Variable Dependency in Local Search: Prevention Is Better Than Cure

Steven Prestwich

Cork Constraint Computation Centre
Department of Computer Science, University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract. Local search achieves good results on a variety of SAT problems and often scales up better than backtrack search. But despite recent advances in local search heuristics it has failed to solve some structured problems, while backtrack search has advanced greatly on such problems. We conjecture that current modelling practices are unintentionally biased in favour of solution by backtrack search. To test this conjecture we remodel two problems whose large instances have long resisted solution by local search: parity learning and Towers of Hanoi as STRIPS planning. By reducing variable dependencies and using other techniques we boost local search performance by several orders of magnitude in both cases, and we can now solve 32-bit and 6-disk instances for the first time using a standard SAT local search algorithm.

1 Introduction

Local search is often more scalable than backtrack search, and in some areas of combinatorial optimisation is the only practical way of obtaining good solutions. Yet it currently has the reputation of being inferior to DPLL (the Davis-Putnam-Logemann-Loveland SAT backtracking algorithm) on structured SAT instances and only good for random problems. Nevertheless, it cannot be denied that local search generally performs badly on problems classed as *industrial* in SAT solver competitions, and the winners are all DPLL variants (see for example [35]). Hybridising local search with unit propagation [11,24] or explicitly handling variable dependencies [15,23] helps, but DPLL is still unbeaten on these problems. Improving local search on structured problems would have many practical applications, perhaps solving larger instances of real-world applications than is currently possible, but we cannot begin to improve it until we understand the cause of its poor performance.

We conjecture that *current SAT-encodings are unintentionally designed to favour DPLL*, and that this explains the poor ranking of local search algorithms in solver competitions. For example, in SAT modelling we often eliminate symmetrical solutions. But when applying local search, symmetry appear to be harmless or even helpful, and eliminating it can adversely affect performance [25,27]. Moreover, SAT encodings of specific constraints have been explored by

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 107–120, 2007.

[1,2,7,8,9,13] with the aim of improving the consistency reasoning of unit propagation in DPLL. But unit propagation is not used in most local search algorithms so consistency reasoning is not necessarily relevant. In fact it was shown in [26] that the *ladder* structure introduced in some of these encodings has a harmful effect on local search performance. We also make a more specific conjecture: that the model feature to blame for local search's poor performance is in many cases *dependent variables*. These are known to slow down local search [15], and when they form long chains they may cause local search to take polynomial or exponential time to propagate effects [24,33].

Our conjectures can be tested empirically by devising new SAT encodings with reduced variable dependency, and comparing local search on the old and new encodings. We do this for two problems whose large instances have so far resisted solution by local search. In Section 2 we reformulate the *parity learning* problem to avoid dependency chains, via new SAT-encodings of parity and cardinality constraints, and show that local search can solve 32-bit instances. In Section 3 we reformulate the *Towers of Hanoi* problem expressed as a planning problem, breaking up long dependency chains into short ones and artificially increasing solution density, and show that local search can solve the 6-disk instance. Both results are firsts for an off-the-shelf SAT local search algorithm.[1] demonstrating the power of the remodelling approach. Section 4 discusses further applications and concludes the paper.

All our experiments are performed on a 733 MHz Pentium II under Linux. We use only one local search algorithm: RSAPS [12] implemented in the UBCSAT system [30]. RSAPS is a state-of-the-art dynamic local search algorithm, and was chosen after preliminary experiments indicated that it was one of the best algorithms for these problem. It also has the advantage that its default parameter settings give good results over a wide range of SAT problems, so we did not need to tune them.

2 Minimal Disagreement Parity Learning

This problem description is taken from [4]. Given vectors $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ $(i = 1 \ldots m)$ with each $x_{ij} \in \{0, 1\}$, a vector $\mathbf{y} = (y_1, \ldots, y_m)$ and an error tolerance integer k. Find a vector $\mathbf{a} = (a_1, \ldots, a_n)$ such that $|\{i : \text{parity}(\mathbf{a} \cdot \mathbf{x}_i) \neq y_i\}| \leq k$. To make hard instances set $m = 2n$ and $k = 7n/8$. n is referred to as the number of bits, and 32-bit instances of this problem have proved intractable for both DPLL and local search. They have been solved by paying special attention to the parity constraints, either by transforming many of them away in a preprocessing phase or by augmenting DPLL with equivalence reasoning [2,3,19,32].

Until recently local search has never solved 32-bit instances [16]. A special version of the DLM local search algorithm was created for these and other very hard benchmarks [34] but fails on 32-bit instances, as does a more recent algorithm [22]. It was not until this year that an extended local search algorithm solved them by exploiting knowledge about variable dependencies [23]. Why is

[1] But a recent extended algorithm [23] has solved 32-bit problems — see Section 2.4.

this problem so hard for local search? An explanation suggested by [16] is the existence of local minima in which a small subset of clauses is never satisfied simultaneously. We conjecture that chains of dependent variables are the culprit (this does not necessarily contradict the explanation of [16]) and define a new encoding of parity constraints that contains no such chains.

What we shall call the *standard encoding* is that used in [4] and SATLIB[2] parX-Y-c instances (which are improved versions of the much harder parX-Y instances) and described in [4]. It contains three families of clauses: the first calculates the parities of $\mathbf{a} \cdot \mathbf{x}_i$; the second computes disagreements in parities; the third encodes a cardinality constraint to limit the disagreements (n is set to a power of 2 so that cardinality is easy to enforce). We start with a slightly different model of parity learning that allows experimentation with different encodings of parity constraints. If we can find improved encodings then these may be useful when modelling other problems.

2.1 A Constraint-Based Model

Define variables A_i to contain the solution and P_j to denote parities. Force each scalar product $\mathbf{a} \cdot \mathbf{x}_j$ to have parity P_j:

$$P_j \equiv \bigoplus_{i \in \tau_j} A_i$$

where $\tau_j = \{i \,|\, x_{ij} = T\}$. Then at most k of m literals are true:

$$\mathrm{LE}(k, \pi_1, \ldots, \pi_m)$$

where literal π_j is $\bar{P}_j$ if $y_j = T$ and P_j if $y_j = F$. By using a cardinality constraint we can encode parity learning instances of any size, not just with n a power of 2. We now discuss encodings for the $\bigoplus$ and LE constraints.

2.2 Encoding Parity Constraints

It is possible to SAT-encode a parity constraint $\bigoplus_{i=1}^{p} P_i = k$ simply by enumerating all possible combinations of P_i truth values, together with their parity k. But this creates exponentially many clauses and is only reasonable for small constraints. We shall call it the *exponential encoding*. A more practical method due to [19] decomposes the constraint by introducing new variables:

$$P_1 \oplus z_1 \equiv k \qquad P_2 \oplus z_2 \equiv z_1 \qquad \ldots \qquad P_{p-3} \oplus z_{p-3} \equiv z_{p-2} \qquad P_p \equiv z_{p-1}$$

The remaining binary and ternary constraints are then expanded via the exponential encoding. We shall call this the *linear encoding*. It has $O(p)$ new variables and literals and is essentially the method used to encode the parities of $\mathbf{a} \cdot \mathbf{x}_i$ in the standard encoding of parity learning.

[2] http://www.cs.ubc.ca/~hoos/SATLIB/

A drawback with the linear encoding is that it creates a long chain of variable dependencies, which has been shown to be bad for local search performance [24,33]. An obvious alternative is a divide-and-conquer approach: bisect the constraint, solve the two subproblems, and merge the two results by a ternary constraint. That is, express $\bigoplus_{i=1}^{p} P_i = k$ as $\bigoplus_{i=1}^{p/2} P_i = k_1$, $\bigoplus_{i=p/2+1}^{p} P_i = k_2$ and $k = k_1 \oplus k_2$, and recursively decompose until reaching a base case of size 2 or 3. All binary and ternary parity constraints are expanded into clauses via the exponential encoding. We shall call this the *bisection encoding*. It replaces the chains of dependency of length p by a tree of depth $\log p$.

We also try a third technique. Decompose $\bigoplus_{i=1}^{p} P_i = k$ into

$$\bigoplus_{i=1}^{\alpha} P_i \equiv k_1 \qquad \bigoplus_{i=\alpha+1}^{2\alpha} P_i \equiv k_2 \qquad \ldots \qquad \bigoplus_{i=p-\alpha+1}^{p} P_i \equiv k_\beta \qquad \text{and} \qquad \bigoplus_{i=1}^{\beta} k_i \equiv k$$

where $\beta = \lceil p/\alpha \rceil$ and the tree branching factor α is a number in the range $1 < \alpha < p$. Expand the $\beta + 1$ parity constraints into clauses via the exponential encoding. This creates $O(\beta)$ new variables and $O(\beta 2^\alpha + 2^\beta)$ literals. This still gives a tree of variable dependencies but only of depth 2. We exponentially increase the number of clauses but the number is quite manageable for (say) $p \leq 100$. For larger p it can use a slightly less shallow tree of depth (say) 3 or 4. We shall call this the *shallow encoding* and use trees of depth 2 in our experiments.

2.3 Encoding Cardinality Constraints

We use a new SAT encoding of a cardinality constraints $\mathrm{LE}(k, \pi_1, \ldots, \pi_m)$ that places an upper bound on the number of literals in a given set that are allowed to be true. We use this encoding mainly for convenience (it is very easy to implement).

First consider the special case where the upper bound is 1, so that we have an at-most-one (AMO) constraint. Define new Boolean variables b_k where $k = 1 \ldots \lceil \log_2 m \rceil$. Add clauses

$$\bar{\pi}_i \vee b_k \text{ [or } \bar{b}_k\text{]}$$

if bit k of the binary representation of $i - 1$ is 1 [or 0], where $k = 1 \ldots \lceil \log_2 m \rceil$. This encoding has $O(\log m)$ new variables and $O(m \log m)$ binary clauses. This *bitwise* encoding was defined in [26] and shown to work well with local search; other known encodings either have higher space complexity or interact poorly with local search (because of chains of dependent variables).

Now suppose we want to prevent more than k of literals $\pi_1 \ldots \pi_m$ from being true. Suppose we have k bins. Define $x_{ij} = T$ if π_i is placed in bin j. Every true π_i must be placed in a bin:

$$\pi_i \rightarrow \left(\bigvee_j x_{ij} \right)$$

and no more than one π_i may be placed in a bin:

$$\mathrm{AMO}_i(x_{ij})$$

using the bitwise encoding. Of course this encoding introduces a great deal of symmetry, as the π_i can be permuted among the bins. Here we invoke [25,27]: symmetry does not necessarily harm local search performance, and may even improve it.

This cardinality encoding has already been used in [26] to solve clique problems by SAT local search, but the bin structure was entangled in the clique model and a cardinality encoding was not explicitly described. In future work we will compare it with other known encodings such as that of [2], which create trees of dependent variables that may slow down local search.

2.4 Experiments

We do not have access to the original parity learning instances, only their standard SAT encodings. Instead we generate 30 random instances (using the method described in [4]) of each required size and take the median of 30 runs of RSAPS, one run per instance (except for the expensive 28- and 32-bit instances which use only 10 runs). The aim is to estimate typical local search performance on a typical problem.

par8-X-c

X	flips	secs
1	1,144	0.0014
2	1,518	0.0017
3	3,060	0.0034
4	1,477	0.0018
5	2,339	0.0028

par16-X-c

X	flips	secs
1	13,377,611	20
2	16,533,206	24
3	12,863,749	19
4	8,601,612	13
5	13,505,657	20

par32-X-c

X	flips	secs
1	—	—
2	—	—
3	—	—
4	—	—
5	—	—

Fig. 1. Local search results on SATLIB parity learning instances

Median results for the SATLIB instances are shown in Figure 1 and our encoding results are shown in Figure 2. An entry "—" denotes that more than 1 billion flips are needed while "?" denotes experiments not done. The 8- and 16-bit results for the linear encoding are similar to those for the standard encoding (perhaps slightly better). This is a good sanity check: the linear encoding has similar characteristics to the standard encoding, so any major improvements we obtain will be due to improvements in parity constraint encoding. Extrapolating by applying linear regression to the logarithms of the four flip results, and using a measured flip rate of 373,134 flips per second for 32-bit instances under the linear encoding, we expect RSAPS to take approximately 20 trillion flips and 2 years to solve them: it is unsurprising that no successes have been reported with the standard encoding.

The bisection encoding is hardly better than the linear encoding, which is a surprise: the failure of local search does not seem to be caused purely by the length of the chains of dependency, and trees of dependencies may be almost

flips needed to find a solution

			shallow				
n	linear	bisecting	$\beta = 6$	$\beta = 8$	$\beta = 10$	$\beta = 12$	$\beta = 14$
8	2,517	2,836	1,119	747	904	955	891
12	157,433	83,708	10,089	4,494	4,929	3,905	2,265
16	7,139,810	5,326,518	599,662	59,051	23,630	18,457	25,038
20	223,090,992	156,378,976	11,381,251	3,283,580	460,165	167,936	173,826
24	—	—	282,226,496	35,025,380	16,078,792	3,647,108	?
28	—	—	?	251,928,288	131,614,608	?	?
32	—	—	?	?	1,454,529,796	?	?

seconds needed to find a solution

			shallow				
n	linear	bisecting	$\beta = 6$	$\beta = 8$	$\beta = 10$	$\beta = 12$	$\beta = 14$
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.02	0.12	0.02	0.02	0.03	0.02	0.01
16	11	8.5	1.2	0.22	0.47	0.83	1.03
20	408	297	29	13	11	27	69
24	—	—	788	149	272	693	?
28	—	—	?	2,386	3,640	?	?
32	—	—	?	?	49,633	?	?

Fig. 2. Local search results on new parity learning encodings

as harmful. More research is clearly required into what types of structure are bad for local search performance, but if trees of dependency are harmful then standard cardinality encodings seem to be unsuitable for local search.

The shallow encoding gives much better results and is able to solve 32-bit instances in a few hours. The choice of β has a large effect on performance: roughly speaking, the greater the value of β the fewer flips are required to solve the problems; but higher β also means larger models and thus lower flip rates, so in terms of CPU time there is a trade-off. A reasonable value for these instances is $\beta = 10$. We also experimented on 16-bit instances with two other automated local search algorithms: AdaptNovelty+ was also faster on the new ($\beta = 10$) encoding than on the standard encoding, though less so than RSAPS, while VW behaved similarly on both. In contrast, the DPLL algorithms ZChaff, SATO and SATZ were all slowed down by a factor of at least 100 on the new encoding. Modelling for local search is clearly distinct from modelling for backtrack search.

Though our local search results are vastly improved we have not yet matched the best DPLL performance. But local search performance might be further improved by using similar techniques: directly handling parity constraints during search, or preprocessing the problems to eliminate them. Recently Pham et al. [23] solved the *standard* encodings of the 32-bit instances, using a new non-CNF local search algorithm that exploits problem structure analysis. Their execution times are similar to ours (though on an unspecified machine) and they use an order of magnitude fewer flips than we do on 16-bit instances (but do not provide flip figures for 32-bit instances).

3 Towers of Hanoi as STRIPS Planning

Planning problems expressed in the STRIPS language have been SAT-encoded many times [5,14,17,18]. SAT-based planning has achieved state-of-the-art results on STRIPS [6] planning problems and is one of the success stories of SAT research. We study the Towers of Hanoi (ToH) problem modelled as a STRIPS planning problem. ToH is perhaps not a very interesting problem in itself, and solving it via STRIPS and SAT is certainly not the best approach. Its interest lies in the fact that ToH makes very hard planning problems for local search: harder than the Blocks World instances, which are in turn harder than the logistics instances [28]. In the 2002 SAT solver competition, no local search algorithm solved the ToH problems with 4, 5 and 6 disks, while the BerkMin backtracker solved the 6-disk problem in 2551 sec. A special version of the DLM local search algorithm was created for these and other very hard benchmarks, and solved 4 disks in almost 1 billion flips and over 2 hours [34].

Why is ToH so hard for local search? It may be because it has only one solution [28]. We believe that the explanation is a combination of low solution density and the chain-like structure of dependent variables in SAT-encoded planning problems. We modify both the STRIPs model and the SAT-encoding, increasing solution density and breaking up variable chains, to obtain huge improvements in local search performance. First we define what we shall refer to as a *standard approach* to SAT-encoding ToH as a STRIPS planning problem. There is of course no single standard approach but ours is based on published techniques.

3.1 ToH as STRIPS

The ToH problem consists of P pegs (or towers) and D disks of different sizes; usually $P = 3$. All d disks are initially on the first peg. A solution is a plan that moves all disks to the third peg with the help of the second peg so that (i) only one disk can be moved at a time; (ii) only the disk on top can be moved; (iii) no disk can be put onto a smaller disk. There is always a plan with $2^D - 1$ steps.

We have two fluents $\mathrm{on}(d, dp)$ and $\mathrm{clear}(dp)$, where d denotes a disk and dp either a disk or a peg. We also have an action $\mathrm{move}(d, dp, dp')$ that moves d from dp to dp' with preconditions $\{\mathrm{clear}(d), \mathrm{clear}(dp'), \mathrm{on}(d, dp)\}$, add effects $\{\mathrm{on}(d, dp'), \mathrm{clear}(dp))\}$ and delete effects $\{\mathrm{on}(d, dp), \mathrm{clear}(dp')\}$. In the initial state fluents $\{\mathrm{on}(disk_1, disk_2), \ldots, \mathrm{on}(disk_{D-1}, disk_D), \mathrm{on}(disk_D, peg_1), \mathrm{clear}(disk_1), \mathrm{clear}(peg_2), \ldots, \mathrm{clear}(peg_P)\}$ are true and all others are false. In the goal state fluents $\{\mathrm{on}(disk_1, disk_2), \ldots, \mathrm{on}(disk_{D-1}, disk_D), \mathrm{on}(disk_D, peg_P)\}$ are true and all others have unspecified truth values.

3.2 STRIPS as SAT

We start from an encoding similar to those used in [14,17,18]. First we set an upper bound on the plan length of discrete times $t = 0 \ldots N-1$. Define two sets of variables: τ_{pt} ($t = 0 \ldots N$ where N denotes the state after the last action) and predicate p, where $\tau_{pt} = T$ iff p is true at the start of time t; and α_{at} ($t = 0 \ldots N$)

where $\alpha_{at} = T$ iff action a occurs at time t. The clauses are as follows. Exclusion axioms restrict the plan to be linear, in which at most one action occurs at any time t:

$$\overline{\alpha_{at}} \vee \overline{\alpha_{a't}}$$

We do not force actions to occur at every time, which creates additional solutions (if the plan length is overestimated) that may help local search. Actions imply preconditions:

$$\alpha_{at} \rightarrow \tau_{pt} \quad \text{or} \quad \alpha_{at} \rightarrow \overline{\tau_{pt}}$$

and effects:

$$\alpha_{at} \rightarrow \tau_{p\,t+1} \quad \text{or} \quad \alpha_{at} \rightarrow \overline{\tau_{p\,t+1}}$$

Frame axioms preserve fluents that are unaffected by actions. These may be in either *classical* or *explanatory* form. We use the explanatory form [10] which is more compact [5,14] and also obviates the need for every time slot to contain an action:

$$\tau_{pt} \wedge \overline{\tau_{p\,t+1}} \rightarrow \left(\bigvee_{a \in E_{\bar{p}}} \alpha_{at} \right) \quad \text{or} \quad \overline{\tau_{pt}} \wedge \tau_{p\,t+1} \rightarrow \left(\bigvee_{a \in E_p} \alpha_{at} \right)$$

where E_p ($E_{\bar{p}}$) denotes the set of actions with add (delete) effect p. The initial state is represented by unit clauses:

$$\tau_{p0} \quad \text{or} \quad \overline{\tau_{p0}}$$

as is the goal state:

$$\tau_{pN} \quad \text{or} \quad \overline{\tau_{pN}}$$

We now try to improve the standard approach in several ways.

3.3 Exploiting Domain Knowledge

ToH has been modelled in the same way as a Blocks World problem, but its special form allows a simpler STRIPS model. We retain the *move* operator and *on* predicate as before, but drop the *clear* predicate and only specify which peg a disk is on, not which disk or peg. The action move(d, x, y, t) now has preconditions $\{\text{on}(d, x),\ \neg\text{on}(1, x, t),\ \ldots,\ \neg\text{on}(d-1, x, t),\ \neg\text{on}(1, y, t),\ \ldots,\ \neg\text{on}(d-1, y, t)\}$ for all $d' < d$, add effects $\{\text{on}(d, y, t+1)\}$ and delete effects $\{\text{on}(d, x, t+1)\}$. In the initial state fluents $\{\text{on}(1, 1),\ \ldots,\ \text{on}(D, 1)\}$ are true and all others are false, while in the goal state $\{\text{on}(1, P),\ \ldots,\ \text{on}(D, P)\}$ are true and the others unspecified.

Besides having fewer predicates, this model has fewer actions because each disk can only be on P pegs instead of $D + P$ disks or pegs. In effect we are using the domain knowledge that disks are stacked in decreasing order of size; this trick would not work on general Blocks World problems.

3.4 Superparallelism

An important technique in planning is the use of *parallel plans* in which more than one action may occur at a given time. Besides being more appropriate for some applications, this allows the plan length to be shorter and thus the SAT problem to be smaller. As the size of the SAT problem can be a bottleneck for real-world planning problems, this benefits both DPLL and local search algorithms. Parallelism may have another advantage for local search: it increases the solution density of the SAT problem. This is because any linear plan has multiple representations as a parallel plan, typically an exponential number of them. It has been shown that increasing the solution density of a SAT problem can boost local search performance (though this is not guaranteed).

Unfortunately there is no natural parallelism in ToH. But we can allow some actions to be performed in parallel in the new model, by removing some exclusion axioms:

- Allow (say) disk 1 to move from peg 1 to peg 2, and disk 2 to move from peg 3 to peg 2, at the same time: this can be uniquely transformed to: move the larger disk to peg 2 then the smaller one.
- Allow (say) disk 1 to move from peg 1 to peg 2, and disk 2 to move from peg 2 to peg 3, at the same time. This can be uniquely transformed to: move disk 2 then disk 1. There is no danger of an illegal cycle of three moves as the preconditions will prevent one of the disks from moving onto a peg with a smaller one.

We shall call this *superparallelism* because it adds parallelism beyond any that is naturally present in the model (in this case none). Superparallel moves are illegal and must be transformed away after finding a plan. A drawback with superparallelism is that we can no longer force the search to find optimal plans. Even if we reduce the number of times to the smallest possible value, after transformation we may obtain a very suboptimal linear plan. But it may be a useful technique for applications in which any feasible plan will do, or for quickly obtaining an initial plan for subsequent improvement. Another possibility is to place an upper limit on the total number of actions via a SAT-encoded cardinality constraint.

It is not possible to force parallelism in the standard STRIPS model of Blocks World by dropping exclusion axioms, because performing any two actions at the same time would lead to an inconsistent state. However, it would be possible to define new actions that move more than one disk at a time; we leave this for future work.

3.5 Long-Range Dependencies

The encoding of Section 3.2 has a potential drawback for local search: the frame axioms create chains of dependent variables $\tau_{pt} \ldots \tau_{pt'}$ for each p and pair t, t'. As noted above, dependent variables are known to be a major source of slowdown in local search [15], especially when they occur in chains [24,33].

At first glance there seems to be no way to avoid these chains, as they are a property of the problem itself and not the encoding. However, it is possible to break up the chain structure by using the method of [33]: add implied clauses to cause *long-range dependencies* between times further apart than 1 unit. The clauses we add are a generalisation of explanatory frame axioms to time differences ≥ 1:

$$\tau_{pt} \wedge \overline{\tau_{pt'}} \rightarrow \left[\bigvee_{t''=t}^{t'-1} \left(\bigvee_{a \in E_{\bar{p}}} \alpha_{at''} \right) \right] \quad \text{or} \quad \overline{\tau_{pt}} \wedge \tau_{pt'} \rightarrow \left[\bigvee_{t''=t}^{t'-1} \left(\bigvee_{a \in E_p} \alpha_{at''} \right) \right]$$

where $t' > t$. We shall call these *generalised explanatory frame (GEF) axioms*. The usual explanatory frame axioms are given by the case $t' = t + 1$.

Adding all GEF axioms increases the space complexity, but we can add a randomly-chosen subset of them (but including the usual explanatory frame axioms), by analogy with [33] who found that adding a relatively small number of implied clauses gave optimal improvement.

Unlike the fixed-length added clauses of [33] ours grow with n, so their effect on search time may be inferior. We could reduce their length by defining new variables $\epsilon_{pt} \rightarrow \left(\bigvee_{a \in E_p} \alpha_{at} \right)$ where $\epsilon_{pt} = T$ only if an action with effect p occurs at time t. This allows us to simplify the GEF axioms but in experiments it made the problems harder to solve.

3.6 Implied Clauses

Besides the GEF axioms we add another set of implied clauses: exclusion axioms corresponding to two disks making the same move. This can never occur because the larger disk's preconditions are unsatisfied if the smaller one is on the same peg, so these clauses are redundant.

3.7 Experiments

We compare four models: the standard model, the compact model (using special domain knowledge), the compact model with parallelism, and the compact model with parallelism and GEF axioms. All models use the implied constraints described above. In experiments 5% was approximately the best proportion of randomly-selected GEF axioms, which is less than the 20% figure of [33].[3] All results are medians of 30 runs. In each case the number of times was set to $2^D - 1$, the optimum for a linear ToH plan. The results are shown in Figure 3, with "—" denoting that RSAPS failed to find a solution after 1 billion flips.

The hardness of ToH grows extremely rapidly with D in all models. The compact model gives much better results than the standard model, and parallelism and GEF axioms greatly improve performance. By combining several modelling

[3] Actually, this was for the largest instances, and a higher percentage was better for smaller instances, possibly indicating that the optimum number of GEF axioms is less than linear in the problem size.

local moves (flips)				
D	standard	compact	parallel	GEF
3	38,271	3,730	546	410
4	—	2,757,378	4,866	5,985
5	—	—	532,488	51,453
6	—	—	—	40,163,929

execution time (seconds)				
D	standard	compact	parallel	GEF
3	0.096	0.0058	0.0010	0.0010
4	—	5.8	0.0093	0.017
5	—	—	1.8	0.30
6	—	—	—	980

Fig. 3. Results of experiments

techniques we have obtained the best-reported SAT local search results for 4, 5 and 6 disks, and they are comparable to the best DPLL results (though at the price of reducing plan quality through superparallelism). We also added GEF axioms to the standard model but were still unable to solve 4 disks. We expect further improvements by using the well-known techniques of operator splitting (which reduces the space complexity of SAT-encoded planning problems) and preprocessing by unit propagation and subsumption.

In further local search experiments, AdaptNovelty+ and VW were faster on compact model than on the standard model, and even faster with superparallelism. AdaptNovelty+ was faster with GEF axioms, while VW was hardly affected (apart from the overhead of maintaining the additional clauses). The DPLL algorithms ZChaff, SATZ and SATO were all improved by the compact encoding, ZChaff was faster with superparallelism while SATZ and SATO were slower, and SATO was faster with GEF axioms while ZChaff and SATZ were slower. Thus the compact encoding helps all the algorithms, while the other techniques mostly help local search but have an erratic effect on DPLL. Again, modelling for local search is shown to be distinct from modelling for DPLL.

4 Conclusion

We showed that local search performance on two hard problems can be boosted by several orders of magnitude, simply by remodelling the problems. The aims of our remodelling were to reduce variable dependency chains and to increase solution density, and we believe that these aims should be borne in mind when modelling a problem for solution by local search. They are quite different from the aims of modelling for DPLL, such as symmetry elimination and the level of consistency achieved by unit propagation. Thus modelling for local search is distinct from modelling for DPLL and is worth studying in its own right. Increased solution density might also be expected to aid backtrack search but this is not necessarily true. Structured SAT problems are likely to contain clusters of solutions, and Minton et al.'s *nonsystematic search hypothesis* [21] is that local search may benefit more than backtrack search from high solution density. This is because local search is largely immune to solution clustering, whereas backtrack search may start from a point that is very far from any cluster.

The remodelling approach can be seen as complementary to the preprocessing/algorithmic approaches of [15,23]. We are able to use an off-the-shelf local

search algorithm instead of a more complex new algorithm, and do not incur the runtime overhead of maintaining additional information. On the other hand, we require larger SAT encodings that also incur runtime overheads, and may use a prohibitive amount of memory in some cases. It would be interesting to combine the two approaches, by removing some structure via remodelling then handling what remains via dependency analysis.

Our new modelling techniques should find application to other problems. The parity constraint shallow encoding should be useful on other problems containing both clauses and parity constraints, such as the cryptanalysis problems of [20]. The new cardinality constraint encoding has many potential applications but we have not yet compared it empirically to known encodings. The superparallelism technique used to improve Towers of Hanoi can be applied to STRIPS models of other planning problems. Long-range dependencies based on explanatory frame axioms can be added to planning-as-SAT systems. Bounded model checking has a similar structure to planning and contains parity constraints, so it may also benefit from these techniques.

Finally, recall SAT challenge number six from [29]: to handle variable dependencies in local search. Our results further confirm the importance of this challenge, and show that a powerful alternative to modifying local search heuristics is to reduce or eliminate dependencies by remodelling the problem. In fact avoiding variable dependency by remodelling gives better results than (at least some) attempts to handle dependencies during search. Prevention does seem to be better than cure.

Acknowledgements. This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075. Thanks to the anonymous referees for helpful comments.

References

1. C. Ansótegui, F. Manyà. Mapping Problems With Finite-Domain Variables into Problems With Boolean Variables. *Seventh International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3542, Springer, 2004, pp. 1–15.
2. O. Bailleux, Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer, 2003, pp. 108–122.
3. P. Baumgartner, F. Massacci. The Taming of the (X)OR. *Computational Logic*, 2000.
4. J. M. Crawford, M. J. Kearns, R. E. Shapire. The Minimal Disagreement Parity Problem as a Hard Satisfiability Problem. Technical report, Computational Intelligence Research Laboratory and AT&T Bell Labs, 1994.
5. M. Ernst, T. Millstein, D. S. Weld. Automatic SAT-Compilation of Planning Problems. *Fifteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1997.

6. R. E. Fikes, N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3-4):189–208, 1971.
7. I. P. Gent. Arc Consistency in SAT. *Fifteenth European Conference on Artificial Intelligence*, IOS Press, 2002, pp. 121–125.
8. I. P. Gent, P. Prosser. SAT Encodings of the Stable Marriage Problem With Ties and Incomplete Lists. *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, 2002.
9. I. P. Gent, P. Prosser, B. Smith. A 0/1 Encoding of the GACLex Constraint for Pairs of Vectors. *International Workshop on Modelling and Solving Problems With Constraints*, ECAI, 2002.
10. A. Haas. The Case for Domain-Specific Frame Axioms. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, F. M. Brown (ed.), Lawrence, KS, 1987. Morgan Kaufmann, 1987.
11. E. A. Hirsch, A. Kojevnikov. Solving Boolean Satisfiability Using Local Search Guided by Unit Clause Elimination. *Seventh International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer, 2001, pp. 605–609.
12. F. Hutter, D. A. D. Tompkins, H. H. Hoos. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer, 2002, pp. 233–248.
13. S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. *Artificial Intelligence* vol. 45, Elsevier, 1990, pp. 275–286.
14. H. Kautz, D. McAllester, B. Selman. Encoding Plans in Propositional Logic. *Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 1996.
15. H. Kautz, D. McAllester, B. Selman. Exploiting Variable Dependency in Local Search. *Poster Sessions of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
16. H. Kautz, B. Selman. Ten Challenges *Redux*: Recent Progress in Propositional Reasoning and Search. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer, 2003, pp. 1–18.
17. H. Kautz, B. Selman. Planning as Satisfiability. *Tenth European Conference on Artificial Intelligence*, Wiley, 1992, pp. 359–363.
18. H. Kautz, B. Selman. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. *National Conference on Artificial Intelligence*, AAAI Press / The MIT Press, 1996, pp. 1194–1201.
19. C. Li. Integrating Equivalence Reasoning into Davis-Putnam Procedure. *Seventeenth National Conference on Artificial Intelligence*, AAAI/MIT Press, 2000, pp. 291–296.
20. F. Massacci. Using Walk-SAT and Rel-SAT for Cryptographic Key Search. *International Joint Conference on Artificial Intelligence*, 1999, pp. 290–295.
21. S. Minton, M. D. Johnston, A. B. Philips, P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1-3):161–205, 1992.
22. R. Muhammad, P. J. Stuckey. A Stochastic Non-CNF SAT Solver. *Trends in Artificial Intelligence, Ninth Pacific Rim International Conference on Artificial Intelligence, Lecture Notes in Computer Science* vol. 4099, Springer, 2006, pp. 120–129.

23. D. N. Pham, J. R. Thornton, A. Sattar. Building Structure into Local Search for SAT. *Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007, pp. 2359–2364.
24. S. D. Prestwich. SAT Problems With Chains of Dependent Variables. *Discrete Applied Mathematics* vol. 3037, Elsevier, 2002, pp. 1–22.
25. S. D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research* vol. 118, Kluwer Academic Publishers, 2003, pp. 137–150.
26. S. D. Prestwich. Modelling Clique Problems for SAT Local Search. *Third International Workshop on Local Search Techniques in Constraint Satisfaction*, 2006 (to appear).
27. S. D. Prestwich, A. Roli. Symmetry Breaking and Local Search Spaces. *Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer, 2005, pp. 273-287.
28. B. Selman, personal communication.
29. B. Selman, H. A. Kautz, D. A. McAllester. Ten Challenges in Propositional Reasoning and Search. *Fifteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1997, pp. 50–54.
30. D. A. D. Tompkins, H. H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. *Seventh International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3542, 2005, pp. 306–320.
31. J. P. Warners. A Linear-Time Transformation of Linear Inequalities Into Conjunctive Normal Form. *Information Processing Letters* 68:63–69, 1998.
32. J. Warners, H. van Maaren. A Two Phase Algorithm for Solving a Class of Hard Satisfiability Problems. *Operations Research Letters* 23(3–5):81–88, 1999.
33. W. Wei, B. Selman. Accelerating Random Walks. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer, 2002, pp. 216–232.
34. Z. Wu, B. Wah. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *Seventeenth National Conference on Artificial Intelligence*, 2000, pp. 310–315.
35. E. Zarpas. Back to the SAT05 Competition: an a Posteriori Analysis of Solver Performance on Industrial Benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation* vol. 2, 2006, research note, pp. 229–237.