SAT Encodings of the At-Most-k Constraint

Some Old, Some New, Some Fast, Some Slow

Alan M. Frisch and Paul A. Giannaros [⋆]

Artificial Intelligence Group, Department of Computer Science,
University of York, United Kingdom.
`frisch@cs.york.ac.uk`

Abstract. This paper examines the encoding into SAT of the $\leq_k(X_1, \ldots, X_n)$ constraint, which is true if and only if at most k of the Boolean variables $X_1, \ldots, X_n$ are true. Though this is probably the constraint most commonly encoded into SAT, the encoding that is usually used is inferior to several other encodings. This paper surveys the available encodings, introduces some new encodings, and presents some experiments evaluating their performance.

1 Introduction

Considering the phenomenal advancement in the state of the art in SAT solvers and in their application to hard problems, progress in the understanding of SAT encodings and their use in practice seems much slower. This paper considers a prime example: the SAT encoding of the $\leq_k(X_1, \ldots, X_n)$ constraint, which is true if and only if at most k of the Boolean variables $X_1, \ldots, X_n$ are true.

The $\leq_k(X_1, \ldots, X_n)$ constraint is probably the constraint most commonly encoded into SAT. Here's why: many naturally arising problems can be conceived of as requiring the assignment of values to non-Boolean variables, variables whose domain of potential values has more than two elements. A non-Boolean variable whose domain is of size n is most often encoded as a set of Boolean variables $X_1, \ldots, X_n$ where the assignment of *TRUE* to X_i indicates that the original variable is assigned the i^{th} value from its domain. Since the original variable must be assigned exactly one variable, we must ensure that exactly one of $X_1, \ldots, X_n$ is assigned *TRUE*. This is most commonly considered as two constraints: $\leq_1(X_1, \ldots, X_n)$ and $\geq_1(X_1, \ldots, X_n)$. This $\leq_1(X_1, \ldots, X_n)$ constraint is most commonly encoded as

$$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \neg X_i \vee \neg X_j,$$

which we call the binomial encoding.

[⋆] We are grateful to Jingchao Chen for providing us with a copy of his paper prior to its publication, thus enabling us to generalise his product encoding and to experiment with it. Thanks also go to Neil Moore for comments on our paper.

There is much about SAT encodings that is unclear, but one thing that is clear — though perhaps not widely known — is that the binomial encoding is inferior to several others. That's right — the constraint most widely encoded in SAT is usually encoded in an inferior manner.

This paper addresses two questions: what alternative SAT encodings of $\leq_k(X_1, \ldots, X_n)$ are available, and how good are they? We do this by surveying the available encodings, introducing some new encodings, and presenting some experiments evaluating their performance.[1]

2 The Name of the Game

Here we define three properties of encodings: correctness and two properties relating to the power obtained by applying unit propagation to an encoding.

Let $\phi_k(\boldsymbol{X}, \boldsymbol{A})$ denote an encoding of $\leq_k(X_1, \ldots, X_n)$, where $\boldsymbol{X} = X_1, \ldots, X_n$ and $\boldsymbol{A}$ denotes the possibly empty set of new variables introduced by the encoding. We say that $\phi_k(\boldsymbol{X}, \boldsymbol{A})$ is a correct encoding if for every assignment α to $\boldsymbol{X}$, α satisfies $\leq_k(X_1, \ldots, X_n)$ if and only if α can be extended to an assignment that satisfies $\phi_k(\boldsymbol{X}, \boldsymbol{A})$.

We say that unit propagation enforces arc consistency on an encoding $\phi_k(\boldsymbol{X}, \boldsymbol{A})$ if for any partial assignment α to $\boldsymbol{X}$ the result of performing unit propagation on $\phi_k(\boldsymbol{X}, \boldsymbol{A})$ results in the generation of an empty clause (i.e., unsatisfiability is detected) or

- at most k variables in $\boldsymbol{X}$ are assigned *TRUE*, and
- if exactly k variables in $\boldsymbol{X}$ are assigned *TRUE* then the remaining $n - k$ variables are assigned *FALSE*.

If, in addition, whenever the empty clause α' is not generated and all the variables of $\boldsymbol{X}$ are assigned, all the variables of $\boldsymbol{A}$ are assigned, then we say that unit propagation enforces arc consistency$^+$ on the encoding.

3 An Inventory of Encodings

This section presents an inventory of SAT encodings of the $\leq_k(X_1, \ldots, X_n)$ constraint, some new, some old. The encodings are presented in the order in which they were invented. Table 1 summarises the methods and their properties.

3.1 Binomial Encoding

The most widely known encoding of $\leq_1(X_1, \ldots, X_n)$, often referred to in the literature as the pair-wise or naïve encoding, is $\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \neg X_i \vee \neg X_j$. The obvious generalisation of this method with binomial selection leads us to name this the 'binomial' encoding of $\leq_k(X_1, \ldots, X_n)$:

[1] We recommend that this paper is read in colour as plots of the experimental results are in colour.

Method	Origin	Clauses	New vars	AC
binomial	folklore	$\binom{n}{k+1}$	0	AC$^+$
binary ($k=1$)	Frisch et al. [7]	$O(n \log_2 n)$	$O(\log_2 n)$	AC$^+$
binary ($k \geq 2$)	this paper	$O(kn \log_2 n)$	$O(kn)$	none
sequential	Sinz [9]	$O(kn)$	$O(kn)$	AC$^+$
commander	$k=1$: Kwon & Klieber[8] $k \geq 2$: this paper	$\left(\binom{2k+2}{k+1} + \binom{2k+2}{k-1}\right) n/2$	$kn/2$	AC$^+$
product	$k=1$: Chen [3] $k \geq 2$: this paper	$(k+1)(n + O(kn^{1/(m+1)})$	$(k+1)n^{1/(k+1)}$	AC

Table 1: A summary of the encoding methods. The "new vars" column reports the number of new variables introduced by the encoding. The AC column indicates whether unit propagation on the encoding enforces arc consistency, arc consistency$^+$ or neither.

$$\bigwedge_{\substack{I \subseteq \{1,\ldots,n\},\\ \#I=k+1}} \bigvee_{i \in I} \neg X_i$$

This encoding introduces no new variables and comprises $\binom{n}{k+1}$ clauses each of size $k+1$.

Later we will have need to encode an $\geq_k$ constraint. The familiar encoding of $\geq_1$ produces one clause, $\bigvee_{i=1}^{n} v_i$. The obvious generalisation is again with binomial selection by taking $\geq_k$ in terms of $\leq_{n-k}$:

$$\geq_k(v_1, \ldots, v_n) \equiv \leq_{n-k}(\neg v_1, \ldots, \neg v_n)$$

3.2 Binary Encoding

At Most One The binary encoding of $\leq_1(X_1, \ldots, X_n)$ was originally introduced by Frisch et al. [7, 6]. The encoding introduces new variables $B_1, \ldots, B_{\lceil \log_2 n \rceil}$. It then associates with each X_i a unique bit string $s_i \in \{1,0\}^{\lceil \log_2 n \rceil}$. The binary encoding of $\leq_1(X_1, \ldots, X_n)$ is

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg X_i \vee \phi(i,j)$$

where $\phi(i,j)$ denotes B_j if the j^{th} bit of s_i is 1 and otherwise denotes $\neg B_j$.

In cases where n is not a power of 2, Frisch et al. [7] introduced a small optimisation that omits some clauses from the encoding. We shall not consider this here.

This encoding introduces $\lceil \log_2 n \rceil$ extra variables and uses $n\lceil \log_2 n \rceil$ clauses.

At Most *k* Here we generalise the binary encoding to handle the $\leq_k(X_1, \ldots, X_n)$ constraint. As before associate with each X_i a unique bit string $s_i \in \{1, 0\}^{\lceil \log_2 n \rceil}$. The encoding introduces new variables $B_{i,g}$ $(1 \leq i \leq k,\ 1 \leq g \leq \lceil \log_2 n \rceil)$, which are essentially k copies of the previous B variables. The binary encoding of $\leq_k(X_1, \ldots, X_n)$ is

$$\bigwedge_{i=1}^{n} \bigvee_{g=1}^{k} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg X_i \vee \phi(i, g, j)$$

where $\phi(i, g, j)$ denotes $B_{g,j}$ if the j^{th} bit of s_i is 1 and otherwise denotes $\neg B_{g,j}$.

This formula can be transformed to CNF by introducing new variables $T_{g,i}$ $(1 \leq g \leq k, 1 \leq i \leq n)$ and replacing the outer disjunction with

$$(\neg X_i \vee T_{1,i} \vee \cdots \vee T_{k,i}) \ \wedge \bigwedge_{g=1}^{k} \ \bigwedge_{j=1}^{\lceil \log_2 n \rceil} (\neg T_{g,i} \vee \phi(i, g, j))$$

This encoding introduces $k(n + \lceil \log_2 n \rceil)$ extra variables and uses n clauses of size $k + 1$ and $nk\lceil \log_2 n \rceil$ clauses of size 2.

When $k > 2$ this encoding has many symmetries. For example, a solution can be changed to another by permuting the values of the "registers" (and making the corresponding changes to the T variables.) This symmetry can be broken by using lexicographic ordering constraints to constrain the values of the registers to be in non-decreasing order. Furthermore, we can consistently break more symmetries by adding the constraint that if any register contains the value 0 then (at least) the first register does. Thus, $T_{2,1}, \ldots, T_{k,1}$ must all be false, and the clause $(X_1 \vee T_{1,1} \vee T_{2,1} \vee \cdots \vee T_{k,1})$ can replaced by $(X_1 \vee T_{1,1} \vee T_{2,1})$ and correspondingly the clauses of the form $(\neg T_{g,i} \vee \phi(i, g, j))$ can be dropped for all values of g other than one. (One could also eliminate $T_{1,1}$ but we shall not consider that since it is a special case and has very little consequence.) We can also add the constraints that if any register has the second value then (at least) one of the first two does, and so on. Each of these enables the clauses to be simplified as previously. Similar symmetry-breaking constraints and corresponding simplifications can be imposed at the other end.

We have conducted some experiments on the performance of this encoding, which suggest that the lexicographic ordering constraints slightly harm performance, so we shall not consider these further. The elimination of some T variables is clearly advantageous as it simplifies clauses, which also facilitates unit propagation. So the encoding we use is

$$\bigwedge_{i=1}^{n} \left((\neg X_i \vee \bigvee_{g=\max(1,k-n+i)}^{\min(i,k)} T_{g,i}) \ \wedge \bigwedge_{g=\max(1,k-n+i)}^{\min(i,k)} \ \bigwedge_{j=1}^{\lceil \log_2 n \rceil} (\neg T_{g,i} \vee \phi(i, g, j)) \right)$$

3.3 The Sequential Counter Encoding

Sinz [9] introduced an encoding of $\leq_k(X_1, \ldots, X_n)$ that works by encoding a circuit that sequentially counts the number of X_i that are true. For each $1 \leq$

$i \le n$ there is a register whose value is constrained to contain the number of $X_1, \ldots, X_i$ that are true. Each register maintains its count in base one and hence uses k bits to count to k. Thus the encoding introduces the new variables $R_{i,j}$, $1 \le i \le n,\ 1 \le j \le k$, where each $R_{i,j}$ represents the i^{th} bit of register j. The clauses of the encoding are as follows.

$$\bigwedge_{i=1}^{n-1} \neg X_i \vee R_{i,1} \tag{1}$$

$$\bigwedge_{j=2}^{k} \neg R_{1,j} \tag{2}$$

$$\bigwedge_{i=2}^{n-1} \bigwedge_{j=1}^{k} \neg R_{i-1,j} \vee R_{i,j} \tag{3}$$

$$\bigwedge_{i=2}^{n-1} \bigwedge_{j=2}^{k} \neg X_i \vee \neg R_{i-1,j-1} \vee R_{i,j} \tag{4}$$

$$\bigwedge_{i=1}^{n} \neg X_i \vee R_{i-1,k} \tag{5}$$

Formula (1) states that if X_i is true then the first bit of register i must be true. Formula (2) ensures that in the first register only the first bit can be true. Formulas (3) and (4) together constrain each register i $(1 < i < n)$ to contain the value of the previous register plus X_i. Finally (5) asserts that there can't be an overflow on any register as it would indicate that more than k variables are true.

As claimed by Sinz, this encoding introduces $k(n-1)$ new variables and comprises $2nk + 2n - 3k + 1$ clauses.

3.4 The Commander Encoding

At Most One Klieber and Kwon [8] introduced the commander encoding of $\le_1$, which works by recursively partitioning the set of variables into groups of some fixed size and encoding $\le_1$ over each group. Klieber and Kwon describe the algorithm as follows: partition the variables into groups of size s, denoting the groups $G_1, \ldots, G_g$, and introduce a 'commander' variable for each group, denoting these $c_1, \ldots, c_g$. Then:

1. Encode $\le_1$ for each group G_i using the binomial method.
2. If a commander variable is *true* then some variable in its corresponding group must be *true*:
$$\bigwedge_{i=1}^{g} \left(c_i \Rightarrow \bigvee_{v \in G_i} v \right) \tag{6}$$
3. If a commander variable is *false*, no variable in its corresponding group can be *true*:
$$\bigwedge_{i=1}^{g} \bigwedge_{v \in G_i} \neg c_i \Rightarrow \neg v \tag{7}$$

4. Encode $\leq_1$ on the set of all commander variables, e.g. with recursive application of this algorithm.

We now give a new, simpler description of the commander encoding for $\leq_1$, which is useful because it readily generalises to an encoding for $\leq_k$. First observe that (6) is an encoding of $\leq_1$ on the variables $G_i \cup \{\neg c_i\}$, for each group i. Now observe that the combination of the $\leq_1$ constraint of step 1 together with (7) is the binomial encoding of $\leq_1(G_i \cup \{\neg c_i\})$, for each group i. Thus the clauses of steps 1, 2 and 3 together encode an exactly-one constraint on the variables of $G_i \cup \{\neg c_i\}$ for each group i. Steps 1, 2, and 3 can be replaced with an encoding of the exactly-one constraint (using the binomial methods of $\leq_1$ and $\geq_1$), turning the algorithm into:

1. Encode $\geq_1$ and $\leq_1$ for each set $G_i \cup \neg c_i$ using the binomial method.
2. Encode $\leq_1$ on the commander variables, e.g. with recursive application of this algorithm.

If on each recursive application of the encoding the variables are partitioned into groups of of size g, Kwon and Klieber show that in the limit of large n the encoding produces

$$n \cdot \frac{(s+1)/2 + 1/s}{1 - 1/s} \quad = \quad n \cdot \frac{s^2 + s + 2}{2s - 2}$$

clauses.[2] Klieber and Kwon claim without proof that the fewest clauses are produced when $s = 3$, leading to $3.5n$ clauses and introducing $n/2$ commander variables.

At Most k Our generalisation follows from the new view of the encoding for $\leq_1$. Again, we partition the variables into $g > k$ sets $G_1, \ldots, G_g$. For each group G_i we introduce k commander variables, $c_{i,1}$ to $c_{i,k}$. The algorithm is then as follows:

1. Encode $\geq_k$ and $\leq_k$ for each set $(G_i \cup \{\neg c_{i,j} \mid j = 1..k\})$ using the binomial method.
2. Remove symmetrical solutions when less than k variables in a group are *true* by ordering the commander variables:
$$\bigwedge_{j=1}^{k-1} c_{i,j} \Rightarrow c_{i,j+1}$$
3. Encode $\leq_k$ on the commander variables, e.g. with recursive application of this algorithm.

The symmetry-breaking step is easy to add and yields an encoding which performs much better.

Proof of the encoding size proceeds in a similar vein to that presented by Klieber and Kwon for $\leq_1$.

[2] It is approximately equal to the number of clauses per group multiplied by the number of groups as n tends to infinity. For details of this result see [8].

Lemma 1. *Encoding $\leq_k$ with the generalised commander encoding on n variables and a group size s creates $n/(s-k)$ groups in the limit of large n.*

Proof. To begin with the n variables are partitioned into n/s groups. Each group has k commander variables introduced. This is repeated for the kn/s commanders. The total groups is therefore:

$$\sum_{i=1}^{\infty} \frac{k^{i-1}n}{s^i} \;=\; \frac{n}{s}\sum_{i=0}^{\infty}\left(\frac{k}{s}\right)^i \;=\; \frac{n}{s}\cdot\frac{1}{1-\left(\frac{k}{s}\right)} \;=\; \frac{n}{s-k} \qquad \square$$

Now consider the number of clauses constructed for each group. Each group has exactly-k encoded on it and its commanders. The binomial encoding of exactly-k produces $\binom{s+k}{k+1} + \binom{s+k}{k-1}$ clauses. This leads to a total number of clauses for the generalised commander encoding equal to:

$$\frac{n}{s-k}\left(\binom{s+k}{k+1} + \binom{s+k}{k-1}\right) \;=\; n\frac{(s^2+s+k^2+k)(s+k)!}{(s-k)(s+1)!(k+1)!}$$

We have disregarded the symmetry breaking step in this analysis but note that it requires an additional $k-1$ clauses for each group. To minimise the number of clauses produced we conjecture that s should be set to $k+2$.

3.5 Product Encoding

At Most One Chen [3] recently introduced a novel encoding of the $\leq_k$ constraint. He calls this the "product encoding" because it decomposes $\leq_1(X_1,\ldots,X_n)$ into two constraints, $\leq_1(Y_1,\ldots,Y_{p_1})$ and $\leq_1(Z_1,\ldots,Z_{p_2})$, where $p_1 \times p_2 \geq n$.

At Most *k* Here we generalise the product encoding of $\leq_1$ to to an encoding of $\leq_k(X_1,\ldots,X_n)$.

1. Choose natural numbers $p_1,\ldots p_{k+1}$ such that $p_1 \times \cdots \times p_{k+1} \geq n$.
2. For each X_i use a unique tuple $\boldsymbol{x_i} = \langle x_1,\ldots,x_{k+1}\rangle$, where $1 \leq x_j \leq p_j$, to refer to that variable. Let $\boldsymbol{X}$ be the set of all such tuples.
3. For all $\boldsymbol{x_i} \in \boldsymbol{X}$ let $\boldsymbol{x_i}/j$ be the tuple that results from removing the j^{th} element from $\boldsymbol{x_i}$.
4. For all $1 \leq d \leq k+1$ for all $\boldsymbol{y} \in \{\boldsymbol{x}/d | \boldsymbol{x} \in \boldsymbol{X}\}$ introduce a variable $A_{d,\boldsymbol{y}}$.

The product encoding of $\leq_k(X_1,\ldots,X_n)$ is

$$\bigwedge_{d=1}^{k+1}\left(\left(\bigwedge_{\boldsymbol{x}\in\boldsymbol{X}} \neg\boldsymbol{x} \vee A_{d,\boldsymbol{x}/d}\right) \wedge \;\leq_k(\{A_{d,\boldsymbol{x}/d} | \boldsymbol{x} \in \boldsymbol{X}\})\right)$$

One must ensure that the "recursive" use of $\leq_k$ in the definition is a constraint on strictly fewer than n variables. In certain cases this requires taking care in how the tuples are assigned to the variables. We will not discuss the issue in detail here but remark that the problem is avoided if the tuples that are assigned to variables include the following $k+2$ elements:

$$\begin{array}{l}\langle 1,1,\ldots,1\rangle \\ \langle 2,1,1,\ldots,1\rangle \;\; \langle 1,2,1,1,\ldots,1\rangle \;\; \langle 1,1,2,1,\ldots,1\rangle \;\; \cdots \;\; \langle 1,1,\ldots,1,2\rangle\end{array}$$

This scheme obviously does not work if $n = k + 1$. Indeed, Chen's original algorithm for $\leq_1$ does not work if $n = 2$. This is not a problem in practice because if $n = k + 1$ then the binomial method provides a simple constraint: $\neg X_1 \vee \cdots \vee \neg X_n$.

3.6 Other Encodings

In addition to the sequential counter encoding, Sinz [9] also introduced the parallel counter encoding, which recursively splits the X_i variables in two halves, counts the true variables in each half. The two counts, represented in base two, are then added. This encoding uses only $O(n)$ clauses but unit propagations does not enforce arc consistency.

The totalizer encoding of Bailleux and Boufkhad [1] can simultaneously handle at-most-k and at-least-j constraints. It uses $\Theta(n^2)$ clauses. Unit propagation enforces arc-consistency, but not in optimal time.

There are several pieces of work on translating pseudo-Boolean constraints, which generalise Boolean cardinality constraints, into SAT [10, 2, 5].

4 Performance Evaluation

This section evaluates the performance of five SAT encodings of the $\leq_k(X_1,\ldots,X_n)$ constraint, some fast, some slow. In particular, experiments totalling hundreds of hours of CPU time were conducted on five encodings: binomial, commander, binary, product and sequential. We tested each encoding in detecting the unsatisfiability of a generalisation of the well-known pigeonhole problem, both with and without symmetry-breaking constraints. We also conducted a controlled experiment designed to measure the speed with which variable assignments propagate to a solution.

Our implementation of the commander encoding always uses a group size $s = k + 2$ as per our conjecture in §3.4; that our conjecture holds for all k used in the experiments was verified for large s. The method is used recursively except that the binomial encoding is used if $n < 7$ (in this case the binomial encoding has fewer clauses) or $n \leq k + s$ (in this case the commander encoding does not decompose to a smaller subproblem).

Our implementation of the product encoding uses the method recursively except that the binomial encoding is used if $n < 7$ (in this case the binomial encoding has fewer clauses) or $n \leq k + 1$ (in this case the product encoding does not decompose to a smaller subproblem).

Our experiments were conducted using MiniSat 2 [4] (release 070721) and distributed over tens of machines with the same hardware: Intel Core 2 6600

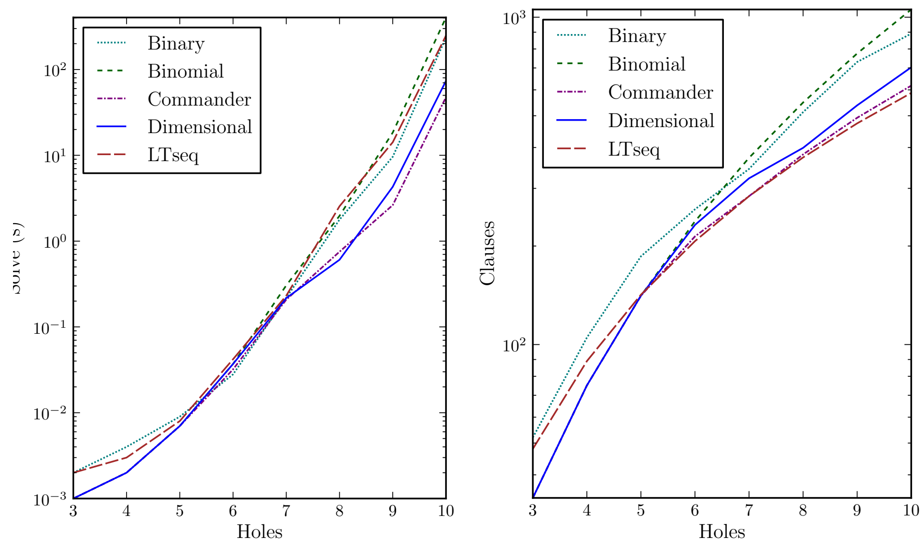

Fig. 1: Pigeonhole problem without symmetry breaking, one pigeon per hole

(2x2.40GHz) with 3 GiB of RAM. Experiments were stopped if the encoding time plus the solving time exceeded 10 minutes.

In the graphs that follow log scales are used on the axes as necessary to produce the most informative graph.

4.1 Experiments with the Pigeonhole Problem

The pigeonhole problem was used by Klieber and Kwon [8] to compare the performance of the commander encoding of $\leq_1$ against binomial and $\mathrm{LT}_{\mathrm{SEQ}}$. The goal of the problem is to prove that that p pigeons cannot be put into $h = p - 1$ holes, where each pigeon must be in at least one hole and no hole can contain more than one pigeon. Experiments are presented by Klieber and Kwon with and without symmetry broken by ordering the pigeons, which has a large impact on the size of the problems that can be solved.[3]

As well as conducting more thorough experiments with the problem on the different implementations of $\leq_1$, we generalise the pigeonhole problem to allow us to experiment on $\leq_k$: given that each hole can now fit k pigeons in it, the goal is to prove that $p = hk - 1$ pigeons be placed into h holes?

Without Symmetry Breaking With one pigeon per hole, Fig. 1 shows that binomial is the clear loser and commander the clear winner in terms of solve time for the original pigeonhole problem without symmetry breaking. Commander performs almost an order of magnitude better than binomial before the time limit

[3] Further details of the pigeonhole experiment and its implementation can be found in [8, § 3].

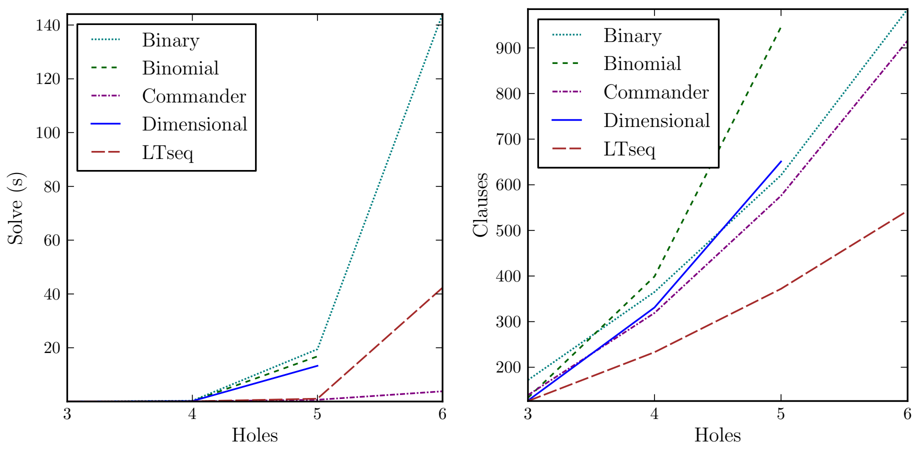

Fig. 2: Pigeonhole problem without symmetry breaking, two pigeons per hole

is reached. Product briefly overtakes commander – with such few variables the finer details of how each encoding performs near group or square-root boundaries is significant.

With multiple pigeons per hole commander continues to perform the best, with the effect more pronounced now with 2 pigeons per hole (Fig. 2). Binomial and product do not scale well, being just capable of solving at most 11 pigeons in 5 holes within the time limit. This trend continues up until our experiments conclude, where commander is the only encoding able to solve 16 pigeons in 3 holes and 21 pigeons in 4 holes (30,455 clauses, solve time 55s) before reaching its limit.

With Symmetry Breaking Adding symmetry-breaking constraints into the encoding enables the solution of instances an order of magnitude bigger for all five encoding methods. When there is one pigeon per hole, Fig. 3 shows that little variation exists between the product, commander, and $\mathrm{LT}_{\mathrm{SEQ}}$ encodings in this experiment, though commander is consistently marginally faster. The small bumps in the number of clauses produced by binary and product occur when the number of pigeons passes a power of 2 or a square number respectively.

Our results confirm previous results [8] that the commander encoding outperforms $\mathrm{LT}_{\mathrm{SEQ}}$ on the pigeonhole problem in the unordered case. Though the number of clauses generated for the $\mathrm{LT}_{\mathrm{SEQ}}$ encoding for the ordered case with 129 holes is similar to that reported by Klieber and Kwon, we have observed a vast difference in solve time. This may be due to slightly different implementations of the benchmark, or to Klieber and Kwon using a different version of MiniSat (the version is not reported in [8]).

When there are multiple pigeons per hole, solve times fluctuate for 2 pigeons per hole, though once smoothed the solve times show approximately the same

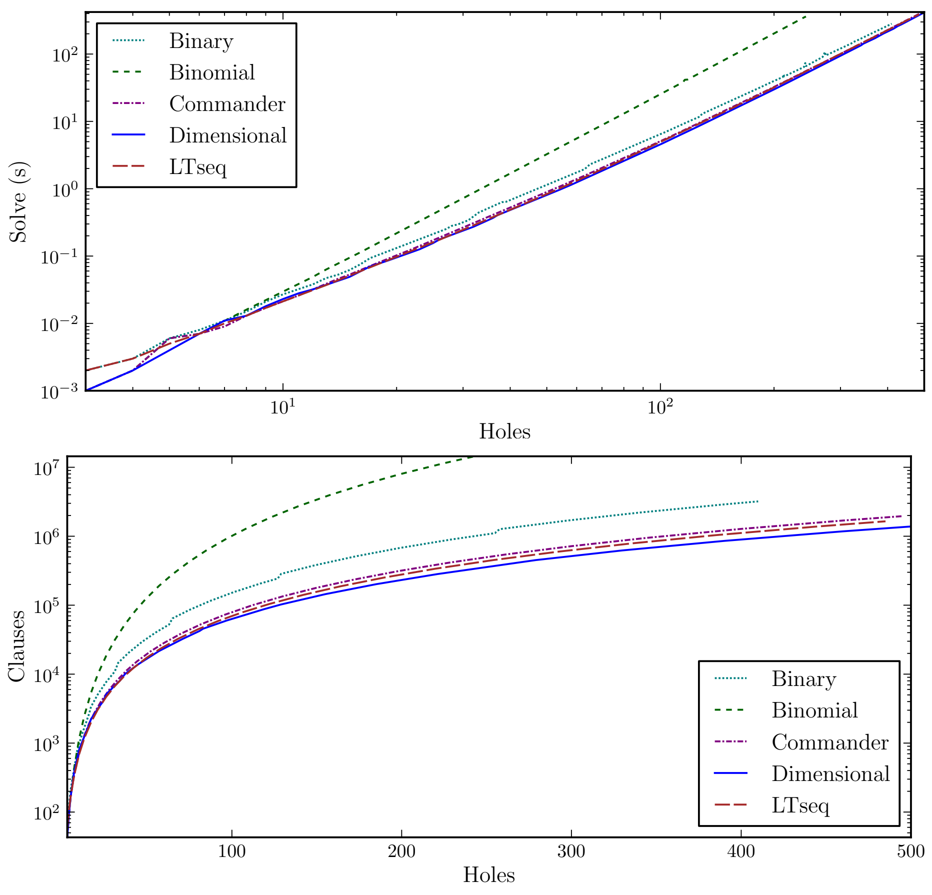

Fig. 3: Pigeonhole problem with symmetry breaking, one pigeon per hole

result as for one pigeon per hole, with binomial significantly worse than the other methods which perform similarly.

As the number of pigeons per hole increases past 2, however, binary and $\mathrm{LT}_{\mathrm{SEQ}}$ break away from product and commander, with the effect increasing in severity until our experiments end at five pigeons per hole (Fig. 4). Product begins to perform worse than even binomial around this point.

4.2 Experiments Measuring Propagation Speed

The goal of this benchmark is to determine how quickly an encoding propagates to detect unsatisfiability. To test this, large instances of $\leq_k$ are encoded, and then $k+1$ variables are randomly selected and set to *true* by appending the necessary $k+1$ clauses to the end of the expression. To push the encodings to their extreme, problem instances for $n < 1,000,000/k$ were tested for $k = 1..5, 10, 15, 20$. Pre-

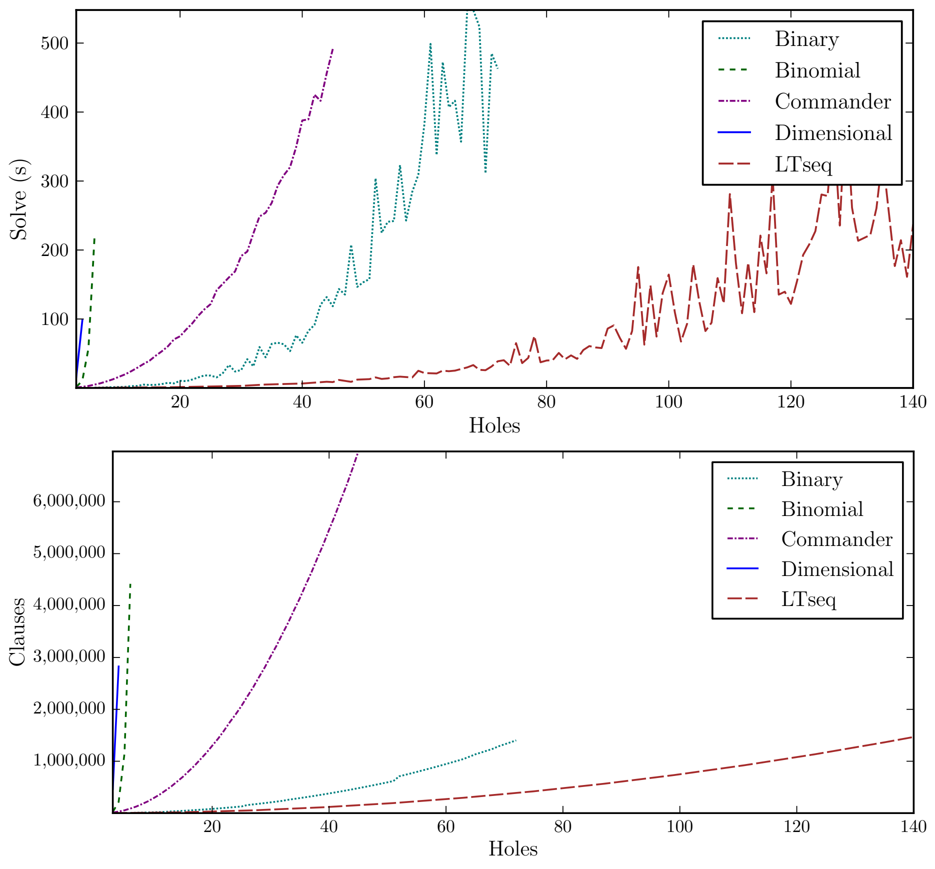

Fig. 4: Pigeonhole problem with symmetry breaking, five pigeons per hole

liminary experiments on the binomial encoding showed it to be significantly worse than the other methods, so it is not included in the comparison that follows.

As shown in Fig. 5, the product encoding propagates $\leq_1$ the quickest, detecting unsatisfiability between 1.5 and 2 times as quickly as its nearest competitor, LT_{SEQ}. Similarly, the product method is the fastest at propagating $\leq_2$ for all tested values of n.

However, as in the pigeonhole problem with symmetry breaking, the product encoding is not the fastest for $k \geq 3$. In all these test cases LT_{SEQ} is the fastest followed by binary, a trend which continues for all k with which we experimented. For example, Fig. 6 shows the results for $\leq_{10}$, where only LT_{SEQ} and binary could successfully run for n into the hundreds.

5 Conclusion

We have presented some existing methods and introduced some new methods of encoding the $\leq_1(X_1, \ldots, X_n)$ and $\leq_k(X_1, \ldots, X_n)$ constraints. We have examined

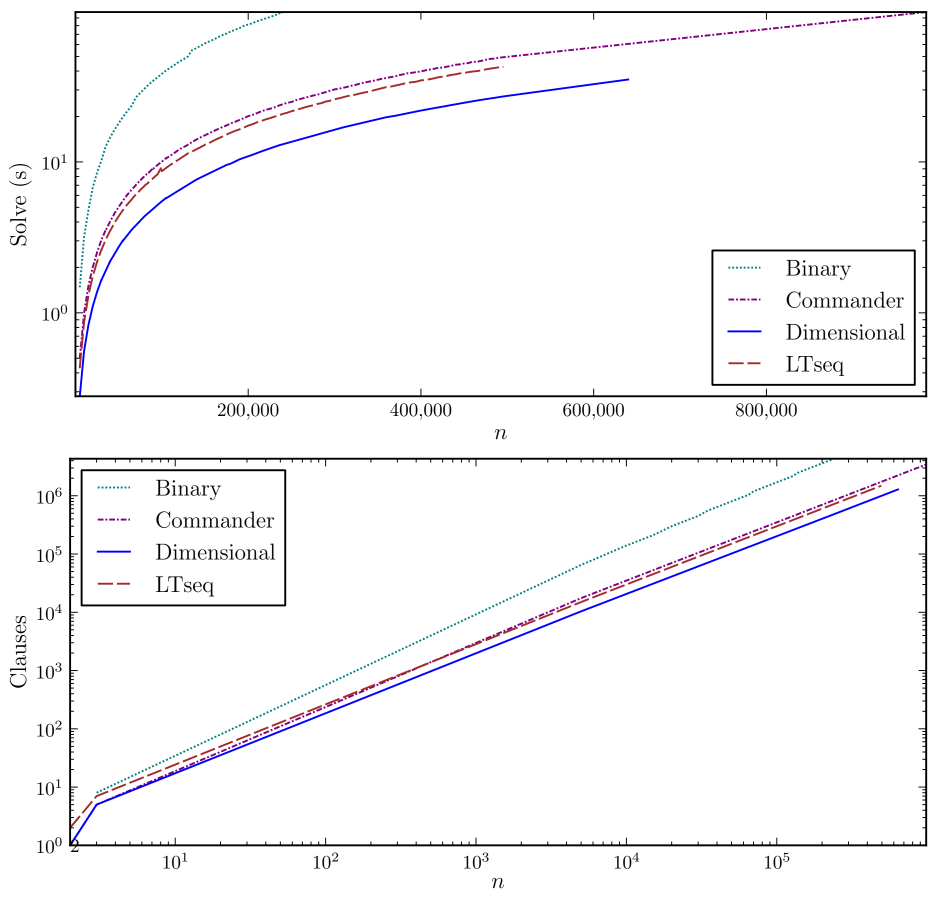

Fig. 5: Propagation problem, $\leq_1$

the size of the encodings and their ability to achieve arc consistency through unit propagation.

Experiments presented here compare the performance of the methods on three benchmarks. As expected from previously reported experiments and from it's large size, the binomial encoding performs poorly overall and is a reasonable performer only if $k = 1$ and n is small.

We expected the binary encoding to be competitive for $k = 1$ because of compact size, but to be poor for $k \geq 2$ because unit propagation does not enforce arc consistency. That turned out be roughly the case although, for pigeon hole problems with symmetry breaking and more than one pigeon per hole, the binary encoding was out-performed by only the sequential counter encoding.

The relative performance of the three recent encodings — sequential counter, commander and product — shows no clear pattern. In terms of pure propagation speed, the product encoding is slightly faster than the other two for $k = 1$, but for $k \geq 2$, the sequential counter encoding is the clear winner as k grows. In

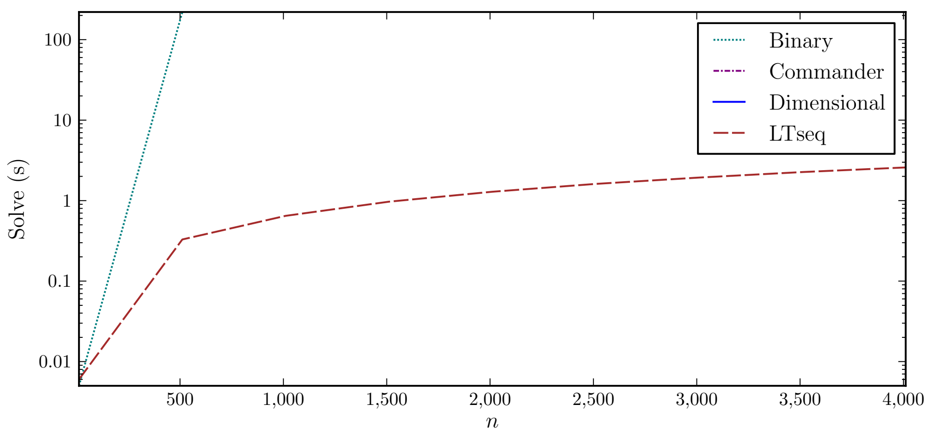

Fig. 6: Propagation problem, $\leq_{10}$, $n = 11, 511, 1011, ...$

solving the pigeon hole problem propagation speed is not all that matters as the encoding effects both the search heuristic and the conflict-directed clause learning. For one pigeon per hole, the three methods had roughly similar performance. However with more pigeons per hole the methods differ greatly. When symmetry-breaking constraints are used, the sequential counter encoding is clearly the best, the commander encoding is poor (worse than binary) and the product encoding is terrible. When symmetry-breaking constraints are not used, and hence only smaller problems could be solved, the commander encoding is the best followed by the sequential counter encoding.

In general, we noticed that smaller encodings tend to be faster encodings.

We consider this work to be the start of a larger effort and there are many possible directions to pursue. The conjecture that the commander encoding is smallest when the group size is $k+2$ could be proved or disproved. We think that further work on the binary encoding for $k \geq 2$ could yield improvements.

The experiments could be extended by considering more encoding methods, more benchmark problems, and other SAT solvers. In addition, further experiments on propagation times could measure the time it takes to propagate a $\leq_k$ constraint when exactly k variables are set to *TRUE*. Other experiments could be used to understand better *why* some encodings perform better than others.

References

1. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.
2. Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-Boolean constraints into CNF. In *SAT '09: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, pages 181–194, Berlin, Heidelberg, 2009. Springer-Verlag.

3. Jing-Chao Chen. A new SAT encoding of the at-most-one constraint. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, September 2010.
4. N. Eén and N. Sörensson. Minisat v2.0 (Beta). http://fmv.jku.at/sat-race-2006/descriptions/27-minisat2.pdf, 2006.
5. Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.
6. Alan M. Frisch and Timothy J. Peugniez. Solving non-Boolean satisfiability problems with stochastic local search. In *Proc. of the Seventeenth Int. Joint Conf. on Artificial Intelligence*, pages 282–288, Seattle, Washington, August 2001.
7. Alan M. Frisch, Timothy J. Peugniez, Anthony J. Doggett, and Peter Nightingale. Solving non-Boolean satisfiability problems with stochastic local search: A study of encodings. *Journal of Automated Reasoning*, 35:143–179, 2005.
8. Will Klieber and Gihwon Kwon. Efficient CNF encoding for selecting 1 from N objects. In *Fourth Workshop on Constraints in Formal Verification (CFV '07)*, July 2007.
9. Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, Sitges, Spain, October 2005.
10. Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.*, 68(2):63–69, 1998.