Towards an Optimal CNF Encoding of Boolean Cardinality Constraints
(Technical Report)

Carsten Sinz

Symbolic Computation Group, WSI for Computer Science
University of Tübingen, 72076 Tübingen, Germany
sinz@informatik.uni-tuebingen.de

Abstract. We consider the problem of encoding Boolean cardinality constraints in conjunctive normal form (CNF). Boolean cardinality constraints are formulae expressing that at most (resp. at least) k out of n propositional variables or formulae are true. We present a unifying framework for a whole family of such encodings encompassing previously proposed solutions. We give two novel encodings that improve upon existing results, one which requires only $7n$ clauses and $2n$ additional variables, and another one demanding $\mathcal{O}(n \cdot k)$ clauses, but with the advantage that inconsistencies can be detected in linear time by unit propagation alone. Moreover, we prove a linear lower bound on the number of required clauses for any such encoding.

1 Introduction

Cardinality constraints—expressing numerical bounds on discrete quantities—arise frequently out of the encoding of real-world problems: in product configuration, e.g., engineers want to express that at least n parts of a certain kind are needed for a fully functional product [1, 2]; in scheduling radio frequencies transmitters have to obey certain non-interference distance rules [3]; or in reconstructing images form computer tomographs different projections impose different numerical constraints on the original picture [4]. Moreover, the general problem of re-encoding problems of a finite variable logic in ordinary propositional logic usually requires setting up cardinality constraints expressing that each variable has to take exactly one of its possible values at any time.

With considerable progress made over the last years in solving propositional satisfiability (SAT) instances, interest increased in tackling problems that include cardinality constraints using SAT-solvers. This requires, however, a re-formulation of cardinality constraints in the language of purely propositional logic, or, even more restrictive, in conjunctive normal form (CNF), the predominant input language of modern SAT-solvers. The SAT-approach also offers the advantage that problem instances consisting of a mixture of ordinary Boolean constraints and cardinality constraints (occurring frequently in practical applications like product configuration), can be handled with only one unified technique, thus making the use of hybrid solvers dispensable.

Boolean cardinality constraints are a special—but common—form of cardinality constraints. They put numerical restrictions on the number of propositional variables

that are allowed to be true at the same time. A typical construct like $\leq k\,(x_1, \ldots, x_n)$ means that not more than k out of the n variables $x_1, \ldots, x_n$ are allowed to be true. The traditional way of converting a constraint like $\leq k\,(x_1, \ldots, x_n)$ to purely propositional logic is by explicitly excluding all possible combinations of $k+1$ variables being true. So, exemplarily for the case $k = 1$, we have to exclude all combinations of two variables being simultaneously true, thus obtaining

$$\bigwedge_{1 \leq i < j \leq n} (\neg x_i \vee \neg x_j) \ .$$

This formula, however, consists of $\binom{n}{2}$, i.e. $O(n^2)$, clauses. In general, using the given translation technique, we obtain the formula

$$\bigwedge_{\substack{M \subseteq \{1,\ldots,n\} \\ |M| = k+1}} \ \bigvee_{i \in M} \neg x_i$$

as an encoding of $\leq k\,(x_1, \ldots, x_n)$. Unfortunately, this encoding requires $\binom{n}{k+1}$ clauses of length $k+1$, which in the worst case of $k = \lceil n/2 \rceil - 1$ amounts to $O(2^n/\sqrt{n/2})$ clauses.[1]

Better encodings are known. Bailleux and Boufkhad [4], e.g., presented an encoding requiring only a quadratic number of clauses. Another translation, given by Warners [6] for the more general case of discrete linear inequalities with integer coefficients, results in an encoding with $8n$ clauses and $2n$ additional variables.

General Approach: Counting and Comparing. In this paper we identify a general idea that seems to be common to all efficient encodings of cardinality constraints—although they are often not presented in this form by their authors: first, a (hardware) circuit is built that computes whether or not the cardinality constraint is fulfilled. Then, this hardware is encoded as efficiently as possible by a formula in clausal normal form.

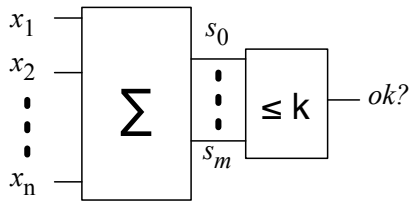

Fig. 1. General approach to compute $\leq k\,(x_1, \ldots, x_n)$: First, sum up the input bits x_i with a unary or binary counter; then, compare the resulting sum with the bound k.

[1] The complexity bound is by Stirling's approximation of $n!$. A more elaborate result can be found in [5].

Obviously, there are different possibilities on how to construct such a circuit, but in general they consist of two subsequent stages: in the first stage, the *counter stage*, the number of inputs that are set to true is counted. Then, in a subsequent *comparator stage*, it is checked whether the counter result lies within the bound given by the cardinality constraint. The general approach is displayed in Fig. 1 for the cardinality constraint $\leq k\,(x_1, \ldots, x_n)$. Two criteria allow a coarse distinction of different counter designs: (i) Does the circuit count in unary or in binary, i.e., is the counter result encoded as a unary or binary number?[2] And (ii), is the counter realized as a sequential or as a parallel implementation?

In the literature we have found two counter-based realizations of CNF encodings: Bailleux and Boufkhad's solution [4] makes use of a parallel, unary counter; the approach by Warners [6] resembles a parallel, binary counter, but with some differences in the encoding and without explicit reference to a hardware design.

The rest of this paper is organized as follows: In the next section we give basic definitions. Then we present two circuit designs resulting in novel CNF encodings: one based on a sequential, unary counter, the other based on a parallel, binary counter. Thereafter, we prove a linear lower bound on the number of clauses for any CNF encoding. Finally, we point to possible future directions of research and conclude.

2 Basic Definitions

We consider the general problem of encoding a full propositional language, extended by cardinality constraints, in conjunctive normal form (CNF)—a problem that frequently occurs with the encoding of real-world problems as outlined in the introduction. For a purely propositional conversion procedure (i.e. with no cardinality constraints present) we refer the reader to the literature [7, 8], and concentrate on the conversion of cardinality constraints themselves instead.

As a starting point for our encoding, we assume a propositional language where formulae $(\phi, \psi, \ldots)$ are built from a set of variables $V := \{x_i \mid 1 \leq i \leq i_{max}\}$ by using the following generation rule:

$$\phi, \psi \ ::= \ \bot \mid \top \mid x_i \mid \neg\phi \mid (\phi \vee \psi) \mid (\phi \wedge \psi) \mid (\phi \Rightarrow \psi) \mid (\phi \Leftrightarrow \psi) \mid (\phi \oplus \psi) \ .$$

Here, $\bot$ denotes falsity, $\top$ denotes truth, $\oplus$ stands for exclusive OR, the other symbols have the usual meaning. The set of all formulae built upon V we will denote by $F(V)$ or simply F. We further assume the customary definition of the semantics of formulae using variable assignments $\alpha : V \to \mathbb{B}$ and corresponding evaluation functions $\hat{\alpha} : F(V) \to \mathbb{B}$ for formulae, where $\mathbb{B} = \{0, 1\}$ denotes the set of Boolean constants.

We extend our language to additionally include *cardinality constaints*, i.e. expressions of the form $\leq k\,(\phi_1, \ldots, \phi_n)$, $\geq k\,(\phi_1, \ldots, \phi_n)$, and $= k\,(\phi_1, \ldots, \phi_n)$, with $k \in \mathbb{Z}$, $n \geq 0$, and $\phi_i \in F$ for $1 \leq i \leq n$. The intended meaning is that at most (at least, exactly) k among the formulae ϕ_i $(1 \leq i \leq n)$ are true. A formal definition of their semantics is given by the following definition.

[2] In a unary number representation, a value n between 0 and k is represented as a vector of k bits $(n_k n_{k-1} \ldots n_1)$, with the first n bits set to true and the others to false.

Definition 1. *For a variable assignment $\alpha : V \to \mathbb{B}$, the truth value of a cardinality constraint is determined as follows:*

$$\begin{aligned}
\hat{\alpha}(\leq k\,(\phi_1,\ldots,\phi_n)) = 1 \quad &\textit{iff} \quad |\{\phi_i \mid \hat{\alpha}(\phi_i) = 1\}| \leq k \\
\hat{\alpha}(\geq k\,(\phi_1,\ldots,\phi_n)) = 1 \quad &\textit{iff} \quad |\{\phi_i \mid \hat{\alpha}(\phi_i) = 1\}| \geq k \\
\hat{\alpha}(= k\,(\phi_1,\ldots,\phi_n)) = 1 \quad &\textit{iff} \quad |\{\phi_i \mid \hat{\alpha}(\phi_i) = 1\}| = k
\end{aligned}$$

We can separate the problem of CNF conversion for cardinality constraints and that of CNF conversion for the ordinary propositional logical operators by the following lemma.

Lemma 1. *We denote by $\phi \overset{SAT}{\equiv} \psi$ that ϕ and ψ are satisfiability equivalent, i.e. that ϕ is satisfiable iff ψ is. Then the following holds:*

$$\begin{aligned}
\leq k\,(\phi_1,\ldots,\phi_n) \quad &\overset{SAT}{\equiv} \quad \leq k\,(y_1,\ldots,y_n) \wedge \bigwedge_{1\leq i\leq n} (\phi_i \Rightarrow y_i) \\
\geq k\,(\phi_1,\ldots,\phi_n) \quad &\overset{SAT}{\equiv} \quad \geq k\,(y_1,\ldots,y_n) \wedge \bigwedge_{1\leq i\leq n} (y_i \Rightarrow \phi_i)\ ,
\end{aligned}$$

where the y_i's are new propositional variables not occurring in any of the ϕ_i's.

Proof. With equivalences instead of implications (on the right), the given formulae are obvious. By the usual polarity argument [8] of CNF conversion, the respective directions of the equivalences can be dropped.

The following lemma summarizes some simple properties of cardinality constraints.

Lemma 2. *For any $k \in \mathbb{Z}, n \in \mathbb{N}$ and propositional variables x_i $(1 \leq i \leq n)$ the following properties hold:*

1. $\leq k\,(x_1,\ldots,x_n)$ *iff* $\geq (n-k)\,(\neg x_1,\ldots,\neg x_n)$.
2. $\neg \leq k\,(x_1,\ldots,x_n)$ *iff* $\geq (k+1)\,(x_1,\ldots,x_n)$.
3. $\leq k\,(x_1,\ldots,x_n)$ *is always true for $k \geq n$ and always false for $k < 0$.*
4. $\geq k\,(x_1,\ldots,x_n)$ *is always true for $k \leq 0$ and always false for $k > n$.*
5. $= k\,(x_1,\ldots,x_n)$ *iff* $\leq k\,(x_1,\ldots,x_n) \wedge\ \geq k\,(x_1,\ldots,x_n)$.

Proof. Trivial.

By Lemma 2, we can restrict our attention to cardinality constraints of the form $\leq k\,(x_1,\ldots,x_n)$, i.e. we need not consider the opposite form $\geq k\,(x_1,\ldots,x_n)$. Alternatively, we could have used both forms and restrict k to the interval $[0, \lfloor n/2 \rfloor]$. Moreover, conversion to negation normal form (NNF) for formulae containing cardinality constraints can now be accomplished by Lemma 2(2).

We now can define what we mean by a clausal encoding:

Definition 2. *A clause set E over the variables $V = \{x_1,\ldots,x_n,s_1,\ldots,s_m\}$ is a* clausal encoding of $\leq k\,(x_1,\ldots,x_n)$ *if for all assignments $\alpha : \{x_1,\ldots,x_n\} \to \mathbb{B}$ the following holds: there is an extension of α to $\alpha^* : V \to \mathbb{B}$ that is a model of E if and*

only if α is a model of $\leq k\,(x_1, \ldots, x_n)$, i.e. if and only if at most k of the variables x_i are set to 1 by α.[3]

Thus, a clausal encoding is a representation of $\leq k\,(x_1, \ldots, x_n)$ that preserves satisfiability for all assignments to the variables x_i. It may employ auxiliary variables $\{s_1, \ldots, s_m\}$ to encode the cardinality constraint.

To compare the efficiency (according to Bailleux and Boufkhad [4]) of different encodings we use the following definition, which defines the complexity of evaluating a clausal encoding of cardinality constraints, i.e. how much time it takes to find out whether or not a cardinality constraint is fulfilled, given an assignment to all variables x_i of the cardinality constraint.

Definition 3. *A clausal encoding E of $\leq k\ (x_1, \ldots, x_n)$ over the variables $V = \{x_1, \ldots, x_n, s_1, \ldots, s_m\}$ is called* decidable in time $t(n)$ *if the following holds: for each (partial) assignment $\alpha : \{x_1, \ldots, x_n\} \to \mathbb{B}$ it can be decided in time $t(n)$ whether or not α possesses an extension $\alpha^* : V \to \mathbb{B}$ that is a model of E.*

3 Encoding Using a Sequential Counter

We now give a CNF encoding for cardinality constraints of the form $\leq k\,(x_1, \ldots, x_n)$ that is based on a sequential counter circuit. The circuit is shown in Fig. 2 and computes partial sums $s_i = \sum_{j=1}^{i} x_j$ for increasing values of i upto the final $i = n$. The values of all s_i's are represented as unary numbers. The overflow bits v_i are set to true if the partial sum s_i is greater than k.

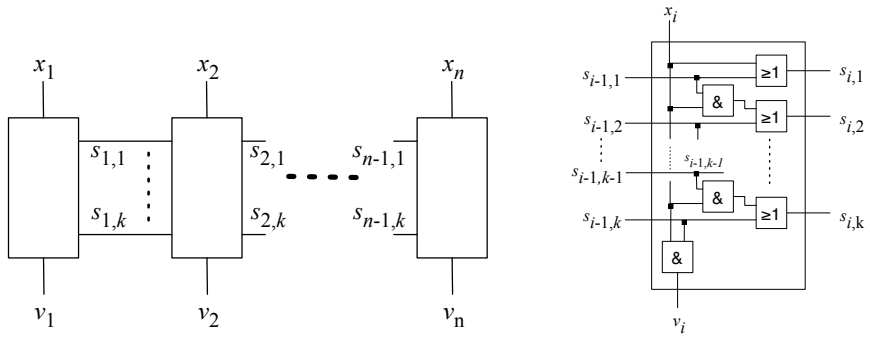

Fig. 2. Left: Circuit for computing $\leq k\ (x_1, \ldots, x_n)$. $s_{i,j}$ denotes the j-th digit of the i-th partial sum s_i in unary representation; variables v_i are overflow bits, indicating that the i-th partial sum is greater than k. **Right:** Sub-circuit for computing a partial sum s_i in unary representation.

[3] A *model* for a formula ϕ is the same as a satisfying assignment of ϕ, i.e. an assignment to the variables of ϕ under which ϕ evaluates to true.

In the following, to avoid special treatment, we tacitly skip the cases $k = 0$ as well as $n > 1$ and always assume $k > 0$ and $n > 1$.[4]

The first partial sum s_1 of the sequential counter is easy to compute, it is just equivalent to x_1, thus $s_{1,1} \Leftrightarrow x_1$ and all other digits of s_1 are zero. We therefore obtain $s_{1,j} \Leftrightarrow \bot$, or simply $\neg s_{1,j}$, for $1 < j \leq k$. Moreover, overflow bit v_1 is never set (as $k > 0$), and therefore $v_1 \Leftrightarrow \bot$. At the other end, the last (n-th) partial sum does not have to be computed, we only need the overflow bit, for which $v_n \Leftrightarrow x_n \wedge s_{n-1,k}$ holds. For all intermediate values of i, i.e. $i \in \{2, \ldots, n-1\}$, we have to add x_i to the already computed partial sum s_{i-1} to obtain s_i. As the partial sums are given in unary representation, we just have to set the lowest digit that is still zero in s_{i-1} to one in s_i if x_i is true. Otherwise, i.e. if x_i is false, $s_i = s_{i-1}$. In Fig. 2, on the right hand side, we have depicted the sub-circuit that accomplishes this task. It is easily seen that for $x_i = 0$ the input bits $s_{i-1,j}$ are just copied to the next partial sum s_i. Otherwise the lowest 0-digit is switched to 1 in s_i. The sub-circuit to compute s_i is characterized by the following set of equivalences:

$$\begin{aligned} s_{i,1} &\Leftrightarrow x_i \vee s_{i-1,1} \\ s_{i,j} &\Leftrightarrow x_i \wedge s_{i-1,j-1} \vee s_{i-1,j} \quad \text{for } 1 < j \leq k \\ v_i &\Leftrightarrow x_i \wedge s_{i-1,k} \end{aligned}$$

Now, for the cardinality constraint to be fulfilled, no overflows are allowed, and thus all v_i have to be false. This property conceptionally already belongs to the comparator part of the whole circuit computing the cardinality constraint. We can use the fact that $\neg v_i$ for $1 \leq i \leq n$ has to hold to simplify the equivalences defining the counter circuit.

We can further simplify the set of equivalences, by applying a common technique to compute compact CNF representations: from the equivalences defining the $s_{i,j}$ only one direction is of relevance, the other can be dropped due to polarity considerations (we just need the direction from "right" to "left"). This technique, that preserves satisfiability of the whole set of equivalences, was introduced by Tseitin [9] and was later re-invented and extended by different authors [10, 11, 7, 8]. We refer the reader interested in the details of this technique to the literature.

We thus arrive at a set of clauses, call it $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$, defining the cardinality constraint $\leq k\,(x_1, \ldots, x_n)$ based on the sequential counter with the additional property that no

[4] The cardinality constraint for the case $k = 0$ is easily seen to be equivalent to $\neg x_1 \wedge \cdots \wedge \neg x_n$, which is already in CNF. For $n = 0$, the constraint is always true. For $n = 1$ it is true if $k > 0$, and otherwise equivalent to $(\neg x_1)$.

overflows must occur:

$$
\begin{array}{l}
(\neg x_1 \vee s_{1,1}) \\
(\neg s_{1,j}) \qquad \text{for } 1 < j \leq k
\end{array}
$$

$$
\left.
\begin{array}{l}
(\neg x_i \vee s_{i,1}) \\
(\neg s_{i-1,1} \vee s_{i,1}) \\
\left.
\begin{array}{l}
(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \\
(\neg s_{i-1,j} \vee s_{i,j}) \\
(\neg x_i \vee \neg s_{i-1,k})
\end{array}
\right\} \quad \text{for } 1 < j \leq k
\end{array}
\right\} \text{for } 1 < i < n
$$

$$
(\neg x_n \vee \neg s_{n-1,k})
$$

The first two lines represent the case $i = 1$, the last line the case $i = n$. Counting the number of clauses n_{CL} in this set, we arrive at $k + 1$ clauses altogether for the cases $i = 0$ and $i = n$, and $(n-2) \cdot (2(k-1)+3)$ clauses for the remaining cases, summing up to $n_{\mathrm{CL}} = 2nk + n - 3k - 1$ clauses. Moreover, apart from the essential x_i's, the clause set uses $n_{\mathrm{V}} = (n-1) \cdot k$ auxiliary variables.

Due to its practical importance, we give in brief the clause set $\mathrm{LT}^{n,1}_{\mathrm{SEQ}}$ for the case $k = 1$ explicitly (as a formula):

$$
(\neg x_1 \vee s_{1,1}) \wedge (\neg x_n \vee \neg s_{n-1,1}) \wedge \bigwedge_{1<i<n} \Big((\neg x_i \vee s_{i,1}) \wedge (\neg s_{i-1,1} \vee s_{i,1}) \wedge (\neg x_i \vee \neg s_{i-1,1}) \Big)
$$

This clause set consists of $3n - 4$ clauses (and $n - 1$ additional encoding variables) and is thus—with regard to the number of clauses—superior to the naïve encoding for all $n > 5$. The following theorem summarizes our results.

Theorem 1. $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ *is a clausal encoding of* $\leq k\,(x_1, \ldots, x_n)$ *requiring* $\mathcal{O}(n \cdot k)$ *clauses and* $\mathcal{O}(n \cdot k)$ *auxiliary variables.*

Proof. We just give an outline of the proof that $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ is a clausal encoding of $\leq k\,(x_1, \ldots, x_n)$. By induction we show two parts: first, that setting $k+1$ variables of the x_i's to true leads to a contradiction and, second, that by setting exactly k of the variables x_i to true all auxiliary encoding variables $s_{i,j}$ get fixed values. From the second part it follows that fixing k variables to true does not lead to a contradiction, and thus setting less than k variables does so, too. The upper bounds on the number of clauses and variables can be read off directly from the encoding given in this section.

All details of the proof can be found on the Internet under http://www-sr.informatik.uni-tuebingen.de/~sinz/CardConstraints.

The encoding $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ also fulfills the *efficiency condition* given by Bailleux and Boufkhad [4]. If more than k variables are set to true (which violates the cardinality constraint $\leq k\,(x_1, \ldots, x_n)$), this can be detected by unit propagation alone, i.e. by a linear time decision procedure. This can be seen immediately from the fact that $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ consists solely of Horn clauses, and sets of Horn clauses are well known to be decidable by unit propagation. Therefore $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ is decidable in linear time. Another requirement

of efficiency as defined by Bailleux and Boufkhad is that, for a partial assignment that sets k of the variables x_i to true, the value of all other x_i's must be derivable by unit propagation. That $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ fulfills this condition, too, can be seen by the argument given in the proof of Theorem 1: if k of the variables x_i are set to true, all auxiliary variables $s_{i,j}$ have fixed values, and thus the remaining x_i's are forced to be set to false.

We have observed that many cardinality constraints occurring in practice require only small values of k (compared to n). Take the following as a typical representative for such a constraint: in product configuration products are typically equipped with a set of Boolean variables representing the denomination country of the product. Out of those, exactly one variable has to be selected, making up a cardinality constraint with $k = 1$. Thus, $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ can deliver compact encodings for many cases of practical relevance.

Besides, note that the clause set $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ can be further simplified by propagating and thus removing the unit clauses $(\neg s_{1,j})$ for $1 < j \leq k$. This would result in a further reduction in the number of clauses and variables that we do not want to elaborate on here.

4 Encoding Using a Parallel Counter

The second encoding we present is based on a parallel counter circuit designed by Muller and Preparata [12]. Their counter (shown in Fig. 3) recursively splits the input bits x_i into two halves, and counts the number of inputs that are set to true in each half. The results—represented as binary numbers—are then added using a standard m-bit binary adder. In order to obtain a circuit for cardinality constraints based on this counter, the output bits of the counter are handed on to a subsequent comparator which checks whether or not the counter value is less than k. (The comparator is not shown in Fig. 3.)

4.1 Parallel Counter Circuit

The input to the parallel counter consists of n variables $x_1, \ldots, x_n$ that are to be counted. The input variables are divided into two halves of preferably equal size that are processed by two independent sub-counters in parallel. The first half is always assumed to be at least as large as the second one and to possess $2^m - 1$ inputs (for some integer m). Thus the result of the counter for the first half can always be represented as an m-bit binary number. The second half is handled by another counter that is responsible for the remaining inputs (except for one, which is treated separately). These conditions can be met by choosing m in such a way that $2^m \leq n < 2^{m+1}$ holds, i.e. $m = \lfloor \log n \rfloor$.[5] Note that by choosing m that way, it may happen that the second half has to process less inputs than the first one, including the border case where the second half has no inputs at all.

The results of each half, represented as binary numbers $y_{m-1} \ldots y_0$ and $z_{m-1} \ldots z_0$ with the usual bit order (i.e. y_0 and z_0 are the least significant bits), are then added by an

[5] By 'log' we denote the *logarithmus dualis*, i.e. the logarithm to the base 2.

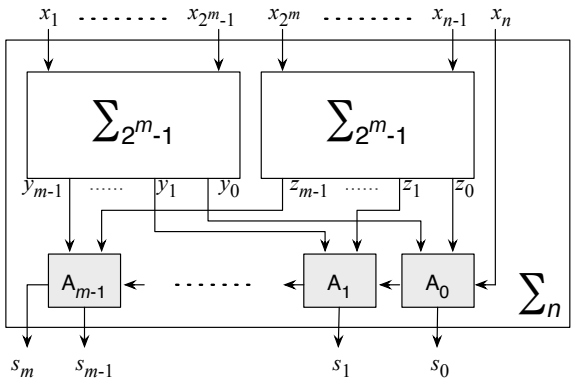

Fig. 3. Parallel counter according to Muller and Preparata [12] for recursively computing the number of inputs x_i that are set to true (represented as a binary number). The sub-circuits A_j are 1-bit-(full-)adders.

ordinary m-bit adder. This m-bit adder is built from m 1-bit adders. In the case shown in Fig. 3, where the second half also consists of $2^m - 1$ inputs, all 1-bit-adders have to be full-adders. In other cases, where n is not of the form $2^{m+1} - 1$, some of them may be replaced by half-adders (with only two inputs), as some of the counter's inputs must be 0. More exactly, this happens for each 0 in the binary representation of n.

Before we give the defining equations for the basic constituents of the parallel counter—the 1-bit (full and half) adders—we want to derive upper bounds on the number of those basic building blocks that are needed for the complete circuit. To estimate the number of full-adders, Muller and Preparata observe that each full-adder has one output less than it has inputs, and thus reduces the number of "wires" by one (the half-adders do not change this number). As we start with n inputs and end with $m + 1$ outputs, we know that our circuit consists of $n - (m + 1)$ full adders. As noted above, the number of half adders equals the number of zeros in the binary representation of n and therefore must be less than or equal to m. Thus our parallel counter possesses exactly $n - \lfloor \log n \rfloor - 1$ full-adders and at most $\lfloor \log n \rfloor$ half-adders.

Each half-adder is determined by the following set of equivalences

$$\begin{aligned} s_{\text{out}} &\Leftrightarrow a \oplus b \\ c_{\text{out}} &\Leftrightarrow a \wedge b \ , \end{aligned}$$

where we assume that the half-adder computes $a + b$ and that s_{out} and c_{out} are the sum resp. carry output bit of the half-adder. The equivalences describing the full-adder, computing $a + b + c$, are:

$$\begin{aligned} s_{\text{out}} &\Leftrightarrow a \oplus b \oplus c \\ c_{\text{out}} &\Leftrightarrow a \wedge b \vee c \wedge (a \oplus b) \ , \end{aligned}$$

again with the same meaning of the output bits s_{out} and c_{out}. Dropping once more one direction of the equivalences, and converting the remaining implications (form "right"

to "left") to CNF, yields three clauses

$$\begin{array}{ll} (a \vee \neg b \vee s_{\text{out}}) & (\neg a \vee \neg b \vee c_{\text{out}}) \\ (\neg a \vee b \vee s_{\text{out}}) & \end{array}$$

for each half-adder and seven clauses

$$\begin{array}{ll} (a \vee b \vee \neg c \vee s_{\text{out}}) & (\neg a \vee \neg b \vee c_{\text{out}}) \\ (a \vee \neg b \vee c \vee s_{\text{out}}) & (\neg a \vee \neg c \vee c_{\text{out}}) \\ (\neg a \vee b \vee c \vee s_{\text{out}}) & (\neg b \vee \neg c \vee c_{\text{out}}) \\ (\neg a \vee \neg b \vee \neg c \vee s_{\text{out}}) & \end{array}$$

for each full-adder. Combining this with the result of the last paragraph, we obtain

$$7 \cdot (n - \lfloor \log n \rfloor - 1) + 3 \cdot (\lfloor \log n \rfloor) = 7n - 4\lfloor \log n \rfloor - 7$$

as an upper bound on the number of clauses required for the parallel binary counter.

We may further note that the additionally required variables for the parallel counter are exactly the sum and overflow bits of the half- and full-adders. We therefore need at most $2 \cdot (n-1)$ additional variables.

4.2 Comparator Circuit

We still need a comparator circuit as a companion to the parallel counter of the last section in order to encode a cardinality constraint $\leq k\,(x_1, \ldots, x_n)$. This comparator circuit has to make sure that the result $s_m s_{m-1} \ldots s_0$ of the binary counter is not greater than k. For building this binary comparator we assume that the constraint's limit k is given as an $(m+1)$-bit binary number, say $k = k_m \ldots k_0$. We can then easily give recursive equations to generate the clauses for the $(m+1)$-bit comparator:

$$L(k_0) = \begin{cases} \{\{\neg s_0\}\} & \text{if } k_0 = 0 \\ \emptyset & \text{if } k_0 = 1 \end{cases}$$

$$L(k_i \ldots k_0) = \begin{cases} \{\{\neg s_i\}\} \cup L(k_{i-1} \ldots k_0) & \text{if } k_i = 0 \\ \{\{\neg s_i\}\} \otimes L(k_{i-1} \ldots k_0) & \text{if } k_i = 1 \end{cases}$$

Here $\otimes$ denotes clause distribution, i.e. $A \otimes B = \{x \cup y \mid x \in A, y \in B\}$ for sets of clauses A, B. Clause distribution corresponds to computing the disjunction of two clause sets by applying the distributivity law. With this definition, $L(k_m \ldots k_0)$ then is the clause set that ensures that the counter's output is less than k. The intention behind the inductive definition is this: for an i-bit comparator, if the highest, i.e. the i-th, bit of k is zero, then the counter bit s_i must also be zero (otherwise the counter value would be greater). If the highest bit is 1, then the lower $(i-1)$ bit-value of the counter must be smaller than the lower $(i-1)$ bit-value of k, but only if s_i is set.

We want to demonstrate the functioning of the comparator by an example. Let $k = 20 = (10100)_2$. Then we obtain the following clause set expressing that the 5-

bit counter value $s_4 \dots s_0$ is less than 20:

$$\begin{aligned} L(k_0) &= \{\{\neg s_0\}\} \\ L(k_1 k_0) &= \{\{\neg s_1\}, \{\neg s_0\}\} \\ &\vdots \\ L(k_4 k_3 k_2 k_1 k_0) &= \{\{\neg s_4, \neg s_3\}, \{\neg s_4, \neg s_2, \neg s_1\}, \{\neg s_4, \neg s_2, \neg s_0\}\} \end{aligned}$$

It is easily seen that the $(m+1)$-bit comparator generates as many clauses as there are zeros in the $(m+1)$-bit binary representation of k. As we may assume k to be not greater than n (see Lemma 2), we obtain a $\lfloor \log n \rfloor$ upper bound on the number of clauses required for the comparator.

Combining the results of this paragraph with that of the last one and doing the same for the clause sets of the parallel counter and the comparator (resulting in the clause set $\mathrm{LT}^{n,k}_{\mathrm{PAR}}$), we arrive at a theorem characterizing the encoding of cardinality constraints with a parallel binary counter:

Theorem 2. *$\mathrm{LT}^{n,k}_{\mathrm{PAR}}$ is a clausal encoding of $\leq k\,(x_1, \dots, x_n)$ requiring at most $7n - 3\lfloor \log n \rfloor - 6$ clauses and $2n - 2$ auxiliary variables.*

Proof. Along the lines of the proof for Theorem 1. For details see http://www-sr.informatik.uni-tuebingen.de/~sinz/CardConstraints.

5 Comparison

In this section we want to compare the two encodings given in this paper with other encodings that can be found in the literature. Criteria for assessing the different encodings are (i) the number of clauses required for the encoding; (ii) the number of additional propositional variables required for the encoding; and (iii) the time needed to decide the encoding (as defined in Def. 3). Besides our two encodings, we have included the naïve encoding (given in the introduction) and the encodings of Bailleux&Boufkhad and Warners into our comparison.

Table 1. Comparison of different encodings for $\leq k\,(x_1, \dots, x_n)$.

Encoding	#clauses	#add. vars	decided
Naïve	$\binom{n}{k+1}$	0	immediate
Sequential unary counter ($\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$)	$\mathcal{O}(n \cdot k)$	$\mathcal{O}(n \cdot k)$	by unit prop.
Parallel binary counter ($\mathrm{LT}^{n,k}_{\mathrm{PAR}}$)	$7n - 3\lfloor \log n \rfloor - 6$	$2n - 2$	by search
Bailleux & Boufkhad [4]	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot \log n)$	by unit prop.
Warners [6]	$8n$	$2n$	by search

Table 1 shows the results. With respect to the number of clauses required, our encoding $\mathrm{LT}^{n,k}_{\mathrm{PAR}}$ is the best with a little less than $7n$ clauses; however, it requires search

to check whether the constraint is fulfilled or not. Among the encodings requiring no search, which are thus decidable in linear time, is that of Bailleux&Boufkhad and our $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ encoding. The latter performs better for small values of k, whereas the former is better for large bounds.

6 Lower Bound

Considering the different encodings given in the last section, the question arises in how far these results can still be improved. To handle this question, we prove a linear lower bound on the number of clauses required for any clausal encoding of cardinality constraints. This means that no sub-linear (e.g. logarithmic) encoding is possible. So, the formulae derived in the last section are asymptotically optimal with respect to the number of clauses (i.e. the encoding problem needs $\Omega(n)$ clauses) . However, there is still a gap remaining, in the sense that our lower bound can only show that at least n clauses are required, whereas the best encoding still uses almost $7n$ clauses. We believe, however, that the upper bound is closer to the exact complexity bound.

We want to show now that every encoding of $\leq k\ (x_1, \ldots, x_n)$ requires at least n clauses:

Theorem 3. *For all $n \in \mathbb{N}$ and all k with $0 \leq k < n-1$, each clausal (CNF) encoding of $\leq k\ (x_1, \ldots, x_n)$ requires at least n clauses.*

Proof. Our proof runs in two parts: First, we show that each literal $\neg x_i$ has to occur at least once in the clause set. Then, we prove that multiple negative literals $\neg x_i$ in the same clause make additional clauses unavoidable.

Let E be any CNF-encoding of $\leq k\ (x_1, \ldots, x_n)$. To prove the first part, assume that for some p with $1 \leq p \leq n$, literal $\neg x_p$ does not occur in the clause set E. We know that, by definition, every assignment $\alpha : \{x_1, \ldots, x_n\} \to \mathbb{B}$, in which exactly k variables are set to 1, can be extended to a satisfying assignment $\alpha^*$of E by assigning appropriate values to the auxiliary variables. Take any such assignment α with the additional property that $\alpha(x_p) = 0$. Such an α always exists, as $k < n$. Now, let β be α altered at point x_p (i.e. $\beta(x_i) = \alpha(x_i)$ for all $i \neq p$ and $\beta(x_p) = 1$) and β^* the respectively altered extension. As, by assumption, there is no clause containing $\neg x_p$, all clauses of E remain satisfied. Thus, β^* is a satisfying assignment of E in which $k+1$ variables are set to 1. This, however, is a contradiction to our assumption that E is a clausal encoding of $\leq k\ (x_1, \ldots, x_n)$.

Now, for the second part of our proof, assume that there is a clause $c \in E$ containing some literals $\neg x_p$ and $\neg x_q$ (with $p \neq q$) simultaneously. If there is no such clause, E must contain at least n clauses, one for each literal $\neg x_p$, and we are done.

Consider an assignment α with $\alpha(x_p) = \alpha(x_q) = 0$ and exactly k literals set to 1. As $k < n-1$ such an assignment always exists. By definition, α can be extended to a model α^* of E. Changing α to β by setting $\beta(x_q) = 1$, we arrive at an assignment with $k+1$ variables set to 1 and its extension β^* still satisfying clause c. As no extension of β, and thus neither β^*, is a model of E, there must be another clause $d \in E$ which is not satisfied by β^*, and thus both $\neg x_p \notin d$ and $\neg x_q \in d$ must hold. Thus, we have shown that for all p, q with $1 \leq p, q \leq n$ and $p \neq q$ there must be at least one clause

$c_{q,p}$ that contains $\neg x_q$, but does not contain $\neg x_p$. This requirement can only be met by at least n clauses.

We want to close this section with two notes: First, note that the case $k = n - 1$ can be accomplished with less than n clauses. One clause, namely $(\neg x_1 \vee \cdots \vee \neg x_n)$ is sufficient. For $k = n$ no restictions apply, and thus no clauses at all are required. Second, if $k \neq 0$, it turns out that all clauses in an optimal CNF-encoding of E must be non-unit clauses. To see this, assume that there exists a unit-clause that is essential, i.e. contains exactly one of the variables x_i and thus cannot be removed. Such a unit-clause must contain a negative literal (as can be seen from the prove above), thus it must be of the form $(\neg x_i)$. This, however, is a contradiction, as setting x_i to true must be allowed if $k > 0$.

7 Conclusions and Future Work

We have presented two methods for encoding Boolean cardinality constraints in conjunctive normal form. Both methods improve on existing results: one achieves the currently least known number of clauses (to the best of the author's knowledge), the other has advantages for constraints of the form $\leq k\,(x_1, \ldots, x_n)$ for small values of k.

Moreover, we have shown a lower bound on the number of clauses required for any clausal encoding of cardinality constraints. Such a proof touches the realm of Boolean function complexity [13, 14]. It might be an interesting topic for future research to see in how far results from this field are transferrable to the area of minimal clausal encodings.

We think that looking for improved lower bounds is worthwhile and still expect much room for improvement. Moreover, an experimental evaluation of the different encodings should be of interest.

Availability: We have implemented generators for both the $\mathrm{LT}^{n,k}_{\mathrm{SEQ}}$ and the $\mathrm{LT}^{n,k}_{\mathrm{PAR}}$ encoding in C++. They can be downloaded at http://www-sr.informatik.uni-tuebingen.de/~sinz/CardConstraints.

References

1. Sabin, D., Weigel, R.: Product configuration frameworks – a survey. IEEE Intelligent Systems **13** (1998) 42–49
2. Küchlin, W., Sinz, C.: Proving consistency assertions for automotive product data management. J. Automated Reasoning **24** (2000) 145–163
3. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency assignment. Constraints **4** (1999) 79–89
4. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In Rossi, F., ed.: 9th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2003). Volume 2833 of Lecture Notes in Computer Science., Springer (2003) 108–122
5. Stanica, P.: Good lower and upper bounds on binomial coefficients. Journal of Inequalities in Pure and Applied Mathematics **2** (2001)
6. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. **68** (1998) 63–69

7. Nonnengart, A., Rock, G., Weidenbach, C.: On generating small clause normal forms. In Kirchner, C., Kirchner, H., eds.: 15th Intl. Conf. on Automated Deduction (CADE-15). Volume 1421 of Lecture Notes in Computer Science., Springer (1998) 397–411
8. Jackson, P., Sheridan, D.: The optimality of a fast CNF conversion and its use with SAT. Technical Report APES-82-2004, APES Research Group (2004) Available from http://www.dcs.st-and.ac.uk/ apes/apesreports.html.
9. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In Silenko, A.O., ed.: Studies in Constructive Mathematics and Mathematical Logic. (1970) 115–125
10. Murray, N.V.: Completely non-clausal theorem proving. Artif. Intell. **18** (1982) 67–85
11. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. **2** (1986) 293–304
12. Muller, D.E., Preparata, F.P.: Bounds to complexities of networks for sorting and for switching. J. ACM **22** (1975) 195–201
13. Wegener, I.: The Complexity of Boolean Functions. Wiley-Teubner (1987)
14. Paterson, M., ed.: Boolean Function Complexity. Number 169 in London Mathematical Society Lecture Note Series. Cambridge University Press (1992)