}, \overline{d}'_{p_{D'}+1}$ by applying Lemma 8.

Finally these two unit propagations allow us to use the IH to infer that $S_D \cup A_D \cup \overline{d}_{p_D+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_1 \ldots a_k\} - A_D)$ and also that $S_{D'} \cup A_{D'} \cup \overline{d}'_{p_{D'}+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_{k+1} \ldots a_n\} - A_{D'})$, which concludes the proof. □

5 Application to SAT Solving and Extensions

In this section we show how to apply the previous constructions in practice and we further present some extensions:

• **Use of *Card* in practice.** Theorem 1 indicates how to apply the construction *Card* in practice. Assume we are given a formula F to which we want to impose the cardinality constraint $a_1 + \ldots a_n \leq p$. We should first find k, the smallest power of two with $k > p$ and consider the construction $\mathit{Card}(\langle a_1 \ldots a_{n+m}\rangle, k) = (\ \langle c_1, \ldots c_k\rangle, S\ )$. Note that we may need to add m extra variables to the input sequence to obtain a sequence of size multiple of k, but these variables are initially set to false and do not enlarge the search space. Now, the problem amounts to check the satisfiability of $F \wedge S \wedge \overline{c}_{p+1}$ since, due to Theorem 1, as soon as p variables in $\langle a_1, \ldots, a_{n+m}\rangle$ are set to true, the remaining ones will be unit propagated to false, hence disallowing any model not satisfying the cardinality constraint.

• **Incremental strengthening.** Another important feature of these encodings can be exploited in applications where one needs to solve a sequence of problems that only differ in that a cardinality constraint $a_1 + \ldots + a_n \leq p$ becomes increasingly stronger by decreasing p to p', as it happens in optimization problems. In this setting, we only need to assert the corresponding literal $\overline{c}_{p'+1}$, and the search can be resumed keeping all lemmas generated in the previous problems. Most state-of-the-art SAT solvers used as black boxes provide a user interface for doing this.

• **Constraints of the form** $a_1 + \ldots a_n \geq p$**.** For these type of constraints, we should first find k, the smallest power of two with $k \geq p$. After that, we should consider a new construction $\mathit{Card}^{\geq}(\langle a_1, \ldots, a_{n+m}\rangle, k) = (\langle c_1, \ldots, c_k\rangle, S)$, identical to $\mathit{Card}(A, k)$, except that its blocks *HMerge* and *SMerge* contain, in their basic case, the clauses $\{a \vee b \vee \overline{c_1},\ \ a \vee \overline{c_2},\ \ b \vee \overline{c_2}\}$ and, for the recursive case, the clause set S' is built from the clauses $\{d_{i+1} \vee \overline{c_{2i+1}},\ \ e_i \vee \overline{c_{2i+1}},\ \ d_{i+1} \vee e_i \vee \overline{c_{2i}}\}$. We have the following result:

Theorem 2. *If* $\mathit{Card}^{\geq}(\langle a_1 \ldots a_n\rangle, k) = (\langle c_1 \ldots c_k\rangle, S)$ *and* $A' \subsetneq A$ *with* $|A'| = n - p$*, for some* $p \in \mathbb{N}$ *with* $1 \leq p \leq k$*, then*

$$S \cup \overline{A'} \cup c_p \models_{up} a_j \text{ for all } a_j \in (A \setminus A'),$$

where $\overline{A'}$ *contains the negation of all variables of* A'*.*

This theorem ensures that, if we set c_p to true, as soon as $n - p$ literals are set to false, the remaining p will be set to true, hence forcing the constraint to be satisfied.

• **Constraints of the form** $p \leq a_1 + \ldots a_n \leq q$**.** For these constraints, of which equality constraints $a_1 + \ldots + a_n = p$ are a particular case, we should first find k, the smallest power of two such that $k > q$. Then, we will use another construction $\mathit{Card}^{rng}(\langle a_1, \ldots, a_{n+m}\rangle, k) = (\langle c_1, \ldots, c_k\rangle, S)$, identical to $\mathit{Card}(A, k)$, except that its blocks *HMerge* and *SMerge* contain, in their basic and recursive cases, all 6 mentioned clauses (the ones for *Card* and the ones for $\mathit{Card}^{\geq}$). This allows one to avoid encoding the two constraints independently, which would roughly duplicate the number of variables. For this construction, we have:

Theorem 3. *Let* $\mathit{Card}^{rng}(\langle a_1 \ldots a_n\rangle, k) = (\langle c_1 \ldots c_k\rangle, S)$ *and* $A' \subsetneq A$*.*

- *If* $|A'| = n - p$ *for some* $p \in \mathbb{N}$ *with* $1 \leq p \leq k$ *then*

$$S \cup \overline{A'} \cup c_p \models_{up} a_j \text{ for all } a_j \in (A \setminus A'),$$

- *If* $|A'| = p$ *for some* $p < k$ *then*

$$S \cup A' \cup \overline{c}_{p+1} \models_{up} \overline{a}_j \text{ for all } a_j \in (A \setminus A')$$

This theorem ensures that, if we set c_p and $\overline{c}_{q+1}$, then (i) as soon as $n - p$ variables are set to false, the remaining ones will be set to true and (ii) as soon as q variables are set to true, the remaining ones will be set to false, which forces the constraint to be satisfied.

• **Constraints** $a_1 + \ldots + a_n \leq p$ **with** $p > \frac{n}{2}$. Note that Cardinality Networks were designed to improve upon Sorting Networks when n is much larger than p. If $p > \frac{n}{2}$ we can use the fact that the constraint above can be rewritten as $(1 - a_1) + \ldots + (1 - a_n) \geq n - p$. The latter constraint, where now $n - p < \frac{n}{2}$, can be encoded using Cardinality Networks by simply changing the input variables by their negations.

6 Evaluation

We first show some statistics, for a constraint $a_1 + \ldots + a_n \leq k$, about the number of variables and clauses in Cardinality Networks compared with the Sorting Networks of [Bat68,ES06] (figures for our Half Sorting Networks are as for Sorting Networks, except that the number of clauses is halved). Cardinality Networks provide a huge advantage for small values of k, whereas for $k = \frac{n}{2}$ (its worst case) there is still more than a factor-two advantage due to the use of Half Sorting/Merging Networks instead of full ones.

	Sorting Network		Cardinality Network							
			k=5		k=10		k=15		k=n/2	
n	vars	clauses	vars	clauses	vars	clauses	vars	clauses	vars	clauses
10^5	$18 \cdot 10^6$	$54 \cdot 10^6$	$77 \cdot 10^4$	$12 \cdot 10^5$	$12 \cdot 10^5$	$18 \cdot 10^5$	$12 \cdot 10^5$	$19 \cdot 10^5$	$15 \cdot 10^6$	$23 \cdot 10^6$
10^4	$15 \cdot 10^5$	$45 \cdot 10^5$	$77 \cdot 10^3$	$12 \cdot 10^4$	$12 \cdot 10^4$	$18 \cdot 10^4$	$12 \cdot 10^4$	$19 \cdot 10^4$	$12 \cdot 10^5$	$19 \cdot 10^5$
10^3	48150	144403	7713	12065	12223	18825	12857	19771	39919	59879
10^2	2970	8855	773	1205	1251	1917	1325	2023	2279	3419

We now also assess the practical performance of the encodings. To the best of our knowledge there is no standard library for SAT benchmarks with cardinality constraints. However, there exists a very large and diverse source of realistic instances, namely the ones produced by the *msu4* algorithm [MSP08] where Max-SAT problems are reduced to a series of SAT problems with cardinality constraints.

We have made a simple *msu4* implementation which, every time a non-trivial cardinality constraint is used (that is, that cannot be converted into a single clause or a set of unit literals), also writes the SAT + cardinality constraints problem into a file. We have run this prototype on all benchmarks used in the Partial Max-SAT division of the Third Max-SAT evaluation[2]. Hence, for every benchmark in this division (some 1800), we have created a *family* of SAT + cardinality constraints problems, usually between 2 and 10, which we believe constitute a large and diverse enough set of benchmarks. We run each one of them with Sorting and with Cardinality Networks on a 2Ghz Linux Quad-Core AMD using our Barcelogic SAT Solver that ranked 3rd in the 2008 SAT-Race[3]. Results are plotted in Figure 3, which shows a clear win for Cardinality Networks. Each cross represents the time to solve a family of benchmarks. Each benchmark was given 600 seconds and timing out in a single benchmark is counted as a timeout for the whole family (in the plot, these are the crosses in the vertical or horizontal lines).

One may wonder where the improvements come from the use of Half Merging/Sorting Networks (3 clauses instead of 6) or from the asymptotically smaller Cardinality Networks ($O(n \log^2 k)$ clauses and auxiliary variables vs. $O(n \log^2 n)$).

[2] See `http://www.maxsat.udl.cat/08/index.php?disp=submitted-benchmarks`.
[3] See `http://baldur.iti.uka.de/sat-race-2008/`.

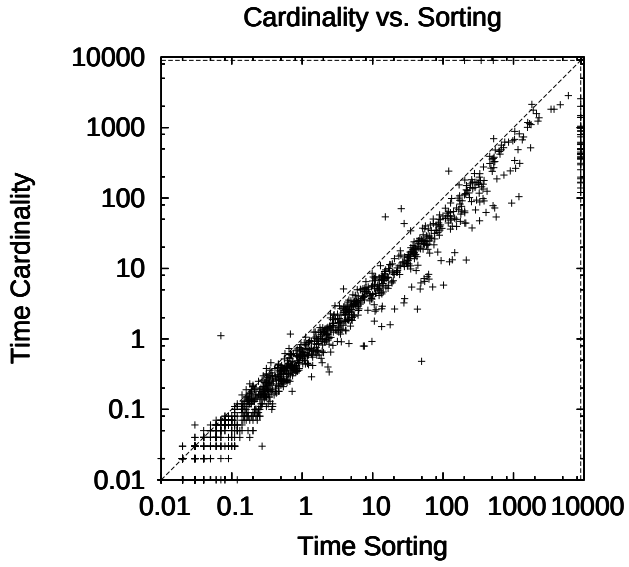

Fig. 3. Times in seconds and logarithmic scale is used.

The answer is: both, as one can see from Figure 4, where we also compare with **3-Sorting:** Half Sorting Networks as described in Section 3.2, and **6-Cardinality:** Cardinality Networks with *HMerge* and *SMerge* using all 6 clauses instead of only the 3 mentioned in Section 3.1 and Section 4.1. In particular, using 3 clauses has beneficial effects for both Sorting Networks and Cardinality Constraints.

7 Conclusions and Further Work

SAT solvers can be used off the shelf, giving high performance push-button tools, i.e., tools that require no tuning for variable or value selection heuristics. In order to exploit these features optimally, it is important to develop a catalogue of encodings for the most common general-purpose constraints, in such a way that the SAT solver's unit propagation can efficiently preserve arc consistency.

The cardinality constraints we have studied here are certainly among the most ubiquitous ones. Therefore, apart from the aforementioned work [ES06], they have also been studied elsewhere. For instance in [Sin05] two encodings are given, one requiring $7n$ clauses and $2n$ auxiliary variables, and another one based on n unary $k-bit$ counters c_i denoting the number of true inputs among $x_1 \dots x_i$; this latter encoding preserves arc consistency like ours, but it requires $O(n \cdot k)$ clauses and new variables. In [SL07] the case of $k = 1$ is studied, showing how a state-of-the-art SAT solver can be adapted to diminish the noise introduced by the auxiliary variables.

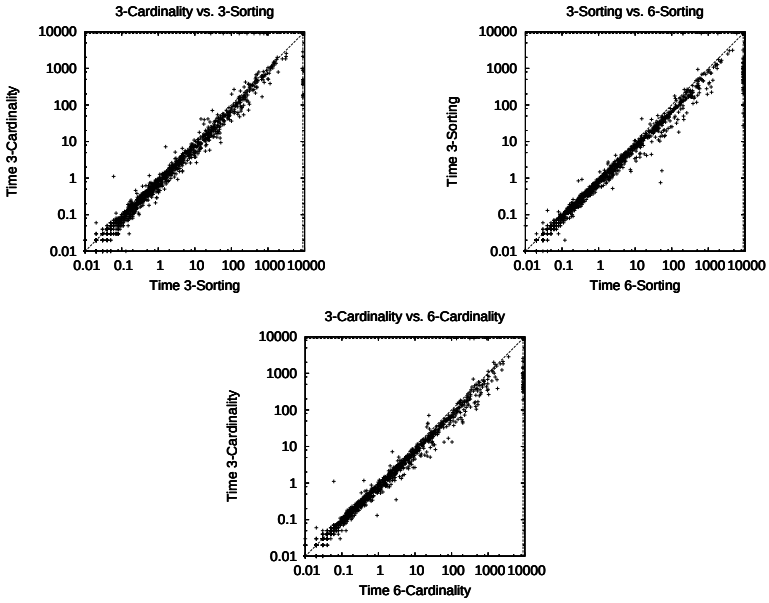

Fig. 4. Times in seconds and logarithmic scale is used. Best settings on the y-axes.

Our approach is based on precise (recursive) definitions of the generated clause sets and on inductive proofs for the arc consistency properties, combined with a careful quantitative and experimental analysis.

We believe that in a similar way it will be possible to go beyond, re-visiting pseudo-boolean constraints and other important constraints that are well known in the Constraint Programming community.

References

[Bat68] K. E. Batcher. Sorting Networks and their Applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM, CACM*, 5(7):394–397, 1962.

[DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM, JACM*, 7(3):201–215, 1960.

[ES06] N . Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[MSP08] J. Marques-Silva and J. Planes. Algorithms for Maximum Satisfiability usint Unsatisfiable Cores. In *2008 Conference on Design, Automation and Test in Europe, DATE'08*, pages 408–413. IEEE Computer Society, 2008.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, Sitges, Spain, October 2005.

[SL07] João P. Marques Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2007, SPringer LNCS 4741*, pages 483–497, 2007.