Perfect Hashing and CNF Encodings of Cardinality Constraints

Yael Ben-Haim, Alexander Ivrii, Oded Margalit, and Arie Matsliah

IBM R&D Labs in Israel
Haifa University Campus, Mount Carmel, Haifa, 31905, Israel
{yaelbh,alexi,odedm,ariem}@il.ibm.com

Abstract. We study the problem of encoding cardinality constraints (threshold functions) on Boolean variables into CNF. Specifically, we propose new encodings based on (perfect) hashing that are efficient in terms of the number of clauses, auxiliary variables, and propagation strength. We compare the properties of our encodings to known ones, and provide experimental results evaluating their practical effectiveness.

1 Introduction

Modern Boolean satisfiability (SAT) solvers are powerful tools, capable of solving many practical problems with millions of variables within minutes. They come well-tuned off-the-shelf, allowing non-expert users to solve a wide range of complex problems quickly. However, using a SAT solver to solve general *constraint satisfaction problems* (CSPs) requires encoding the problem into strict *conjunctive normal form* (CNF). The method of encoding can dramatically affect the runtime of a SAT solver and its memory consumption. Hence, the problem of encoding CSPs into CNF is well studied within both the CSP and SAT communities.

In this paper, we consider the special (and probably the most common) case of encoding cardinality constraints (on Boolean variables) of the form $_{\leq k}(X_1, \ldots, X_n)$ into CNF. The $_{\leq k}(X_1, \ldots, X_n)$ constraint, on variables $X_1, \ldots, X_n$, is satisfied if at most k of them are assigned TRUE. A CNF encoding of $_{\leq k}(X_1, \ldots, X_n)$ is a formula F on variables $X_1, \ldots, X_n$ and (possibly) additional auxiliary variables $Y_1, \ldots, Y_\ell$, satisfying the following conditions:

- for any assignment x to $X_1, \ldots, X_n$ with at most k TRUEs (in short $|x| \leq k$), there is an assignment $y = y_1 y_2 \cdots y_\ell$ to $Y_1, \ldots, Y_\ell$ such that $(x \cup y) \models F$;
- if x is an assignment to $X_1, \ldots, X_n$ with $|x| > k$, then for all assignments y to $Y_1, \ldots, Y_\ell$, $(x \cup y) \not\models F$.

The constraints $_{\circ k}(X_1, \ldots, X_n)$, $\circ \in \{<, >, \geq, =\}$, and their CNF encodings are defined similarly, and can be all translated to at most two[1] constraints of the form $_{\leq k}(X_1, \ldots, X_n)$. Therefore, throughout this paper (except in Section 4.2), we will focus on the $_{\leq k}(X_1, \ldots, X_n)$ type, with a further restriction of $0 < k < n$ (the cases $k = 0$ and $k = n$ can be handled trivially).

[1] One in all cases other than '='.

A. Cimatti and R. Sebastiani (Eds.): SAT 2012, LNCS 7317, pp. 397–409, 2012.

Cardinality constraints arise naturally in many problems with optimization flavor (e.g., "Is there a solution in which at most/at least k of the variables (from a certain set) are set to TRUE/FALSE?"), and in problems where multi-valued variables are expressed with Boolean variables (e.g., "Is there a solution in which exactly one of the Boolean variables representing each multi-valued variable is set to TRUE?"). Furthermore, any symmetric function on n Boolean variables (i.e., a function whose value depends only on the number of input variables assigned TRUE) can be expressed as a disjunction of at most n cardinality constraints.

1.1 Efficient Encodings and Related Work

An efficient CNF encoding F of the cardinality constraint $\leq_k(X_1, \ldots, X_n)$ has the following characteristics:

- Few clauses.
- Few auxiliary variables.
- One (or better both) of the following properties preserving arc-consistency under unit propagation:
  - $\mathbf{P}_{\text{confl}}$**:** for any partial assignment $\hat{x}$ to X with $|\hat{x}| > k$, the formula F under $\hat{x}$ implies the empty clause (contradiction) with unit propagation (UP);
  - $\mathbf{P}_{\text{extend}}$**:** for any partial assignment $\hat{x}$ to X with $|\hat{x}| = k$, the formula F under $\hat{x}$ assigns FALSE to all unassigned variables in X with UP (note that this property implies the first one, but not vice versa).

In short, a (c, a, p) *encoding* is an encoding with $c(n, k)$ clauses, $a(n, k)$ auxiliary variables and propagation strength $p \in \{\mathbf{P}_{\text{confl}}, \mathbf{P}_{\text{extend}}, \emptyset\}$.

The naive (often called *binomial*) encoding of $\leq_k(X_1, \ldots, X_n)$ has parameters $(\binom{n}{k+1}, 0, \mathbf{P}_{\text{extend}})$; it is a conjunction of all[2] $\binom{n}{k+1}$ clauses of the form $(\neg X_{i_1} \vee \cdots \vee \neg X_{i_{k+1}})$. The exponential dependance of its size on k makes the naive encoding impractical except for small n and k. The following table summarizes several interesting methods for encoding the $\leq_k(X_1, \ldots, X_n)$ constraint into CNF efficiently:

Name	parameters	origin
Sequential counter	$(O(kn), O(kn), \mathbf{P}_{\text{extend}})$	[Sin05]
Parallel counter	$(O(n), O(n), \emptyset)$	[Sin05]
Binary*	$(O(kn \log n), O(kn), \emptyset)$	[FPDN05, FG10]
Product	$(O(kn + k^2 n^{k/(k+1)}), O(kn^{k/(k+1)}), \mathbf{P}_{\text{extend}})$	[FG10]
Commander	$(O(2^{2k} n), O(kn), \mathbf{P}_{\text{extend}})$	[FG10]
Sorting networks	$(O(n \log^2 n), O(n \log^2 n), \mathbf{P}_{\text{extend}})$	[ES06]
Cardinality networks	$(O(n \log^2 k), O(n \log^2 k), \mathbf{P}_{\text{extend}})$	[ANOR09]
Totalizer	$(O(n^2), O(n \log^n), \mathbf{P}_{\text{extend}})$	[BB03]
Linear	$(O(n), O(n), \emptyset)$	[War98]

(* The binary encoding for $k = 1$ has parameters $(O(n \log n), \log n, \mathbf{P}_{\text{extend}})$ [FPDN05].) While the parallel-counter based encoding is smallest in size, it lacks the propagation strength that is crucial for SAT solver performance. On the other hand, encodings with

[2] Recall that we assume $k < n$.

strong propagation properties have a size that is super-linear in n, making them impractical for very large n and k. In their recent work, Frisch and Giannaros [FG10] discuss these tradeoffs, and survey and compare several known and new encodings. As [FG10] conclude, the sequential-counter based encoding from [Sin05] seems to perform better than other known encodings in practice (for problems that are not considered too small), and it is the method we use here as a benchmark against our encodings[3].

1.2 Our Contribution

PHF-Based Encoding. In Section 4, we propose a method for encoding cardinality constraints into CNF with perfect hash functions (PHFs) (see the definition in Section 2). The high-level idea behind this method is to reduce (using PHFs) the problem of encoding $_{\leq k}(X_1, \ldots, X_n)$ to several disjoint problems of encoding $_{\leq k}(Y_1, \ldots, Y_r)$ for $r \ll n$. From the special structure of PHFs, we get the following: 1) any (partial) assignment that satisfies the original constraint can be extended to an assignment that satisfies all the smaller constraints; 2) any assignment falsifying the original constraint must also falsify one of the smaller constraints. Hence, it suffices to enforce the smaller constraints only, using any of the known encodings at hand. Furthermore, this reduction inherits propagation strength and incrementality[4] of the used encodings. When constructed using the sequential-counter based method (on the reduced constraints), we obtain an encoding that uses the fewest number of auxiliary variables among all encodings listed in the table above. The parameters of our encodings are of the form $(nk^c \log n, k^c \log n, \mathbf{P}_{\text{extend}})$, where c depends on the PHF used (and can be as small as 4).

Encoding At-Least-k. In Section 4.2, we describe how to encode the $_{\geq k}(X_1, \ldots, X_n)$ constraint using PHFs. Although $_{\geq k}(X_1, \ldots, X_n)$ can be trivially reformulated in terms of an at-most type constraint (namely, $_{\leq n-k}(\neg X_1, \ldots, \neg X_n)$), the problem with this transformation is that for $1 < k \ll n/2$ it blows up the size of the encoding in all but the non-propagating methods. We show how PHF and sequential-counter based encodings can handle $_{\geq k}(X_1, \ldots, X_n)$ constraints natively, without any size penalties. In fact, the encodings we get have parameters $(k^c \log n, k^c \log n, \mathbf{P}_{\text{confl}})$, where c again is a small constant depending on the PHF used. Note that here both the number of clauses and the number of auxiliary variables are subliniar in n (however, the clauses themselves are larger).

Hybrid Encoding. While interesting from a theoretical point of view, our experience shows that the PHF based methods become practical only when k is significantly smaller than n (e.g., a small constant versus hundreds of thousands). This is because the

[3] The performance of the parallel-counter based encodings from [Sin05] was not measured in [FG10]. However, as our results in Section 6 indicate, despite being smallest in size, their lack in propagation strength makes them much worse in practice than the sequential-counter based encodings.

[4] Incrementality is the property of an encoding that allows tightening the cardinality bound by setting values to some variable(s). This property is useful when applying a sequence of decreasing/increasing cardinality constraints in the process of search for the optimal value.

number of copies of the reduced problem (corresponding to the number of hash functions in a perfect hash family) grows quadratically in k. To overcome the impracticality of the PHF based encodings, we propose a hybrid encoding, combining simple (non-perfect) hashing, parallel-counter, and sequential-counter based encodings. The idea here is similar: reduce one big problem to several small ones (but fewer than before), and enforce the smaller constraints with the sequential-counter method. In contrast to the previous construction, here we do not require perfect reduction; that is, we may have a situation where, under some assignment, the original constraint is falsified, but all the small ones to which we reduced are satisfied. Therefore, to make the encoding still correct, we add the parallel-counter based encoding on the original variables. Due to the exponential coverage vs the number-of-copies nature of hashing, we can guarantee that the strong propagation properties of the sequential-counter method are preserved for most of the assignments, even when reducing to only a single small problem. Namely, this hybrid encoding enjoys partial $\mathbf{P}_{\mathrm{extend}}$ propagation strength, in the sense that most (but not all) of the partial assignments with $\geq k$ TRUEs are forcing propagation as required. However, it is much smaller in size (e.g., for $k \ll n$, an encoding that propagates most ($> 99\%$) of the partial assignments has only $O(n)$ clauses and auxiliary variables, and is nearly the same in size as the asymptotically-optimal parallel-counter based encoding). In addition, the hybrid encoding is simpler to implement. It can be viewed as a simple way to augment the parallel-counter encoding with propagation strength using a small number of small sequential counters.

Experimental Evaluation. The experimental evaluation in Section 6 compares various versions of the hybrid encoding to the sequential and parallel-counter based methods on a benchmark set containing encodings of different optimization and scheduling problems, kindly provided to us by the authors of [ANOR09].

It is clear that there is a sharp time-memory tradeoff between the counter-based encodings. Namely, whenever it is possible, with respect to memory limitations, to encode the constraints with the sequential-counter based method, the solver performance improves dramatically, compared to the parallel-counter based encoding. On the other hand, the parallel-counter based encoding is very efficient in terms of memory, but slower due to lack of propagation strength.

Our bottom-line conclusion is that with the hybrid encoding, we can enjoy both worlds: it is as fast as the sequential-counter based encoding (even faster), and its memory consumption is very close to that of the parallel-counter based encoding.

2 Perfect Hashing

An (n, ℓ, r, m) perfect hash family (PHF) is a collection $\mathcal{H} = h_1, \ldots, h_m$ of functions mapping $[n] = \{1, \ldots, n\}$ to $[r]$ that satisfies the following property: for every subset $S \subseteq [n]$ of size $|S| \leq \ell$, there is $i \in [m]$ such that $|h_i(S)| = |S|$. Namely, at least one of the functions hashes S perfectly, with no collisions. Fixing ℓ and r, we usually look for the smallest $m = m(n)$ for which an (n, ℓ, r, m) perfect hash family exists.

Naturally, the task becomes easier when r is larger than ℓ (and is impossible when $r < \ell$). For the case $r = \ell$, m can be bounded by $O(\ell e^{\ell} \log n)$. Allowing $r = \ell^2$,

m can be bounded by $O(\ell \log n)$. These upper bounds can be obtained using standard probabilistic arguments (for example, see [CLRS09]). However, constructing such families explicitly and efficiently imposes additional penalty on their size. See [KM03] and [FK84] for more details on explicit constructions of PHFs.

Before we describe how PHFs are useful for cardinality-constraint encodings, we sketch three simple constructions of PHFs for small ℓ and r. The first construction is straightforward, the second is (to our knowledge) new, and the third one was proposed to us by Noga Alon.

3 Perfect Hashing – Constructions

3.1 $(n, 2, 2, \lceil \log n \rceil)$ PHF

Set $m \triangleq \lceil \log n \rceil$. For $i \in [m]$ and $j \in [n]$, let $h_i(j)$ map to the ith bit of j, when j is written in binary base. Since every two numbers differ in at least one bit, the functions h_i form a $(n, 2, 2, \lceil \log n \rceil)$ PHF.

3.2 $(n, 3, 3, \lceil \log_3^2 n \rceil)$ PHF

For $i_1, i_2 \in [\log_3 n]$, and $j \in [n]$, write j in ternary base and define

$$h_{i_1,i_2}(j) = \begin{cases} (i_1\text{th digit of } j + i_2\text{th digit of } j) \mod 3, & i_1 < i_2 \\ i_1\text{th digit of } j, & i_1 = i_2 \\ (i_1\text{th digit of } j - i_2\text{th digit of } j) \mod 3, & i_1 > i_2 \end{cases}.$$

We now prove the property of perfect hashing. Let j_1, j_2, and j_3 be three different indices $\in [n]$. We need to show the existence of i_1, i_2 such that

$$h_{i_1,i_2}(j_1) \neq h_{i_1,i_2}(j_2),\ h_{i_1,i_2}(j_1) \neq h_{i_1,i_2}(j_3),$$

and $h_{i_1,i_2}(j_2) \neq h_{i_1,i_2}(j_3)$. Since $j_1 \neq j_2$, there exists i_1 such that the i_1th digit of j_1 differs from the i_1th digit of j_2. If the i_1th digits of j_3 has the (remaining) third value, then h_{i_1,i_1} separates the three j's, and we are done. Otherwise, j_3 has the same i_1th digit as, w.l.o.g, j_1. So let i_2 be a digit in which j_1 differs from j_3. W.l.o.g $i_1 < i_2$. If all three j's have different i_2th digits, h_{i_2,i_2} separates the three j's, and we are done. Otherwise, j_2 has the same i_2th digit as either j_1 or j_3. W.l.o.g, the i_2th digit of j_2 is the same as the i_2th digit of j_1; this is depicted in the following table:

j	i_1th digit	i_2th digit
j_1	x	z
j_2	y	z
j_3	x	w

Here $x \neq y$ and $z \neq w$ are four ternary digits. Since $x + z \neq y + z$ and $x + z \neq x + w$, the only way h_{i_1,i_2} will not separate the three j's is if $y + z = x + w$. Similarly, the only way h_{i_2,i_1} will not separate j_1, j_2, and j_3 is if $y - z = x - w$. But adding these two equalities yields the contradiction $x = y$, so either h_{i_1,i_2} or h_{i_2,i_1} must separate j_1, j_2, and j_3.

3.3 $(n, \ell, \widetilde{O}(\ell^2 \log n), \ell^2 \log n)$ PHF

This construction is useful in practice for $\ell \ll n$.

Let $p_1, \ldots, p_m$ be the first m prime numbers. For every $i \in [m]$, let the function h_i map $[n]$ to $\{0, \ldots, p_i - 1\}$ as follows:

$$h_i(j) = j \mod p_i.$$

Now let $S \subseteq [n]$ be a set that is not hashed perfectly by any of the functions h_i. This means that for every $i \in [m]$ there are $j \neq j' \in S$ so that

$$j = j' \mod p_i.$$

Equivalently, for all $i \in [m]$

$$p_i \mid \prod_{j<j' \in S} (j' - j),$$

and since p_i are primes,

$$p_1 \cdot p_2 \cdots p_m \mid \prod_{j<j' \in S} (j' - j).$$

Since $p_1 \cdot p_2 \cdots p_m > 2^m$ and $\prod_{j<j' \in S}(j' - j) < n^{|S|^2}$, we must have $m < |S|^2 \log n$. In other words, setting $m = \ell^2 \log n$ assures that every set of size $\leq \ell$ is perfectly hashed by some function h_i.

The largest prime used, p_m, is the upper bound on r in this construction, which by the Prime Number Theorem is bounded by $O(m \ln m)$. Combined with the inequality above, we get

$$r < O(|S|^2 \log n(\log |S| + \log \log n)) = \widetilde{O}(\ell^2 \log n).$$

4 New Encodings Based on PHFs

In this section, we describe how to encode the ${}_{\leq k}(X_1, \ldots, X_n)$ and ${}_{\geq k}(X_1, \ldots, X_n)$ constraints into CNF using PHFs. We start with the general parameterized constructions, analyze them, and then instantiate them with several different parameters for comparison against known encodings.

4.1 Encoding the ${}_{\leq k}(X_1, \ldots, X_n)$ Constraint

Fix $r > k$ and an $(n, k+1, r, m)$ perfect hash family $\mathcal{H} = h_1, \ldots, h_m : [n] \to [r]$. Perform the following steps for all $i \in [m]$:

1. Introduce r auxiliary variables $Y_1^i, \ldots, Y_r^i$.
2. Encode implications $X_j \to Y_{h_i(j)}^i$ (using binary clauses $(\neg X_j \vee Y_{h_i(j)}^i)$) for all $j \in [n]$.
3. Encode an ${}_{\leq k}(Y_1^i, \ldots, Y_r^i)$ constraint using the sequential-counter based encoding from [Sin05].

The final CNF encoding of ${}_{\leq k}(X_1, \ldots, X_n)$ is the conjunction of all the clauses generated in Step 2 and Step 3 for $i = 1, \ldots, m$.

Correctness. To verify that the encoding is correct, assume first that we start with an assignment x to $X = X_1, \ldots, X_n$ with weight $> k$. Let $b_1, \ldots, b_{k+1}$ be the first $k+1$ indices corresponding to variables assigned TRUE in x. Since $\mathcal{H}$ is an $(n, k+1, r, m)$ perfect hash family, there must be $i \in [m]$ so that $h_i(b_j) \neq h_i(b_{j'})$ for all $1 \leq j < j' \leq k+1$. In addition, by the implication clauses introduced in Step 2, the $k+1$ distinct Y^i_j variables to which $h_i(b_j)$ are mapped (for $j = 1, \ldots, k+1$) are forced to be TRUE. Consequently, the $\leq_k(Y^i_1, \ldots, Y^i_r)$ constraint encoding introduced in Step 3 for that particular index i is falsified.

Now assume that we start with an assignment x to X with weight $\leq k$. Then, by construction, each one of the i copies of the auxiliary variables can be assigned values of overall weight $\leq k$, and furthermore, the corresponding $\leq_k(Y^i_1, \ldots, Y^i_r)$ constraints can all be simultaneously satisfied.

Size. The encoding requires $m \cdot n + m \cdot O(r \cdot k)$ clauses (the first term comes from Step 2, and the second term comes from Step 3) and it uses $m \cdot r + m \cdot O(r \cdot k)$ auxiliary variables (the first term is the Y variables, and the second term comes from the sequential counter encodings on the Y's).

Propagation strength. The encoding is $\mathbf{P}_{\mathrm{extend}}$: Suppose that a partial assignment $\hat{x}$ to X has exactly k variables set to TRUE. We show that unit propagation sets all other variables to FALSE. Let $b_1, \ldots, b_k$ be the k indices corresponding to variables assigned TRUE, and let b_{k+1} be the index of any other variable. Using the perfect hashing property, take h_i so that $h_i(b_j) \neq h_i(b_{j'})$ for all $1 \leq j < j' \leq k+1$. The implications clauses of Step 2 imply by unit propagation a partial assignment $\hat{y}$ on $Y^i_1, \ldots, Y^i_r$ with $Y^i_{h_i(b_j)}$ set to TRUE for $1 \leq j \leq k$. Since the sequential counter (Step 3) is $\mathbf{P}_{\mathrm{extend}}$, unit propagation extends $\hat{y}$ by assigning FALSE to all other variables in that copy, including the variable $Y^i_{h_i(b_{k+1})}$. This triggers again the binary implications from Step 2, and unit propagation assigns $X_{b_{k+1}}$ to FALSE.

Instantiation. Using the probabilistic[5] PHFs with $\ell = k+1, r = \ell^2$ and $m = O(\ell \log n)$ (see Section 2), we get $(O(nk \log n + k^4 \log n), O(k^4 \log n), \mathbf{P}_{\mathrm{extend}})$ encoding for $\leq_k(X_1, \ldots, X_n)$. Compared, e.g., to the sequential-counter based encoding, this construction uses slightly more clauses (a factor of $\log n$), but for large enough n it consumes a significantly smaller number of variables ($k^4 \log n$ vs kn). Furthermore, apart from the naive encoding (whose size is exponential in k) this encoding uses the fewest number of auxiliary variables among all encoding methods listed above.

Instantiating this construction with $k = 1$ and $\ell = r = 2$ based on the corresponding PHF from Section 3.1, we get $(n \log n, \log n, \mathbf{P}_{\mathrm{extend}})$ encoding for $\leq_1(X_1, \ldots, X_n)$, with properties similar to the binary encoding [FPDN05]. The PHFs from Sections 3.2 (for $k = 2$) and 3.3 (for any k) yield to $(n \log^2 n, \log^2 n, \mathbf{P}_{\mathrm{extend}})$ and $(O(k^2 n \log n + k^5 \log^2 n(\log k + \log \log n)), O(k^5 \log n), \mathbf{P}_{\mathrm{extend}})$ encodings, respectively.

[5] We stress that probabilistic arguments are only required to prove the existence of such families; in practice, one can find such PHFs by brute-force search, and then hard-code them in the software once and for all. In the cost of size penalty, there are also derandomized versions of the probabilistic constructions [AN96].

4.2 Encoding the $\geq_k(X_1, \ldots, X_n)$ Constraint

The straightforward way to implement the at-least constraints is to negate the at-most constraints: having at least k TRUEs out of n variables is equivalent to having at most $n-k$ TRUEs out of the n negated literals. The problem with this simple transformation is that moving from $k \ll n$ to $k \approx n$ imposes a big size penalty for all but the non-propagating encodings (this is due to the fact that encoding size grows with k). In this section we show how the PHF-based encodings can handle the $\geq_k(X_1, \ldots, X_n)$ constraint natively. To this end, we need to invert the sequential-counter based encoding.

A Sequential-Counter Based Encoding for the $\geq_k(X_1, \ldots, X_n)$ Constraint

Construction. The sequential-counter circuit described in [Sin05] is an example of a monotone circuit with n inputs $X_1, \ldots, X_n$ and a single output Z computing the $\geq_k(X_1, \ldots, X_n)$ constraint. In other words, given a full assignment to all of the X_is, Z evaluates to TRUE if and only if the number of X_is assigned to TRUE is at least k. One can convert this circuit into CNF via the Tseitsin encoding (see also Theorem 2 from [BKNW09] and Theorems 7 and 8 from [Bai11] for the connection between monotone circuits and the corresponding CNF encodings), and to add the unit clause (Z) enforcing the output of the circuit to be TRUE. The resulting CNF has $O(kn)$ clauses, $O(kn)$ auxiliary variables[6], correctly encodes the $\geq_k(X_1, \ldots, X_n)$ constraint, and is $\mathbf{P}_{\mathrm{extend}}$.

Optimization. One can optimize the above construction due to polarity considerations, encoding the $\geq_k(X_1, \ldots, X_n)$ constraint as follows:

- Introduce the auxiliary variables $Y_{i,j}$ for all $i \in [n]$ and $j \in [k]$.
- Introduce the clauses:
  1. $(\neg Y_{1,1} \vee X_1)$,
  2. $(\neg Y_{1,j})$ for for $1 < j \leq k$,
  3. $(\neg Y_{i,j} \vee Y_{i-1,j-1})$ for $1 < i \leq n$ and $1 < j \leq k$,
  4. $(\neg Y_{i,j} \vee Y_{i-1,j} \vee X_i)$ for $1 < i \leq n$ and $1 \leq j \leq k$,
  5. $(Y_{n,k})$.

The Boolean variables $Y_{i,j}$ represent the statements "at least j out of $X_1, \ldots, X_i$ are TRUE". The clauses from (3) enforce the implications $Y_{i,j} \rightarrow Y_{i-1,j-1}$, and the clauses from (4) enforce the implications $Y_{i,j} \rightarrow (Y_{i-1,j} \vee X_i)$. The unit clause (5) corresponds to the "output of the circuit". (The unit clause is not added when using this encoding as a building block.) This version of the $\geq_k(X_1, \ldots, X_n)$ encoding has roughly $2kn$ clauses and kn auxiliary variables, and is $\mathbf{P}_{\mathrm{extend}}$.

A PHF-Based Encoding for the $\geq_k(X_1, \ldots, X_n)$ Constraint Fix $r > k$ and an (n, k, r, m) perfect hash family $\mathcal{H} = h_1, \ldots, h_m : [n] \rightarrow [r]$. Perform the following steps for each $i \in [m]$:

1. Introduce an auxiliary variable Z_i and r auxiliary variables $Y_1^i, \ldots, Y_r^i$.
2. Introduce implications $Y_d^i \rightarrow (\bigcup_{h_i(j)=d} X_j)$ for all $d \in [r]$.

[6] Here we can see the advantage of having $k \ll n$.

3. Encode the $_{\geq k}(Y_1^i, \ldots, Y_r^i)$ using the sequential-counter based construction described above. Let Z_i denote the variable corresponding to the circuit's output.

The final CNF encoding is the conjunction of the clauses in Step 2, the clauses in Step 3 for $i = 1, \ldots, m$, and the clause $(\bigcup_{i=1}^{m} Z_i)$.

Correctness. Let x be an assignment to $X_1, \ldots, X_n$ with weight $\geq k$, and let $b_1, \ldots, b_s$ ($s \geq k$), be the indices corresponding to the variables assigned TRUE in x. We can extend this assignment to Y_j^is by assigning Y_j^i to TRUE whenever $Y_j^i = h_i(b_j)$ for some j; this assignment satisfies all of the clauses in Step 2. This assignment (uniquely) extends to satisfy all of the clauses in Step 3, and for every i, Z_i is true if and only if the weight of the assignment to $Y_1^i, \ldots, Y_m^i$ is at least k. Since H is an (n, k, r, m) PHF, there exists at least one $i \in [m]$ with $h_i(b_j) \neq h_i(b_{j'})$ for all $1 \leq j < j' \leq k$. Consequently, for that index i, Z_i is TRUE and the remaining clause $(\bigcup_{i=1}^{m} Z_i)$ is also satisfied.

For the other direction, assume that x has weight $< k$. Then by Step 2 for all $i \in [m]$, there are at most $k - 1$ values of d for which Y_d^i can be TRUE. It follows by Step 3 that each Z_i is always FALSE, in conflict with the clause $(\bigcup_{i=1}^{m} Z_i)$.

Properties. The encoding has $O(m \cdot r \cdot k)$ clauses[7] and $O(m \cdot r \cdot k)$ variables, and is $\mathbf{P}_{\text{confl}}$. The $\mathbf{P}_{\text{confl}}$ property can be shown directly, but we do not present it here since it also follows by Theorem 2 from [BKNW09] and Theorems 7 and 8 from [Bai11]. We leave as an open question if it is possible to modify the construction to be able to enforce the stronger $\mathbf{P}_{\text{extend}}$ property as well.

Instantiation. Using the probabilistic PHFs with $\ell = k+1$, $r = \ell^2$ and $m = O(\ell \log n)$ (see Section 2), we get $(O(k^4 \log n), O(k^4 \log n), \mathbf{P}_{\text{confl}})$ encoding for $_{\geq k}(X_1, \ldots, X_n)$. The PHFs from Sections 3.2 (for $k = 2$) and 3.3 (for any k) yield $(\log^2 n, \log^2 n, \mathbf{P}_{\text{confl}})$ and $(O(k^5 \log n), O(k^5 \log n), \mathbf{P}_{\text{confl}})$ encodings, respectively.

5 Hybrid Encodings

In this section we show how to augment the parallel-counter based encoding (which is nearly optimal in size) with partial propagation strength, using (non-perfect) hash functions.

Given c hash functions $\mathcal{H} = \{h_i : [n] \to [r]\}_{i=1}^{c}$, a hybrid encoding of the $_{\leq k}(X_1, \ldots, X_n)$ constraint is formed as follows:

- Enforce the $_{\leq k}(X_1, \ldots, X_n)$ constraint on $X_1, \ldots, X_n$ using the parallel-counter based method.
- For each $1 \leq i \leq c$:
  - Add implications $X_j \implies Y_{h_i[j]}^i$ for all $j \in [n]$.
  - Enforce the $_{\leq k}(Y_1^i, \ldots, Y_r^i)$ constraint on $Y_1^i, \ldots, Y_r^i$ using one of the known methods.

[7] Note that the number of clauses is significantly smaller than for the at-most constraint, however the clauses themselves are larger.

The correctness of this encoding follows by construction. It has $O\Big(c \cdot (n + cl(k,r))\Big)$ clauses and uses $O\Big(n + c \cdot aux(k,r)\Big)$ auxiliary variables, where $cl(k,r)$ and $aux(k,r)$ are the number of clauses and auxiliary variables used by the method of encoding applied to enforce each of the $\leq_k(Y_1^i, \ldots, Y_r^i)$ constraints. When using one of the encodings with $\mathbf{P}_{\mathrm{extend}}$ we also inherit a "partial $\mathbf{P}_{\mathrm{extend}}$", where ρ-fraction of all partial assignments with weight $\leq k$ are propagating as required. The parameter ρ depends on c, r, k and n, and a crude lower bound on it can be obtained with elementary calculation. In practice, however, we observed that setting $r = 2k$ has already a positive effect on propagation strength, even when using only one hash function (i.e. with $c = 1$). In Section 6 we evaluate hybrid encodings that use the sequential-counter based method to enforce the constraints $\leq_k(Y_1^i, \ldots, Y_r^i)$ with $c = 1, 2, 3, 5$ and $r = 2k$.

6 Experiments

In this section we compare various versions of the hybrid encoding to the sequential and parallel-counter based encodings[8] on a benchmark set produced from the Partial Max-SAT division of the Third Max-SAT evaluation[9]. The benchmarks were produced (and kindly provided to us) by the authors of [ANOR09]. Out of roughly 14,000 benchmarks we extracted all those (1344) instances containing at least one cardinality constraint with $1 < k < n$ and $n > 10k$.

6.1 Setting

We ran the experiments on a Linux based workstation with Intel Xeon E5430 processor and 8GB of RAM. The encoding time was not counted, and it was negligible for all but the sequential-counter based method (where it was still small compared to solving time). As a SAT solver we used *Minisat 2.2.0* (with preprocessing enabled).

6.2 Results – Conclusion

Our bottom-line conclusion is that with the simplest version of the hybrid encoding, where $r = 2k$ and $c = 1$ (see Section 5), we can enjoy both of the advantages of the counter based encodings: the hybrid encoding is even faster than the sequential-counter based encoding, and its memory consumption is very close to that of the parallel-counter based encoding.

Another interesting observation is that the performance of hybrid encodings deteriorates (quite consistently) when the number of hash functions increase. This may be explained by the fact that the number of new tuples perfectly hashed by each additional hash function decreases exponentially faster than the increase in the encoding size.

[8] As a byproduct, our experiments compare the two counter-based methods from [Sin05] among themselves.

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

6.3 Results – Details

The following notation is used in all tables/plots below.

- The sequential and parallel-counter based encodings are denoted by Seq and Par respectively.
- Hybrid encodings with $r = 2k$ and 1,2,3, and 5 hash functions (copies), all using the sequential-counter based encoding for the small constraints, are denoted by HybSeq1, HybSeq2, HybSeq3, and HybSeq5 respectively.
- A hybrid encoding with $r = 2k$ and 1 hash function, using the parallel-counter based encoding for the small constraints, is denoted by HybPar1.

We analyze several aspects of encodings' performance:

Table 1. Cell i, j indicates the number of instances (out of 1344) for which the encoding in the ith row is strictly faster than the encoding in the jth column

	Seq	HybSeq1	HybSeq2	HybSeq3	HybSeq5	Par	HybPar1
Seq	0	362	479	626	909	803	491
HybSeq1	963	0	919	1028	1158	918	723
HybSeq2	847	398	0	938	1138	803	553
HybSeq3	702	301	376	0	1060	743	464
HybSeq5	428	175	191	270	0	645	284
Par	520	403	518	589	687	0	355
HybPar	830	590	776	866	1044	965	0

Table 2. Counting for each encoding the number of instances on which it was faster than all other encodings

	Seq	HybSeq1	HybSeq2	HybSeq3	HybSeq5	Par	HybPar1
#fastest	173	422	179	83	19	142	326

Table 3. Means and medians of various measures for each encoding

	Seq	HybSeq1	HybSeq2	HybSeq3	HybSeq5	Par	HybPar1
Medians							
Run time (seconds)	1.54	1.09	1.31	1.48	1.72	1.68	1.13
Number of clauses	186886	107678	127632	147978	181534	88876	95360
Number of variables	116624	72544	80354	88898	103414	65042	65828
Number of decisions	1835	15128	13280	14636	16536	103340	43192
Number of conflicts	208	2470	2214	2128	2126	9992	5429
Memory consumption (MB)	39	24	26	29	34	22	23
Means							
Run time (seconds)	16.57	15.08	15.68	17.34	21.76	31.76	18.9
Number of clauses	560204	218825	296379	371470	521646	144145	158424
Number of variables	391897	192718	228558	264372	336001	156930	158953
Number of decisions	42825	438311	185851	179692	181976	1991480	799956
Number of conflicts	23596	41508	37271	40083	42076	113130	58261
Memory consumption (MB)	123	52	66	77	101	42	43

- Size of the resulting CNF formula, as reflected by the number of clauses, the number of variables, and memory consumption of the SAT solver.
- Solver run time (counting parsing and preprocessing as well).
- Propagation strength, as reflected by the number of decisions and the number of conflicts during the run of the SAT solver.

7 Concluding Remarks and Future Work

We presented new methods to encode cardinality constraints into CNF based on perfect hashing. This approach, coupled with existing methods, leads to encodings with sublinear number of clauses and auxiliary variables, and strong propagation properties.

From a practical perspective, we proposed to use the hybrid approach (with non-perfect hashing) to boost the performance of existing counter based encodings.

We list the following directions for further research.

Optimizing the Ingredients: Several components of our encodings can be tuned or replaced for performance optimization: the underlying encoding applied on the reduced problems, the values of r and c (see Sections 2 and 5), and the underlying perfect hash family constructions. The amount of freedom is large, and we are convinced that the experimental results can be further improved.

Application to More General Constraints: It is possible that similar approach, based on perfect hash families and other related combinatorial structures like block designs, can be used for efficient encoding of other Pseudo-Boolean constraints.

Acknowledgments. We thank the authors of [ANOR09] for sharing with us their benchmark set, and Noga Alon for proposing the construction in Section 3.3. We also thank the anonymous reviewer for pointing out [BKNW09] and [Bai11], whose results were partially reproved in the initial version of this manuscript.

References

[AN96] Alon, N., Naor, M.: Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. Algorithmica 16(4/5), 434–449 (1996)

[ANOR09] Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality Networks and Their Applications. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 167–180. Springer, Heidelberg (2009)

[Bai11] Bailleux, O.: On the expressive power of unit resolution. Technical Report arXiv:1106.3498 (June 2011)

[BB03] Bailleux, O., Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003)

[BKNW09] Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence, IJCAI 2009, pp. 412–418. Morgan Kaufmann Publishers Inc., San Francisco (2009)

[CLRS09] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)

[ES06] Eén, N., Sörensson, N.: Translating Pseudo-Boolean constraints into SAT. JSAT 2(1-4), 1–26 (2006)

[FG10] Frisch, A.M., Giannaros, P.A.: SAT encodings of the at-most-k constraint. In: ModRef (2010)

[FK84] Fredman, M.L., Komlós, J.: On the size of separating systems and families of perfect hash functions. SIAM Journal on Algebraic and Discrete Methods 5(1), 61–68 (1984)

[FPDN05] Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.: Solving non-Boolean satisfiability problems with stochastic local search: A comparison of encodings. J. Autom. Reasoning 35(1-3), 143–179 (2005)

[KM03] Kim, K.-M.: Perfect hash families: Constructions and applications. Master Thesis (2003)

[Sin05] Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)

[War98] Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. 68(2), 63–69 (1998)