Cardinality Networks: A theoretical and empirical study

Article *in* Constraints · April 2011

DOI: 10.1007/s10601-010-9105-0 · Source: DBLP

4 authors:

Roberto Javier Asín Achá
University of Concepción
12 PUBLICATIONS **234** CITATIONS
SEE PROFILE

Robert Nieuwenhuis
Universitat Politècnica de Catalunya
117 PUBLICATIONS **3,493** CITATIONS
SEE PROFILE

Albert Oliveras
Universitat Politècnica de Catalunya
47 PUBLICATIONS **2,436** CITATIONS
SEE PROFILE

Enric Rodríguez-Carbonell
Universitat Politècnica de Catalunya
44 PUBLICATIONS **1,041** CITATIONS
SEE PROFILE

Constraints manuscript No.
(will be inserted by the editor)

Cardinality Networks: a Theoretical and Empirical Study*

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

Received: date / Accepted: date

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 [8] 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 the relation $\#$ belongs to $\{<, \leq, =, \geq, >\}$.

Cardinality constraints appear in many practical problems, 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

* Parts of this paper were presented in a shorter preliminary form at the SAT'2009 conference with the title "Cardinality Networks and their Applications".

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.

From Technical Univ. of Catalonia, Barcelona

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 $\{C_1 \vee x_1, \ldots, C_n \vee x_n\}$ 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 [8]. 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 [8] 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 [8] 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 [5], 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 $k \leq x_1 + \ldots + x_n \leq k'$.

We review the existing literature on cardinality constraints in Section 6 and in Section 7 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 8.

A preliminary version of this paper was accepted at the SAT'2009 conference. This paper is extended in different directions: (i) the constructions are illustrated with some

examples, (ii) all proofs of propagation properties have been included, (iii) all proofs regarding the size of our constructions have been worked out in detail and included, (iv) a thorough study of the literature has been added (Section 6), (v) we have extended the benchmarks used in the experiments with the ones present in the literature, (vi) we have compared our new encoding with three additional approaches to cardinality constraints: one encoding using adders, another one using BDDs and an SMT-based approach.

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 $\overline{p}$. 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, i.e., no contradictory literals appear.. 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 [7]. 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. Note that this latter property is motivated by the fact that our interest is in finding encodings that work well with DPLL-based SAT solvers, all of which implement unit propagation.

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 [5,8], 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 $m \geq 0$. In Section 5 we explain how to treat the case where n is not a power of two.

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:

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

recursively in terms of the odd and the even subsequences:

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

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 $\langle a_1 \ldots a_n\rangle$ and $\langle b_1 \ldots b_n\rangle$ that are already sorted into a single sorted output sequence $\langle c_1 \ldots c_{2n}\rangle$, and the required unit propagation is that if $a_1 \ldots a_p$ and $b_1 \ldots b_q$ 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:

$$\left\{\begin{array}{l}\overline{a}_1\vee\overline{b}_1\vee d_2\\ \overline{a}_1\vee d_1\\ \overline{b}_1\vee d_1\end{array}\right.\qquad \left\{\begin{array}{l}\overline{a}_2\vee\overline{b}_2\vee e_2\\ \overline{a}_2\vee e_1\\ \overline{b}_2\vee e_1\end{array}\right.\qquad \left\{\begin{array}{l}\overline{d}_2\vee\overline{e}_1\vee c_3\\ \overline{d}_2\vee c_2\\ \overline{e}_1\vee c_2\end{array}\right.$$

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*. If we set a_1 and b_1 to true, and 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* $0 \le p, q \le n$, *then* $S \cup \{a_1\ \dots\ a_p\ b_1 \dots\ b_q\} \models_{up}\ c_1, \dots, c_{p+q}$.

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

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

If $p = q = 0$ there is nothing to prove. If $p = 1$ and $q = 0$ it is obvious that setting a propagates c_1, and the case $p = 0$ and $q = 1$ is symmetric. Finally, if $p = q = 1$, setting a and b propagates c_1 and c_2.

For the induction step ($n > 1$) we consider four different cases, depending on the parity of p and q:

<u>*CASE 1*</u>*:* p is odd and q even. (Let $p = 2p' + 1$ and $q = 2q'$). In this case we have $p' \le \frac{n}{2} - 1$ and $q' \le \frac{n}{2}$.

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

$\mathit{HMerge}(\ \langle a_1\, a_3 \dots a_{n-1}\rangle,\ \langle b_1\, b_3 \dots b_{n-1}\rangle\ )\ =\ (\ \langle d_1\ \dots\ d_n\rangle,\ S_{odd}\ \ )$.

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 $\{1, 3, \dots, 2q'-1\}$. Since $0 \le p'+1, q' \le \frac{n}{2}$, by IH we have $S_{odd} \cup \{a_1, \dots, a_{2p'+1}, b_1, \dots, b_{2q'-1}\} \models_{up} d_1, \dots, d_{p'+q'+1}$

Let us now take the even part of *HMerge*:

$$\mathit{HMerge}(\ \langle a_2\, a_4 \dots a_n\rangle,\ \langle b_2\, b_4 \dots b_n\rangle\ )\ =\ (\ \langle e_1\ \dots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \dots a_p\rangle$ there are p' even indices, namely $\{2, 4, \dots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \dots b_q\rangle$ there are q' even indices, namely $\{2, 4, \dots, 2q'\}$. Since we know $0 \le p', q' \le \frac{n}{2}$, by IH we have $S_{even} \cup \{a_2, \dots, a_{2p'}, b_2, \dots, b_{2q'}\} \models_{up} e_1, \dots, e_{p'+q'}$.

Finally, considering the set $S' = \bigcup_{i=1}^{n-1}\{\ \overline{d}_{i+1} \vee \overline{e_i} \vee c_{2i+1},\ \ \overline{d}_{i+1} \vee c_{2i},\ \ \overline{e_i} \vee c_{2i}\ \}$ we can check that: c_1 is propagated because it is d_1, the even c's $(c_2, \dots, c_{2p'+2q'})$ are propagated due to the e's and the third clauses, and $c_3, \dots, c_{2p'+2q'+1}$ are propagated thanks to the d's, the e's and the first clauses.

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

<u>*CASE 3*</u>*:* p and q are odd. (Let $p = 2p' + 1$ and $q = 2q' + 1$). In this case we have $p' \le \frac{n}{2} - 1$ and $q' \le \frac{n}{2} - 1$.

Let us consider the odd part of *HMerge*:

$$\mathit{HMerge}(\ \langle a_1\, a_3 \dots a_{n-1}\rangle,\ \langle b_1\, b_3 \dots b_{n-1}\rangle\ )\ =\ (\ \langle d_1\ \dots\ d_n\rangle,\ S_{odd}\ \ ).$$

In $\langle a_1, a_2, \dots a_p\rangle$ there are $p' + 1$ odd indices, namely $\{1, 3, \dots, 2p' + 1\}$. Similarly, $\langle b_1, b_2, \dots b_q\rangle$ there are also $q' + 1$ odd indices, namely $\{1, 3, \dots, 2q' + 1\}$. Since we know $0 \le p' + 1, q' + 1 \le \frac{n}{2}$, by IH we have $S_{odd} \cup \{a_1, \dots, a_{2p'+1}, b_1, \dots, b_{2q'+1}\} \models_{up} d_1, \dots, d_{p'+q'+2}$

Let us now take the even part of *HMerge*:

$$\mathit{HMerge}(\ \langle a_2\, a_4 \dots a_n\rangle,\ \langle b_2\, b_4 \dots b_n\rangle\ )\ =\ (\ \langle e_1\ \dots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \dots a_p\rangle$ there are p' even indices, namely $\{2, 4, \dots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \dots b_q\rangle$ there are q' even indices, namely $\{2, 4, \dots, 2q'\}$. Since we know $0 \le p', q' \le \frac{n}{2}$, by IH we have $S_{even} \cup \{a_2, \dots, a_{2p'}, b_2, \dots, b_{2q'}\} \models_{up} e_1, \dots, e_{p'+q'}$.

If we now consider the set $S' = \bigcup_{i=1}^{n-1}\{\ \overline{d}_{i+1} \vee \overline{e_i} \vee c_{2i+1},\ \ \overline{d}_{i+1} \vee c_{2i},\ \ \overline{e_i} \vee c_{2i}\ \}$ we see that: c_1 is propagated because it is d_1, the even c's $(c_2, \dots, c_{2p'+2q'+2})$ are propagated due to the d's and the second clauses, and $c_3, \dots, c_{2p'+2q'+1}$ thanks to the d's, the e's and the first clauses.

CASE 4: p and q are even. (Let $p = 2p'$ and $q = 2q'$). In this case we have $p' \leq \frac{n}{2}$ and $q' \leq \frac{n}{2}$.

Let us consider the odd part of *HMerge*:

$$HMerge(\ \langle a_1\, a_3 \ldots a_{n-1}\rangle,\ \langle b_1\, b_3 \ldots b_{n-1}\rangle\ ) \ =\ (\ \langle d_1\ \ldots\ d_n\rangle,\ S_{odd}\ \ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' odd indices, namely $\{1, 3, \ldots, 2p'-1\}$. Similarly, in the sequence $\langle b_1, b_2, \ldots b_q\rangle$ there are also q' odd indices, namely $\{1, 3, \ldots, 2q'-1\}$. Since $0 \leq p', q' \leq \frac{n}{2}$, by IH we have $S_{odd} \cup \{a_1, \ldots, a_{2p'-1}, b_1, \ldots, b_{2q'-1}\} \models_{up} d_1, \ldots, d_{p'+q'}$

Let us now take the even part of *HMerge*:

$$HMerge(\ \langle a_2\, a_4 \ldots a_n\rangle,\ \langle b_2\, b_4 \ldots b_n\rangle\ ) \ =\ (\ \langle e_1\ \ldots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Since we know $0 \leq p', q' \leq \frac{n}{2}$, by IH we have $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}\} \models_{up} e_1, \ldots, e_{p'+q'}$.

If we now consider the set $S' = \bigcup_{i=1}^{n-1}\{\ \overline{d}_{i+1} \vee \overline{e}_i \vee c_{2i+1},\quad \overline{d}_{i+1} \vee c_{2i},\quad \overline{e}_i \vee c_{2i}\ \}$ we can check that: c_1 is propagated because it is d_1, the even c's ($c_2, \ldots, c_{2p'+2q'}$) are propagated due to the e's and the third clauses, and $c_3, \ldots, c_{2p'+2q'-1}$ thanks to the d's, the e's and the first clauses. Note that in the case of $p = q = n$, then $c_{2p'+2q'}$ is indeed e_n, which is in turn $e_{p'+q'}$ and is propagated not due to S' but thanks to the even part of *HMerge*. □

Lemma 2 *Let* $HMerge(\ \langle a_1 \ldots a_n\rangle,\ \langle b_1 \ldots b_n\rangle\ )$ *be* $(\ \langle c_1\ \ldots\ c_{2n}\,\rangle,\quad S\ )$*, 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* $q < n$ *then* $S \cup \{a_1, \ldots, a_p, b_1, \ldots, b_q, \overline{c}_{p+q+1}\} \models_{up} \overline{b}_{q+1}$.
If $p < n$ *and* $q = n$ *then* $S \cup \{a_1, \ldots, a_p, b_1, \ldots, b_q, \overline{c}_{p+q+1}\} \models_{up} \overline{a}_{p+1}$.

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

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

If $p = q = 0$ we can see that setting $\overline{c_1}$ propagates $\overline{a}$ and $\overline{b}$ as expected. If $p = 0$ and $q = 1$ then setting b and $\overline{c_2}$ propagates $\overline{a}$. Finally, if $p = 1$ and $q = 0$, setting a and $\overline{c_2}$ propagates $\overline{b}$.

For the induction step ($n > 1$) we consider again four different cases, depending on the parity of p and q.

CASE 1: p is odd and q even. (Let $p = 2p' + 1$ and $q = 2q'$). In this case we have $p' \leq \frac{n}{2} - 1$ and $q' \leq \frac{n}{2}$.

Let us first notice that if $p' + q' + 1 \leq n - 1$ then the clauses $\overline{d}_{p'+q'+2} \vee c_{2(p'+q'+1)}$ and $\overline{e}_{p'+q'+1} \vee c_{2(p'+q'+1)}$ belong to S. In this case, due to $\overline{c}_{p+q+1}$, which corresponds to $\overline{c}_{2(p'+q'+1)}$ we can propagate $\overline{d}_{p'+q'+2}$ and $\overline{e}_{p'+q'+1}$. Otherwise, if $p'+q'+1 > n-1$, it has to be that $q = n$ and $p = n - 1$. In this case, we can also assume that $\overline{e}_{p'+q'+1}$ holds, because it is $\overline{e}_n$, which corresponds to $\overline{c}_{p+q+1}$. Knowing that in both cases $\overline{e}_{p'+q'+1}$ holds, let us focus on the even part of *HMerge*:

$$HMerge(\ \langle a_2\, a_4 \ldots a_n\rangle,\ \langle b_2\, b_4 \ldots b_n\rangle\ ) \ =\ (\ \langle e_1\ \ldots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Since $p' < \frac{n}{2}$, by IH we have $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \overline{e}_{p'+q'+1}\} \models_{up} \overline{a}_{p+1}$.

Now, if $q < n$ we should also show that $\overline{b}_{q+1}$ is propagated. In this case we have that $q' < \frac{n}{2}$ and also that $\overline{d}_{p'+q'+2}$ is propagated. Take now the odd part of *HMerge*:

$$HMerge(\ \langle a_1\, a_3 \ldots a_{n-1}\rangle,\ \langle b_1\, b_3 \ldots b_{n-1}\rangle\ ) \ =\ (\ \langle d_1\ \ldots\ d_n\rangle,\ S_{odd}\ \ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are $p'+1$ odd indices, namely $\{1, 3, \ldots, 2p'+1\}$. Similarly, in $\langle b_1, b_2, \ldots b_q\rangle$ there are q' odd indices, namely $\{1, 3, \ldots, 2q'-1\}$. Since $q' < \frac{n}{2}$, by IH we have $S_{odd} \cup \{a_1, \ldots, a_{2p'+1}, b_1, \ldots, b_{2q'-1}, \overline{d}_{p'+q'+2}\} \models_{up} \overline{b}_{q+1}$

CASE 2: p is even and q is odd. (Symmetric to the previous one).

CASE 3: p and q are odd. (Let $p = 2p'+1$ and $q = 2q'+1$). In this case we have $p' \leq \frac{n}{2} - 1$ and $q' \leq \frac{n}{2} - 1$.

Let us consider the odd part of *HMerge*:

$$HMerge(\ \langle a_1\, a_3 \ldots a_{n-1}\rangle,\ \langle b_1\, b_3 \ldots b_{n-1}\rangle\ )\ =\ (\ \langle d_1\ \ldots\ d_n\rangle,\ S_{odd}\ \ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are $p'+1$ odd indices, namely $\{1, 3, \ldots, 2p'+1\}$. Similarly, in $\langle b_1, b_2, \ldots b_q\rangle$ there are also $q'+1$ odd indices, namely $\{1, 3, \ldots, 2q'+1\}$. Since $p'+1 \leq \frac{n}{2}$ and $q'+1 \leq \frac{n}{2}$ by Lemma 1 we have that $S_{odd} \cup \{a_1, \ldots, a_{2p'+1}, b_1, \ldots, b_{2q'+1}\} \models_{up} d_{p'+q'+2}$

Now, since $p'+q'+1 \leq n-1$ we know that $\overline{d}_{p'+q'+2} \vee \overline{e}_{p'+q'+1} \vee c_{2(p'+q'+1)+1}$ belongs to S and together with $\overline{c}_{p+q+1}$ and $d_{p'+q'+2}$ it allows us to propagate $\overline{e}_{p'+q'+1}$.

Let us now take the even part of *HMerge*:

$$HMerge(\ \langle a_2\, a_4 \ldots a_n\rangle,\ \langle b_2\, b_4 \ldots b_n\rangle\ )\ =\ (\ \langle e_1\ \ldots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Since we know $p' < \frac{n}{2}$ and $q' < \frac{n}{2}$, by IH we have $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \overline{e}_{p'+q'+1}\} \models_{up} \overline{a}_{p+1}, \overline{b}_{q+1}$

CASE 4: p and q are even. (Let $p = 2p'$ and $q = 2q'$). In this case we have $p' \leq \frac{n}{2}$ and $q' \leq \frac{n}{2}$.

Since for $p = q = n$ there is nothing to prove, let us assume that $q < n$ (case $p < n$ is symmetric). Hence we know that $q' \leq \frac{n}{2} - 1$.

Let us first consider the even part of *HMerge*:

$$HMerge(\ \langle a_2\, a_4 \ldots a_n\ \ \rangle,\ \langle b_2\, b_4 \ldots b_n\ \ \rangle\ )\ =\ (\ \langle e_1\ \ldots\ e_n\rangle,\ S_{even}\ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_1, b_2, \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Since $p', q' \leq \frac{n}{2}$, by Lemma 1 we have $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}\} \models_{up} e_{p'+q'}$.

Now, since $p'+q' \leq n-1$, the clause $\overline{d}_{p'+q'+1} \vee \overline{e}_{p'+q'} \vee c_{2p'+2q'+1}$ belongs to S. Together with $e_{p'+q'}$ and $\overline{c}_{p+q+1}$ this propagates $\overline{d}_{p'+q'+1}$.

The odd part of *HMerge* will now finish the work:

$$HMerge(\ \langle a_1\, a_3 \ldots a_{n-1}\rangle,\ \langle b_1\, b_3 \ldots b_{n-1}\rangle\ )\ =\ (\ \langle d_1\ \ldots\ d_n\rangle,\ S_{odd}\ \ ).$$

In $\langle a_1, a_2, \ldots a_p\rangle$ there are p' odd indices, namely $\{1, 3, \ldots, 2p'-1\}$. Similarly, in $\langle b_1, b_2, \ldots b_q\rangle$ there are also q' odd indices, namely $\{1, 3, \ldots, 2q'-1\}$. If $p' = \frac{n}{2}$ then, since also $q' < \frac{n}{2}$, by IH $S_{odd} \cup \{a_1, \ldots, a_{2p'-1}, b_1, \ldots, b_{2q'-1}, \overline{d}_{p'+q'+1}\} \models_{up} \overline{b}_{q+1}$. If $p' < \frac{n}{2}$ then, since also $q' < \frac{n}{2}$ by IH we can conclude that the propagation $S_{odd} \cup \{a_1, \ldots, a_{2p'-1}, b_1, \ldots, b_{2q'-1}, \overline{d}_{p'+q'+1}\} \models_{up} \overline{a}_{p+1}, \overline{b}_{q+1}$ holds. □

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.*

Proof Let a_m be the number of clauses of $HMerge(A, B)$ if A has length 2^m. It is easy to see that we have the following recurrence:

$$\begin{cases} a_0 = 3 \\ a_m = 2a_{m-1} + 3(2^m - 1) \ \text{ for all } m > 0 \end{cases}$$

with solution $a_m = 3 + 3m \cdot 2^m$. Since $m = \log n$ this proves the result about the number of clauses.

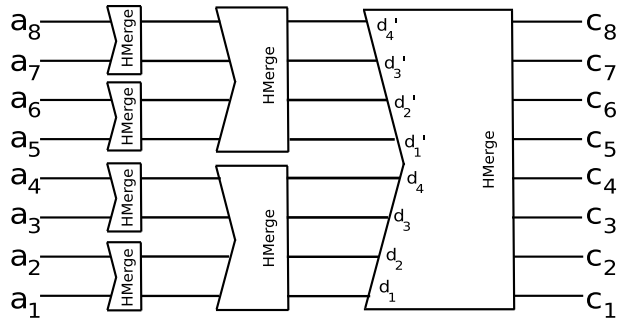

Fig. 1 *HSort* with input $\langle a_1 \dots a_8 \rangle$ and output $\langle c_1 \dots c_8 \rangle$

Regarding the number of auxiliary variables, if b_m is the number of auxiliary variables for inputs of length 2^m, we have the recurrence:

$$\begin{cases} b_0 = 2 \\ b_m = 2b_{m-1} + 2^{m+1} - 2 \ \text{ for all } m > 0 \end{cases}$$

with solution $b_m = 2 + m \cdot 2^{m+1}$. Replacing m by $\log n$ this gives the desired result. □

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:

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

recursively in terms of two subsequences of size n:

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

and the merge of them

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

In Figure 1 one can observe the structure of Half Sorting Networks.

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.*

Proof Let a_m be the number of clauses of $HSort(A)$ if A has length 2^m. It is easy to see that we have the following recurrence:

$$\begin{cases} a_1 = 3 \\ a_m = 2a_{m-1} + (3 + 3(m-1) \cdot 2^{m-1}) \ \text{ for all } m > 1 \end{cases}$$

with solution $a_m = -3 + 3 \cdot 2^m - 3m \cdot 2^{m-2} + 3m^2 \cdot 2^{m-2}$. Since $m = \log n$ this proves the result about the number of clauses.

Regarding the number of auxiliary variables, if b_m is the number of auxiliary variables for inputs of length 2^m, we have the recurrence:

$$\begin{cases} b_1 = 2 \\ b_m = 2b_{m-1} + (2 + (m-1) \cdot 2^m) \ \text{ for all } m > 1 \end{cases}$$

with solution $b_m = (8 - m + m^2) \cdot 2^{m-1} - 2$. Replacing m by $\log n$ this gives the desired result. □

Similar properties to the ones of Half Merging Networks also hold here, but without the requirement that the input ones form a prefix: (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,*

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

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

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

If $p = 0$ there is nothing to prove. If $p = 1$ then either a or b are true, and c_1 is unit propagated. Otherwise, $p = 2$, both a and b are true, and c_1 and then c_2 are propagated.

For the induction step ($n > 1$), let $A = \{a_1, \dots, a_{2n}\}$. We have:

$$\begin{array}{ll} HSort(\, \langle a_1 \dots a_{2n} \rangle\,) & =(\, \langle c_1 \dots c_{2n} \rangle,\ \ S_D \cup S_{D'} \cup S_M\,), \\ \text{with } HSort(\, \langle\, a_1 \dots a_n \,\rangle\,) & =(\, \langle d_1 \dots d_n \rangle,\ S_D\,), \\ \quad HSort(\, \langle a_{n+1} \dots a_{2n} \rangle\,) & =(\, \langle d'_1 \dots d'_n \rangle,\ S_{D'}\,) \text{ and} \\ \quad HMerge(\, \langle d_1 \dots d_n \,\rangle,\ \langle d'_1 \dots d'_n \,\rangle\,) & =(\, \langle c_1 \dots c_{2n} \rangle,\ S_M\,). \end{array}$$

If we now consider the set $A_D = A' \cap \{a_1, \dots, a_n\}$, with size $|A_D| = p_D$, and also $A_{D'} = A' \cap \{a_{n+1}, \dots, a_{2n}\}$, with $|A_{D'}| = p_{D'}$, by IH we have $A_D \cup S_D \models_{up} d_1, \dots, d_{p_B}$ and $A_{D'} \cup S_{D'} \models_{up} d'_1, \dots, d'_{p_{B'}}$. Finally, using these unit propagations and Lemma 1 we know that $S_M \cup \{d_1, \dots, d_{p_D}, d'_1, \dots, d'_{p_{D'}}\} \models_{up} c_1, \dots, c_{p_D + p_{D'}}$, which concludes the proof since $p_D + p_{D'} = p$. □

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

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

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

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

If $p = 0$ and we set $\overline{c_1}$, clearly $\overline{a}$ and $\overline{b}$ are unit propagated. If $p = 1$ and we set $\overline{c_2}$ and a, we can propagate $\overline{b}$; similarly, if we set $\overline{c_2}$ and b, we propagate $\overline{a}$.

For the induction step ($n > 1$), let $A = \{a_1, \ldots, a_{2n}\}$. We have:

$$\begin{array}{ll} HSort(\langle a_1 \ldots a_{2n}\rangle) & =(\langle c_1 \ldots c_{2n}\rangle,\ \ S_D \cup S_{D'} \cup S_M\ ),\\ \text{with } HSort(\langle a_1 \ldots a_n\rangle) & =(\langle d_1 \ldots d_n\rangle,\ S_D\ ),\\ \quad HSort(\langle a_{n+1} \ldots a_{2n}\rangle) & =(\langle d'_1 \ldots d'_n\rangle,\ S_{D'}\ ) \text{ and}\\ \quad HMerge(\langle d_1 \ldots d_n\rangle,\ \langle d'_1 \ldots d'_n\rangle) & =(\langle c_1 \ldots c_{2n}\rangle,\ S_M\ ). \end{array}$$

If we now consider the set $A_D = A' \cap \{a_1, \ldots, a_n\}$, with size $|A_D| = p_D$, and also $A_{D'} = A' \cap \{a_{n+1}, \ldots, a_{2n}\}$, with $|A_{D'}| = p_{D'}$, using Lemma 5 we can infer that $A_D \cup S_D \models_{up} d_1, \ldots, d_{p_D}$ and $A_{D'} \cup S_{D'} \models_{up} d'_1, \ldots, d'_{p_{D'}}$.

If $p_D < n$ and $p_{D'} < n$, then $S_M \cup \{d_1, \ldots, d_{p_D}, d'_1, \ldots, d'_{p_{D'}}, \overline{c_{p+1}}\} \models_{up} \overline{d}_{p_D+1}, \overline{d'}_{p_{D'}+1}$ by Lemma 2 (note that $p = p_D + p_{D'}$). With these two unit propagations we can use the IH to obtain the propagations $S_D \cup A_D \cup \overline{s}_{p_D+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_1 \ldots a_n\} - A_D)$ and $S_{D'} \cup A_{D'} \cup \overline{d}_{p_{D'}+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_{n+1} \ldots a_{2n}\} - A_{D'})$, which concludes the proof in this case.

If $p_D = n$ and $p_{D'} < n$, then $S_M \cup \{d_1, \ldots, d_{p_D}, d'_1, \ldots, d'_{p_{D'}}, \overline{c_{p+1}}\} \models_{up} \overline{d'}_{p_{D'}+1}$ by Lemma 2. Now, using the IH we obtain that $S_{D'} \cup A_{D'} \cup \overline{d}_{p_{D'}+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_{n+1} \ldots a_{2n}\} - A_{D'})$, which concludes the proof in this case since $A - A' = \{a_{n+1}, \ldots, a_{2n}\} - A_{D'}$.

Otherwise, $p_D < n$ and $p_{D'} = n$ and the proof is analogous to the previous case.

□

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 [8]. A main ingredient for Cardinality Networks are the *Simplified Half-Merging Networks*, which we introduce first.

4.1 Simplified Half-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 Half-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

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

The case $n > 1$ is defined

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

recursively in terms of the odd and the even subsequences,

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

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

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*:

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

In $\langle a_1\, a_2 \ldots a_p\rangle$ there are $p'+1$ odd indices, namely $\{1, 3, \ldots, 2p'+1\}$. Similarly, in $\langle b_1\, b_2 \ldots b_q\rangle$ there are q' odd indices, namely $\{1, 3, \ldots, 2q'-1\}$. Hence, by IH we have $S_{odd} \cup \{a_1, \ldots, a_{2p'+1}, b_1, \ldots, b_{2q'-1}\} \models_{up} d_{p'+q'+1}$ (note that $1 \leq (p'+1) + q' \leq \frac{n}{2}+1$). Now, if $p' = q' = 0$ we know that $d_{p'+q'+1}$ is d_1, which is is turn c_1, and hence the result holds. Hence, from now on, let us assume that $1 \leq p' + q'$.

Let us take the even part of *SMerge*:

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

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

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).

CASE 3: p and q are odd. (Let $p = 2p' + 1$ and $q = 2q' + 1$).

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

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

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

CASE 4: p and q are even. (Let $p = 2p'$ and $q = 2q'$).

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

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

In $\langle a_2\, a_4 \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in $\langle b_2\, b_4 \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Hence, using the IH we have that $S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \} \models_{up} e_{p'+q'}$ (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* $S \cup \{b_1, \ldots, b_n, \overline{c}_{n+1}\} \models_{up}\ \overline{a}_1$.

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

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

Hence, if $p = 0$ and $q = 0$, clearly setting $\overline{c_1}$ unit propagates $\overline{a}$ and $\overline{b}$. If $p = 1$ and $q = 0$, setting $\overline{c_2}$ and a propagates $\overline{b}$. Finally, if $p = 0$ and $q = 0$, setting $\overline{c_2}$ and b propagates $\overline{a}$.

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

CASE 1: p is odd and q is even. (Let $p = 2p' + 1$ and $q = 2q'$). We only need to prove the first property because the other conditions cannot hold.

Since $1 \leq p' + q' + 1 \leq \frac{n}{2}$, both $\overline{d}_{p'+q'+2} \vee c_{2p'+2q'+2}$ and $\overline{e}_{p'+q'+1} \vee c_{2p'+2q'+2}$ belong to S. Hence, setting $\overline{c}_{2p'+2q'+2}$ unit propagates $\overline{d}_{p'+q+2}$ and $\overline{e}_{p'+q'+1}$.

Let us consider the odd part of *SMerge*:

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

In $\langle a_1\, a_2 \ldots a_p\rangle$ there are $p' + 1$ odd indices, namely $\{1, 3, \ldots, 2p' + 1\}$. Similarly, in $\langle b_1\, b_2 \ldots b_q\rangle$ there are q' odd indices, namely $\{1, 3, \ldots, 2q' - 1\}$. Now, since $p < n$ and $q < n$ imply $p' + 1 < \frac{n}{2}$ and $q' < \frac{n}{2}$, and we know that $p' + q' + 1 \leq \frac{n}{2}$, by IH we infer that

$$S_{odd} \cup \{a_1, \ldots, a_{2p'+1}, b_1, \ldots, b_{2q'-1}, \overline{d}_{p'+q'+2}\} \models_{up}\ \overline{a}_{2p'+3}, \mathbf{\overline{b}_{2q'+1}}.$$

For propagating $\overline{a}_{2p'+2}$, let us take the even part of *SMerge*:

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

In $\langle a_2\, a_4 \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in $\langle b_2\, b_4 \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Hence, since $p' < \frac{n}{2}$, $q' < \frac{n}{2}$ and $p' + q' \leq \frac{n}{2}$, by IH we conclude the proof since

$S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \overline{e}_{p'+q'+1}\} \models_{up} \overline{\mathbf{a}}_{\mathbf{2p'+2}}, \overline{b}_{2q'+2}$.

CASE 2: p is even and q odd. (Symmetric to the previous one).

CASE 3: p and q are odd. (Let $p = 2p' + 1$ and $q = 2q' + 1$). Again, we only need to prove the first property.

Let us consider the odd part of *SMerge*:

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

In $\langle a_1\, a_2 \ldots a_p\rangle$ there are $p' + 1$ odd indices, namely $\{1, 3, \ldots, 2p' + 1\}$. Similarly, in $\langle b_1\, b_2 \ldots b_q\rangle$ there are $q' + 1$ odd indices, namely $\{1, 3, \ldots, 2q' + 1\}$. Now, since it holds that $(p' + 1) + (q' + 1) \leq \frac{n}{2} + 1$, by Lemma 7 we obtain the propagation $S_{odd} \cup \{a_1, \ldots, a_{2p'+1}, b_1, \ldots, b_{2q'+1},\} \models_{up} d_{p'+q'+2}$.

Using this propagated literal and also $\overline{c}_{2p'+2q'+3}$, and knowing that the clause $\overline{d}_{p'+q'+2} \vee e_{p'+q'+1} \vee c_{2p'+2q'+3}$ belongs to S (because $1 \leq p' + q' + 1 \leq \frac{n}{2}$), we propagate $e_{p'+q+1}$.

Now, let us consider even part of *SMerge*:

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

In $\langle a_2\, a_4 \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_2\, b_4 \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Hence, since $p' < \frac{n}{2}$, $q' < \frac{n}{2}$ and $p' + q' \leq \frac{n}{2}$, by IH we conclude the proof since

$S_{even} \cup \{a_2, \ldots, a_{2p'}, b_2, \ldots, b_{2q'}, \overline{e}_{p'+q'+1}\} \models_{up} \overline{\mathbf{a}}_{\mathbf{2p'+2}}, \overline{\mathbf{b}}_{\mathbf{2q'+2}}$.

CASE 4: p and q are even. (Let $p = 2p'$ and $q = 2q'$).

Let us first focus on the even part of *SMerge*:

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

In $\langle a_2\, a_4 \ldots a_p\rangle$ there are p' even indices, namely $\{2, 4, \ldots, 2p'\}$. Similarly, in the sequence $\langle b_2\, b_4 \ldots b_q\rangle$ there are q' even indices, namely $\{2, 4, \ldots, 2q'\}$. Hence, by Lemma 7 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, 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 the previously propagated literal $e_{p'+q'}$ together with $\overline{c}_{2p'+2q+1}$ propagate $\overline{d}_{p'+q'+1}$.

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

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

In $\langle a_1\, a_2 \ldots a_p\rangle$ there are p' odd indices, namely $\{1, 3, \ldots, 2p' - 1\}$. Similarly, in the sequence $\langle b_1\, b_2 \ldots b_q\rangle$ there are q' odd indices, namely $\{1, 3, \ldots, 2q' - 1\}$. Now, since it holds that $p' + q' \leq \frac{n}{2}$, we can apply IH:

- if $p < n$ and $q < n$, then $p' < \frac{n}{2}$ and $q' < \frac{n}{2}$, and we can conclude that $S_{odd} \cup \{a_1, \ldots, a_{2p'-1}, b_1, \ldots, b_{2q'-1}, \overline{d}_{p'+q'+1}\} \models_{up} \overline{\mathbf{a}}_{\mathbf{2p'+1}}, \overline{\mathbf{b}}_{\mathbf{2q'+1}}$.
- if $p = n$ and $q = 0$, then $p' = \frac{n}{2}$ and $q' = 0$, and we have that
  $S_{odd} \cup \{a_1, \ldots, a_{n-1}, \overline{d}_{\frac{n}{2}+1}\} \models_{up} \overline{\mathbf{b}}_{\mathbf{1}}$
- if $p = 0$ and $q = n$, then $p' = 0$ and $q' = \frac{n}{2}$, and we have that
  $S_{odd} \cup \{b_1, \ldots, b_{n-1}, \overline{d}_{\frac{n}{2}+1}\} \models_{up} \overline{\mathbf{a}}_{\mathbf{1}}$

□

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

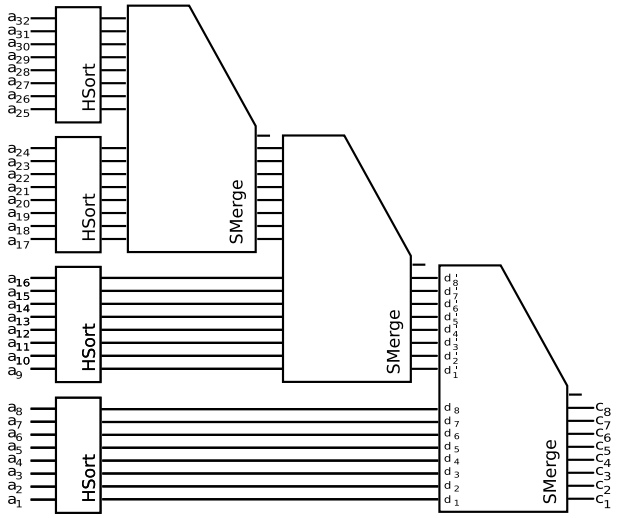

Fig. 2 Representation of $Card(\langle a_1 \dots a_{32} \rangle, 8)$ with output $\langle c_1 \dots c_8 \rangle$.

Proof Let a_m be the number of clauses of $SMerge(A, B)$ if A has length 2^m. It is easy to see that we have the following recurrence:

$$\begin{cases} a_0 = 3 \\ a_m = 2a_{m-1} + 3 \cdot 2^{m-1} \quad \text{for all } m > 0 \end{cases}$$

with solution $a_m = 3 \cdot 2^m + 3m \cdot 2^{m-1}$. Since $m = \log n$ this proves the result about the number of clauses.

Regarding the number of auxiliary variables, if b_m is the number of auxiliary variables for inputs of length 2^m, we have the recurrence:

$$\begin{cases} b_0 = 2 \\ b_m = 2b_{m-1} + 2^m \quad \text{for all } m > 0 \end{cases}$$

with solution $b_m = (m+2) \cdot 2^m$. Replacing m by $\log n$ this gives the desired result. □

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[1].

For sequences of length k, we define:

$$Card(\ \langle a_1\ \dots\ a_k \rangle,\ k\ )\ =\ HSort(\ \langle a_1\ \dots\ a_k \rangle\ )$$

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

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

[1] For k not being a power of two, see Section 5.

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

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

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

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

In Figure 2 one can observe the structure of k-Cardinality Networks.

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.*

Proof Let a_m be the number of clauses if A has length $m \times k$, with $k = 2^r$. It is easy to see that we have the following recurrence:

$$\begin{cases} a_1 = -3 + 3\cdot 2^r - 3r\cdot 2^{r-2} + 3r^2\cdot 2^{r-2} \\ a_m = a_1 + a_{m-1} + (3\cdot 2^r + 3r\cdot 2^{r-1}) \quad \text{for all } m > 1 \end{cases}$$

with solution $a_m = -3m + 6m2^r + 3mr2^{r-2} + 3mr^2 2^{r-2} - 3\cdot 2^r - 3r2^{r-1}$. Since $m = \log n$ and $2^r = k$ this proves the result about the number of clauses, since we obtain $a_m = -3\frac{n}{k} + 6n + \frac{3}{4}n\log k + \frac{3}{4}n\log^2 k - 3k - \frac{3}{2}k\log k$.

Regarding the number of auxiliary variables, if b_m is the number of auxiliary variables for inputs of length $n = m \times k$, with $k = 2^r$, we have the recurrence:

$$\begin{cases} b_1 = (8 - r + r^2)\cdot 2^{r-1} - 2 \\ b_m = b_1 + b_{m-1} + (r+2)\cdot 2^r \ \text{ for all } m > 1 \end{cases}$$

with solution $b_m = m(8 - r + r^2)\cdot 2^{r-1} - 2 + (m-1)(r+2)\cdot 2^r$. Since $m = \log n$ and $2^r = k$ we obtain $b_m = \frac{1}{2}mk(8 - \log k + \log^2 k) - 2 + k(m-1)(\log k + 2) - 2$. □

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 $|A'| = p \le k$, 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:

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

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 using 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* $Card(\langle a_1 \dots a_n\rangle, k) = (\langle c_1 \dots c_k \rangle, S)$ *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 $Card(\langle a_1, \dots a_k \rangle, k) = HSort(\langle a_1, \dots, a_k\rangle)$ and in this case the theorem amounts to Lemma 6.

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

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

If we now consider the sets $A_D = A' \cap \{a_1, \dots, a_k\}$, with size $|A_D| = p_D$, and $A_{D'} = A' \cap \{a_{k+1}, \dots, a_n\}$, with $|A_{D'}| = p_{D'}$, by Lemma 11 we know that $A_D \cup S_D \models_{up} d_1, \dots, d_{p_D}$ and $A_{D'} \cup S_{D'} \models_{up} d'_1, \dots, d'_{p_{D'}}$. 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 $S_M \cup \{d_1, \dots, d_{p_D}, d'_1, \dots, d'_{p_{D'}}, \overline{c}_{p+1}\} \models_{up} \overline{d}_{p_D+1}, \overline{d}'_{p_{D'}+1}$ 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 $a_j \in (\{a_1 \dots a_k\} - A_D)$ and also that $S_{D'} \cup A_{D'} \cup \overline{d}'_{p_{D'}+1} \models_{up} \overline{a}_j$ for all $a_j \in (\{a_{k+1} \dots a_n\} - A_{D'})$, 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 $a_1 + \dots a_n \leq p$. We should first find k, the smallest power of two with $k > p$ and consider the construction $Card(\langle a_1 \dots a_{n+m}\rangle, k) = (\langle c_1, \dots c_k\rangle, S)$. 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 checking the satisfiability of $F \wedge S \wedge \overline{c}_{p+1}$ since, due to Theorem 1, as soon as p variables in $\langle a_1, \dots, a_{n+m}\rangle$ are set to true, the remaining ones will be unit propagated to false, hence disallowing any model not satisfying the cardinality constraint.

Example 2 Let us illustrate how the previous procedure works for $a_1 + a_2 + a_3 + a_4 \leq 1$. We want to remark that there are specific encodings for "at most one" constraints [11], but we use this case here because of its simplicity. First of all we identify $k = 2$ as the smallest power of two greater than the right-hand side of the constraint. Since the number of variables is already multiple of 2 there is no need to add extra input variable and the construction to be built is $Card(\langle a_1, a_2, a_3, a_4\rangle, 2) = (\langle c_1, c_2\rangle, S)$ where we will impose c_2 to be false. As it can be seen in the Figure 3, it consists of two Half Sorting Networks and one Simplified Half Merging Network, which produce the following clauses (in the first row we list the clauses produced by the Half Sorting Networks and in the second row the ones produced by the Simplified Half Merging Network, which consists of two smaller networks and three "linking" clauses):

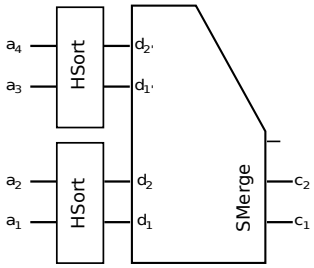

Fig. 3 Representation of $Card(\langle a_1, a_2, a_3, a_4\rangle, 2)$ with output $\langle c_1, c_2\rangle$.

$$\begin{cases} \overline{a}_1 \vee \overline{a}_2 \vee d_2 \\ \overline{a}_1 \vee d_1 \\ \overline{a}_2 \vee d_1 \end{cases} \quad \begin{cases} \overline{a}_3 \vee \overline{a}_4 \vee d'_2 \\ \overline{a}_3 \vee d'_1 \\ \overline{a}_4 \vee d'_1 \end{cases}$$

$$\begin{cases} \overline{d}_1 \vee \overline{d}'_1 \vee s_2 \\ \overline{d}_1 \vee s_1 \\ \overline{d}'_1 \vee s_1 \end{cases} \quad \begin{cases} \overline{d}_2 \vee \overline{d}'_2 \vee s_4 \\ \overline{d}_2 \vee s_3 \\ \overline{d}'_2 \vee s_3 \end{cases} \quad \begin{cases} \overline{s}_2 \vee \overline{s}_3 \vee c_3 \\ \overline{s}_2 \vee c_2 \\ \overline{s}_3 \vee c_2 \end{cases}$$

If, in addition to set c_2 to false, we set a_3 to true, it is not difficult to see that unit propagation sets a_1, a_2 and a_4 to false, as expected. □

• **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 $Card^{\geq}(\langle a_1, \ldots, a_{n+m}\rangle, k) = (\langle c_1, \ldots, c_k\rangle, S)$, identical to $Card(A, k)$, 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* $Card^{\geq}(\langle a_1 \ldots a_n\rangle, k) = (\langle c_1 \ldots c_k\rangle, S)$ *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 $Card^{rng}(\langle a_1, \ldots, a_{n+m}\rangle, k) = (\langle c_1, \ldots, c_k\rangle, S)$, identical to $Card(A, k)$, 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 $Card^{\geq}$). 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* $Card^{rng}(\langle a_1 \ldots a_n\rangle, k) = (\langle c_1 \ldots c_k\rangle, S)$ *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 abov 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 Related Work

Due to the practical importance of cardinality constraints, their encoding into SAT has been subject of thorough study in the last few years. In the following we present the most important contributions, and we describe, for each work, the idea on which the encoding is based, its size and whether it preserves arc consistency or not.

In [15], Warners focused on the more general pseudo-Boolean case, where constraints are of the form $a_1x_1 + \ldots + a_nx_n \leq k$, being the a_i's and the k integer coefficients and the x_i's Boolean variables. The encoding is based on using adders for numbers in binary representation. More concretely, first of all the left hand side of the constraint is split into two halves, each of which is recursively treated to compute the corresponding partial sum. After that, the two partial sums are added and the final result is compared with k. For pseudo-Boolean constraints, the size of the encoding depends on the coefficients a_i, but for cardinality constraints it can be shown to use $O(n)$ clauses and variables, not being able to preserve arc consistency.

Bailleux and Boufkhad, in [2], presented an arc-consistent encoding of cardinality constraints, using $O(n \log n)$ variables and $O(n^2)$ clauses. They describe their encoding as consisting of a totalizer and a comparator. The totalizer can be seen as a binary tree, where the leaves are the x_i's variables. Each intermediate node is labelled with a number s and uses s auxiliary variables to represent, in unary, the sum of the leaves of the corresponding subtree. Given such a node, its two children are labelled with integers $\lfloor s/2 \rfloor$ and $s - \lfloor s/2 \rfloor$ and clauses are used to encode the relation between the

children and parent variables. Since a unary representation of numbers is used, the comparator is trivial and, moreover, it easily allows to encode constraints of the form $k_1 \leq x_1 + \ldots + x_n \leq k_2$, without resorting to two cardinality constraints.

A more applied work is the one of Büttner and Rintanen [14]. The work focuses on planning as a satisfiability problem and the use of cardinality constraints is only a secondary contribution. They suggest two encodings of cardinality constraints. The first one is based on encoding an injective mapping between the true x_i's variables and k elements. It uses $O(nk)$ clauses and variables and is not arc-consistent. The other encoding they present is a small modification of [2]. The idea is simple and similar to the one we have used in our work: there is no need to count up to n, it is enough to count up to $k+1$, which can be used to reduce the number of variables used in each node. Their analysis reveals a use of $O(nk^2)$ clauses, which is an improvement if k is small enough.

In [13], Sinz proposed two different encodings, both based on counters. The first encoding relies on a sequential counter where numbers are represented in unary. It needs $O(nk)$ clauses and variables and is arc-consistent. The second encoding is based on a parallel counter, where numbers are represented in binary. The counter recursively splits the input bits into two halves and counts the number in each half. Results are added using a standard binary adder and a final comparator is used. Is uses $O(n)$ clauses and variables but it is not arc-consistent.

Yet another type of encoding was used in [3], where a BDD-like approach was proposed for pseudo-Boolean constraints. The encoding is arc-consistent and can be exponential in the worst case, but it only uses $O(n^2)$ clauses and variables when applied to cardinality constraints and other particular cases. The idea is simple: given a pseudo-Boolean constraint $a_1x_1 + \ldots + a_nx_n \leq k$, the root of the node is labelled with variable $D_{n,k}$, expressing the sum of the first n terms if no more then k. The two corresponding children are $D_{n-1,k}$ and $D_{n-1,k-a_n}$, indicating the two cases that correspond to setting x_n to false and true, respectively. As expected, the necessary clauses are added to express the relations between the variables and trivial cases are treated accordingly. This construction restricted to cardinality constraints turns out to be polynomial due to the high amount of sharing between nodes, a situation that does not always happen in the pseudo-Boolean case.

Eén and Sörensson [8] presented three encodings for pseudo-Boolean constraints. The first encoding is BDD-based, similar to [3]. It is exponential in the worst case and preserves arc consistency. Their second encoding, based on adder-networks improves the one of [15] in that it uses less adders, but it is still linear and does not preserve arc consistency. Their third encoding is based on sorting networks, and, when restricted to cardinality constraints, preserves arc consistency and requires $O(n \log^2 n)$ clauses and variables.

The work of [4] presents a polynomial and arc-consistent encoding of pseudo-Boolean constraints. When restricted to cardinality constraints it is similar to [2], but the latter is better in terms of size.

Finally, we want to remark the work of [1] on cardinality constraints. The authors revisit the idea of using totalizers, and realize that totalizers require two parameters: the encoding used (unary or binary) and the way the totalizers are grouped (e.g. $(a+b)+(c+d)$ or $(((a+b)+c)+d)$). A thorough experimentation of several possibilities is performed, to which they add two extra aspects: how to order the variables in the totalizers and the use of hybrid encodings, based on using two encodings in parallel

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

Table 1 Comparison of Sorting and Cardinality Networks in terms of size for a constraint of the form $a_1 + \ldots + a_n \leq k$.

hoping that the SAT solver will focus on the most appropriate one for that particular problem.

7 Experimental Evaluation

We start the evaluation of Cardinality Networks focusing on their size. In Table 1 we show, for a constraint $a_1 + \ldots + a_n \leq k$, the number of variables and clauses[2] in Cardinality Networks compared with the Sorting Networks of [5,8] (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.

We now also assess the practical performance of Cardinality Networks, that is, how the encoding affects the performance of SAT solvers. We have compared them with other well-known encodings present in the literature. The encodings we have chosen are the Sorting Networks of [8] (**Sort6** in the tables), Half Sorting Networks (**Sort3**) as introduced in this paper, Cardinality Networks (**Card3**), Cardinality Networks using all 6 clauses (**Card6**), the adder-based encoding (**Adder**) of [8] and the BDD-based encoding (**BDD**) of [3]. We believe these encodings are representative of all different approaches that have been used to tackle cardinality constraints. Other works, like the adder-based encoding of [15], the BDD-based one of [8] or the work by Anbulagan and Grastien [1], are small variations or combinations of the encodings we have chosen.

In addition, we have implemented an SMT-based approach to Cardinality Constraints. Roughly speaking, we have coupled a SAT solver with a theory solver that deals with all cardinality constraints. As soon as some cardinality constraint is violated by the current partial assignment, the SAT solver is forced to backtrack and, as soon as the value of some variable can be propagated due to a constraint, this information is passed to the SAT solver. That is, cardinality constraints are not translated into SAT, but rather dealt with by a dedicated algorithm, very similar in nature to what some pseudo-Boolean solvers do. For more information about SMT, we refer the reader to [12].

The SAT solver we have used is Precosat [6], a very efficient and well-known CDCL (Conflict-Driven Clause Learning) SAT solver. All experiments were conducted on a 2Ghz Linux Quad-Core AMD. We also made some experiments with SLS SAT solvers

[2] Since for every ternary clause there are two binary clauses, the number of literals in the encodings is $\frac{7}{3}$ times the number of clauses.

	Speed-up factor of Card3					Slow-down factor of Card3				
Encoding	**TO**	**4**	**2**	**1.5**	**TOT.**	**1.5**	**2**	**4**	**TO**	**TOT.**
Sort6	29	46	85	160	*320*	24	20	15	12	*71*
Sort3	5	40	38	42	*125*	23	23	3	9	*58*
Card6	26	13	59	78	*176*	19	29	14	17	*79*
Adder	177	50	45	33	*305*	29	11	8	1	*49*
BDD	70	104	57	54	*285*	9	8	5	18	*40*
SMT	168	54	14	7	*243*	7	24	215	12	*258*

Table 2 Comparison in terms of SAT solver runtime of the Cardinality Network encoding with other encodings on the MSU4 suite. Numbers indicate number of families in which Cardinality Networks showed the corresponding speed-up or slow-down factor w.r.t. the encoding indicated in the row.

but their performance was not competitive on the benchmarks we have chosen. This is not a big surprise since the encoding we have presented is designed to work well on unit propagation-based systems. A possible line of research would be to construct SLS-oriented SAT encodings of cardinality constraints.

MSU4 suite. The first set of benchmarks on which we have compared the different encodings comes from the Partial Max-SAT division of the Third Max-SAT evaluation[3]. The benchmarks are encodings of different problems: filter design, logic synthesis, minimum-size test pattern generation, haplotype inference or maximum-quartet consistency. In order to convert them into cardinality constraints we have implemented the *msu4* algorithm [10], which reduces a Max-SAT problem to a series of SAT problems with cardinality constraints. Hence, a single Max-SAT problem is converted into a family of "SAT + cardinality constraints" problems, of which we only kept the ones with non-trivial constraints (that is, that they cannot be converted into a single clause or a set of unit literals). This way, out of the roughly 1600 Max-SAT problems we obtained a set of 14000 benchmarks, each of which contains multiple "less or equal" cardinality constraints.

Results are summarized in Table 2, which presents a comparison of Cardinality Networks with respect to the other 5 different encodings and the SMT approach. All instances coming from the same Partial Max-SAT problem were grouped and their runtimes added. The time limit was set to 600 seconds per benchmark and a single benchmark timing out was enough for considering the whole family to time out. Rows represent different encodings, and columns indicate in how many families Cardinality Networks exhibit the corresponding speed-up or slow-down factor. As an example, the **TO** column in the **Sort6** row indicates that in 29 benchmark families Sorting Networks timed out whereas Cardinality Networks did not. The rightmost 12 in the same row reports 12 families with opposite behavior: Cardinality Network timing out while Sorting Network solving the whole family. The other columns have similar interpretations: column labelled with a speed-up factor of **4** indicates that Cardinality Networks were at least four times as fast, the column **2** that they were between 2 and 4 times as fast and so on. Columns on the right part of the table indicate that Cardinality Networks were four times as slow, between 2 and 4 times as slow, etc. The **TOT.** columns report the total number of benchmarks in which there was some speed-up/slow-down factor. Note that not all 1600 families are present in the table since families in which the

[3] See `http://www.maxsat.udl.cat/08/index.php?disp=submitted-benchmarks`.

two encodings had similar behavior (less than 1.5 speed-up/slow-down factor) or were solved by both encodings in less than a few seconds have not been included.

If we consider the first three encodings of the table, which are all similar in spirit, we can conclude that Cardinality Networks have better behavior overall by observing that most of the families fall on the left-hand side of the table. We can also observe that in this benchmark suite the use of three clauses instead of six has big impact, even stronger than the improvement obtained by using Cardinality Networks instead of Sorting Networks. Regarding the Adder and BDD-based encodings, they both perform poorly, with BDD's being slightly better. Finally, the SMT encoding is much worse if we consider timeouts, but there are 215 benchmarks in which it is at least four times faster than Cardinality Networks (indeed, in about 150 benchmarks it is at least 10 times faster). That seems to indicate that there are some problems in which encoding cardinality constraints into SAT is the best choice, but for some other it is clearly not. Knowing which case applies for each benchmark is a hard and interesting line of research out of the scope of this work.

Discrete-event system diagnosis suite. The second set of benchmarks we have used is the one introduced in [1] [4]. These problems come from discrete-event system (DES) diagnosis. A plant is modeled by a DES, a finite automaton whose transitions are labelled by the events that occur when the transition is triggered. Some events are observable, that is, an observation is emitted when they occur. The goal of the problem is, knowing that there is a set of faulty events in the DES, found a trajectory on the DES consistent with the observations that minimizes the number of faults.

The minimization problem can be solved by imposing that at most k faulty events are used and solving a sequence of problems varying this k. As usual, harder problems take place on the unsatisfiable region, and this is where we generated our cardinality constraint benchmarks. As it happened with the Max-SAT problems, a single DES problem produced a family of "SAT + cardinality constraints" problems. This way, out of the the roughly 600 DES problems, we obtained a set of 6000 benchmarks, each of which contained a single very large "less or equal" cardinality constraint.

Results are summarized in Table 3 where, again, benchmarks have been grouped into families and the same time limit restrictions have been applied. In this case, it is even clearer that Cardinality Networks perform much better. This is probably due to the fact that the cardinality constraints in this suite have lots of variables and quite a small integer, that is, large n and small k, which is precisely where our method outperforms the others. A noteworthy comment is that Half Sorting Networks (i.e. **Sort3**) behave much worse than other encodings, which is not what we expected. After studying the problem in detail, we realized that the preprocessing phase that most state-of-the-art SAT solvers apply turns out to be very costly. This preprocessing phase is built inside Precosat, but some other SAT solver use the external preprocessor Satelite [9]. On those problems, sometimes Satelite consumes all 600 seconds preprocessing, whereas the instance can be solved in less than 5 seconds by an ordinary SAT solver without preprocessing. This indicates that the heuristics that are used to decide which preprocessing techniques to be used are not precise enough for these type of problems.

The last row in Table 3 refers to the best encoding proposed by [1], the paper where the benchmarks were suggested. This encoding is a combination of different techniques

[4] We want to thank Alban Grastien for his assistance with the benchmarks.

	Speed-up factor of Card3					Slow-down factor of Card3				
Encoding	**TO**	**4**	**2**	**1.5**	**TOT.**	**1.5**	**2**	**4**	**TO**	**TOT.**
Sort6	0	216	187	26	*429*	5	0	0	12	*17*
Sort3	166	309	4	5	*484*	0	0	0	0	*0*
Card6	2	0	95	227	*324*	0	0	0	3	*3*
Adder	14	0	23	119	*156*	10	4	0	0	*14*
BDD	97	379	14	0	*490*	0	0	0	0	*0*
SMT	321	53	20	16	*410*	11	4	0	0	*15*
SARA'09	0	0	35	30	*65*	19	26	2	27	*74*

Table 3 Comparison in terms of SAT solver runtime of the Cardinality Network encoding with other encodings on the discrete-event system diagnosis suite. Numbers indicate number of families in which Cardinality Networks showed the corresponding speed-up or slow-down factor w.r.t. the encoding indicated in the row.

in which variables are ordered semantically, that is, using information about their meaning in the original problem. Although our encoding times out more often than theirs, the comparison indicates that Cardinality Networks are competitive, specially if we take into account that only in about 140 out of the 600 families there was a significant difference between the two encodings. We see these results as a strong point in favor of Cardinality Networks since we are comparing with an encoding that was "tuned" to obtain the best behavior on these problems. Our encoding, on the other side, is general and not designed toward this concrete type of problems.

Tomography suite. The last set of benchmarks we have used is the one introduced in [2] [5]. The idea is to first generate an $N \times N$ grid in which some cells are filled and some others are not. These problem consists in finding out which are the filled cells using only the information of how many filled cells there are in each row, column and diagonal. For that purpose, variables x_{ij} are used to indicate whether cell (i, j) is filled and several "exactly" cardinality constraints are used to impose how many filled cells there are in each row, column and diagonal.

For each size $N \times N$, we generated 100 random grids and ran all encodings and the SMT approach on them, computing the average time. Note that in "exactly" constraints one is forced to add all 6 clauses in both Sorting and Cardinality Networks, and hence the 3-clause versions do not appear in the experiments. Again, a time limit of 600 seconds per benchmark was used.

Results are summarized in Table 4. The first obvious conclusion we can draw is that BDDs are by far the best encoding. The SMT approach is also very promising but does not scale so well. Regarding the other encodings, adders scale worse than other encodings and results are inconclusive as to whether Sorting or Cardinality Networks perform better.

All in all, the results in the different suites indicate that Cardinality Networks improve upon previously known encodings. They performance is very good and, more importantly, it is by far the most robust one. As expected, on some particular problems there are encodings that behave better than Cardinality Networks and hence, for practical purposes it is important to use libraries that allow one to choose among different encoding possibilities.

[5] We want to thank Yacine Boufkhad for kindly providing us with the benchmark generator.

Size of grid	Sort6	Card6	Adder	BDD	SMT
15x15	0.49	0.3	0.2	0.16	0.09
16x16	0.57	0.35	0.1	0.21	0.02
17x17	0.47	0.69	0.33	0.24	0.02
18x18	2.17	2.11	0.67	0.28	0.26
19x19	3.26	2.4	3.75	0.33	0.37
20x20	1.79	1.37	0.81	0.36	0.03
21x21	1.52	2.13	2.55	0.42	0.66
22x22	14.69	8.06	5.56	0.49	0.61
23x23	6.35	3.36	8.85	0.57	0.84
24x24	16.81	4.2	24.06	0.63	1.21
25x25	6.7	12.43	32.4	0.73	1.33
26x26	11.64	6.23	28.67	0.83	1.45
27x27	13.3	9.87	35.92	0.91	2.50
28x28	8.64	33.32	59.3	1	2.52
29x29	20.09	16.37	79.77	1.1	5.36
30x30	34.41	64.26	TO	1.18	14.72

Table 4 Comparison in terms of SAT solver runtime of various encodings on the tomography suite. Numbers indicate average time in seconds.

8 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.

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

1. Anbulagan and A. Grastien. Importance of Variables Semantic in CNF Encoding of Cardinality Constraints. In V. Bulitko and J. C. Beck, editors, *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA'09*. AAAI, 2009.
2. O. Bailleux and Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In F. Rossi, editor, *Principles and Practice of Constraint Programming, 9th International Conference, CP'03*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.
3. O. Bailleux, Y. Boufkhad, and O. Roussel. A Translation of Pseudo Boolean Constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 2(1-4):191–200, 2006.
4. O. Bailleux, Y. Boufkhad, and O. Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In O. Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing, SAT'09"*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2009.
5. K. E. Batcher. Sorting Networks and their Applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.
6. A. Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2010. Technical Report 10/1, August 2010, FMV Reports Series.

7. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM, CACM*, 5(7):394–397, 1962.
8. N . Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
9. N. Eén and A. Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In F. Bacchus and T. Walsh, editors, *8th International Conference on Theory and Applications of Satisfiability Testing, SAT'05*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.
10. J. Marques-Silva and J. Planes. Algorithms for Maximum Satisfiability using Unsatisfiable Cores. In *2008 Conference on Design, Automation and Test in Europe Conference, DATE'08*, pages 408–413. IEEE Computer Society, 2008.
11. J. P. Marques-Silva and I. Lynce. Towards robust cnf encodings of cardinality constraints. In C. Bessiere, editor, *Principles and Practice of Constraint Programming, 13th International Conference, CP'07*, volume 4741 of *Lecture Notes in Computer Science*, pages 483–497. Springer, 2007.
12. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM, JACM*, 53(6):937–977, 2006.
13. C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In P. v. Beek, editor, *Principles and Practice of Constraint Programming, 11th International Conference, CP'05*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.
14. M. Büttner and J. Rintanen. Satisfiability planning with constraints on the number of actions. In S. Biundo, K. L. Myers, and K. Rajan, editors, *15th International Conference on Automated Planning and Scheduling, ICAPS'05*, pages 292–299. AAAI, 2005.
15. J. P. Warners. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Information Processing Letters*, 68(2):63–69, 1998.