SAT problems with chains of dependent variables

Steven Prestwich

Department of Computer Science, National University of Ireland at Cork, Cork, Ireland

Received 04 July 2000; received in revised form 03 March 2002; accepted 15 June 2002

Abstract

This paper has two related themes. Firstly, artificial SAT problems are used to show that certain chains of variable dependency have a harmful effect on local search, sometimes causing exponential scaling on intrinsically easy problems. Secondly, systematic, local and hybrid SAT algorithms are evaluated on Hamiltonian cycle problems, exposing weaknesses in all three. The connection between the two themes is that some Hamiltonian cycle problems also cause local search to scale badly, indicating that pathological variable dependencies occur in more realistic applications. More generally, the results highlight the need for alternative models and search algorithms, and new examples of both are described.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Satisfiability; Hamiltonian path; Search; Variable dependencies

1. Introduction

The satisfiability (SAT) problem is of both practical and theoretical importance, and was the first problem shown to be NP-complete. The SAT problem is to determine whether a Boolean expression has a satisfying set of truth assignments. The problems are usually expressed in conjunctive normal form: a conjunction of clauses $C_1 \land \cdots \land C_m$ where each clause $C$ is a disjunction of literals $l_1 \lor \cdots \lor l_n$ and each literal $l$ is either a Boolean variable $v$ or its negation $\overline{v}$. A Boolean variable can be assigned the values $T$ (true) or $F$ (false). The first SAT algorithms used depth-first search with inference rules such as unit propagation, an early example being the Davis–Putnam procedure in Loveland’s form [7]. Modern systematic SAT solvers are still based on this scheme and include Crawford and Auton’s TABLEAU [6], Bayardo and Schrag’s RELSAT [1], Li

E-mail address: s.prestwich@cs.ucc.ie (S. Prestwich).
and Anbulagan’s SATZ [24], Freeman’s POSIT [11], Zhang’s SATO [41], Silva and Sakallah’s GRASP [37] and Moskewicz et al.’s Chaff [26]. One difference between these solvers is their variable ordering heuristics, but SATZ also has a preprocessing phase, RELSAT, SATO and GRASP use look-back techniques, and Chaff monitors selected literals to reduce propagation overheads.

A more recent approach is the use of stochastic search: simulated annealing, evolutionary algorithms, neural networks, ant colonies or hill climbers. A simple form of stochastic search is local search, which searches the neighbourhood of a point \( \sigma \) in a space by making local moves. The neighbourhood consists of the set of points \( \sigma' \) that can be reached by a single local move. The aim is to minimise (or equivalently to maximise) some objective function \( f(\sigma) \) on the space. A local move \( \sigma \rightarrow \sigma' \) can be classified as backward, forward or sideways, depending on whether \( f(\sigma') - f(\sigma) \) is positive, negative or zero. Local search may converge on a local minimum: a point that has lower value than all its neighbours but is not a global minimum. The aim of backward moves is to escape from local minima by providing noise, while sideways moves are used to traverse function plateaus. Local search is incomplete and therefore cannot be used to prove unsatisfiability, but beats systematic search on a variety of satisfiable problems (for example, large random 3-SAT). The usual way of applying local search to SAT (and other constraint problems) is to define the search space as the set of total assignments, a local move as a change in the truth value of a single variable (a flip), and the objective function to be minimised as the number of unsatisfied clauses. Pioneering SAT algorithms of this type were Selman, Levesque and Mitchell’s GSAT [35] and Gu’s algorithms [19]. GSAT has since been enhanced in various ways by Selman et al. [33], giving several WSAT variants augmented by heuristics such as random walk. Other local search SAT algorithms include Shang and Wah’s DLM [36] and Resende and Feo’s GRASP [31] (unrelated to Silva and Sakallah’s systematic GRASP algorithm). DLM is based on Lagrange multipliers, while GRASP is a greedy, randomised, adaptive algorithm. There are other types of SAT algorithm (for example those based on binary decision diagrams) but a complete survey is outside the scope of this paper.

Neither backtracking nor local search is seen as adequate for all constraint problems [13]. SAT encodings of real-world problems often have a great deal of structure, giving them different properties than artificial random problems. Such problems are often solved effectively by backtrackers which are able to exploit structure via constraint propagation, unlike most local search algorithms. In particular, functional dependencies among variables are propagated in linear time by backtrackers but in approximately quadratic time by most local search algorithms [22]. Unfortunately, backtrackers do not always scale well to very large problems, even when augmented with intelligent backtracking techniques. Hence, neither type of search is ideally suited to problems that are both large and highly structured. This situation has recently motivated research into hybrid search algorithms, the aim of which is summarised by two of the 10 SAT challenges posed by Selman et al. [34]:

(6) Improve stochastic local search on structured problems by efficiently handling variable dependencies.
(7) Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.

A recent hybrid is constrained local search (CLS), described for SAT [27,28] and other combinatorial problems [27,29,30]. Its SAT implementation can be summarised as a Davis–Logemann–Loveland (DLL) procedure, with an incomplete form of backtracking that gives it local search-like scalability [27]. In fact, CLS can be viewed as local search in a space of consistent partial truth assignments, hence its name. It handles variable dependencies in the same way as a backtracker—by constraint propagation—so we claim that it answers challenge [6]. On a set of artificial problems combining structure and randomness it out-performed both local search and backtracking [28], so it also answers challenge [7]. However, as we shall show, it has weaknesses.

This paper has two related themes. In Section 2 we use artificial SAT problems to show that certain patterns of variable dependency are particularly harmful for local search, causing poor scaling even on simple problems. This highlights the need for alternative models or algorithms such as CLS. In Section 3 we compare a systematic (SATZ), a local (WSAT) and a hybrid algorithm (CLS) on random and structured Hamiltonian path problems, using both a standard and a novel SAT encoding. We draw some conclusions regarding search algorithms, SAT modeling and Hamiltonian paths as SAT benchmarks. Section 4 discusses results and connects the two themes: some Hamiltonian path problems also cause poor scaling in local search, showing that the artificial SAT problems have realistic counterparts. Section 5 discusses related work and Section 6 concludes the paper.

2. Chains of variable dependency

In this section we explore the effects of dependent variables on local search. When attempting to understand a complex phenomenon it is often useful to consider simplified cases. We therefore construct artificial problems consisting of little but dependent variables, and apply two of the best current local search algorithms to them: WSAT with the original Selman–Kautz–Cohen (SKC) and novelty heuristics [33]. None of the problems are intrinsically hard and are solved instantly by any backtracking or resolution-based SAT algorithm (including CLS). Nevertheless, they are interesting from the point of view of understanding algorithm behaviour.

First consider the SAT problem which we shall denote by $A_k$, consisting of the conjunction of the following formulae:

$v_1,$
$v_1 \rightarrow v_2,$
$v_2 \rightarrow v_3,$
$\vdots$
$v_{k-1} \rightarrow v_k.$
Each variable $v_i$ is dependent on $v_{i-1}$, giving a chain of dependencies. We solved the problems using WSAT with the SKC and novelty heuristics, infinite cutoff, default noise of 50% and 1000 solutions, obtaining the curves in Fig. 1. The graph is a log-log plot and the curves are almost straight lines, indicating that the number of flips is a monomial in $k$. Applying a least-squares fit we find that the SKC heuristic takes $o(k^{2.180})$, which is in good agreement with the observation by Kautz et al. [22] that local search propagates variable dependencies in quadratic time. However, novelty takes $o(k^{1.618})$ and seems to handle dependent variables more efficiently.

In the next problem each variable is dependent on more than one previous variable in a chain. Denote by $B_k$ the conjunction of the following formulae:

\[
\begin{align*}
&v_1 \land v_2, \\
&v_1 \land v_2 \rightarrow v_3, \\
&v_2 \land v_3 \rightarrow v_4, \\
&\vdots \\
&v_{k-2} \land v_{k-1} \rightarrow v_k.
\end{align*}
\]

Plotting on a semi-log scale in Fig. 2 we find near-straight lines, indicating exponential scaling. Applying a least-squares fit we find SKC takes $o(1.722^k)$ and novelty $o(1.612^k)$ flips. It is surprising to find such a simple problem that is so hard for robust, state-of-the-art algorithms. What features of $B_k$ make it so hard for local search?

An unimportant feature is the fact that each clause (in conjunctive normal form) has two negative and one positive literal: we can replace any subset of the literals with their negations in all clauses without affecting the problem hardness. Nor is the exact
Another property of $B_k$ is that it has a single solution and therefore a **backbone** of maximal size. Singer et al. [38] propose an explanation for the fact that solvable random 3-SAT problems at the phase transition are harder for local search than those in the overconstrained region: they have large, fragile backbones. The **backbone** of a SAT problem is the set of literals entailed by its clauses. A backbone is **fragile** if the random removal of a small number of clauses greatly reduces its size. $B_k$’s backbone is also fragile because removing any clause prevents the entailment of all subsequent truth assignments. Random clause removal is likely to remove at least one early clause, drastically reducing the size of the backbone. The $B_k$ problem can be seen as an idealised problem with a large, fragile backbone, and its hardness for WSAT is further evidence of the hardness of such problems for local search. However, the following example shows that a large, fragile backbone is not an important feature of $B_k$. Denote by $C_k$ the conjunction of the following formulae:

\[
\begin{align*}
    (v_1 \land v_2) & \lor (v'_1 \land v'_2), \\
    (v_1 \land v_2 \rightarrow v_3) & \land (v'_1 \land v'_2 \rightarrow v'_3),
\end{align*}
\]
\[(v_2 \land v_3 \rightarrow v_4) \land (v'_2 \land v'_3 \rightarrow v'_4), \]
\[\vdots\]
\[(v_{k-2} \land v_{k-1} \rightarrow v_k) \land (v'_{k-2} \land v'_{k-1} \rightarrow v'_k).\]

\(C_k\) is similarly hard for WSAT. Again each variable is dependent on two that are deeper in the structure. Note that any truth assignment with \(v_1, \ldots, v_k\ T\) is a solution, and similarly for any truth assignment with \(v'_1, \ldots, v'_k\ T\). In each case the other \(k\) variables may be assigned any truth value, which has two consequences. Firstly, two solutions are \((v_i = T, v'_i = F)\) and \((v_i = F, v'_i = T)\) for \(i = 1, \ldots, k\). These have no truth assignments in common so \(C_k\) has no backbone. Secondly, there are \(2^{k+1}\) solutions instead of just one, though this is still a low solution density because there are \(2^{2k}\) possible assignments. Hence neither a backbone nor a single solution are the important features of \(B_k\).

However, the chain of relationships between variables is important. Consider this similar-looking problem:
\[
v_1 \land v_2,
v_1 \land v_2 \rightarrow v_3,
v_4 \land v_5,
v_4 \land v_5 \rightarrow v_6,
\vdots
v_{3k-2} \land v_{3k-1},
v_{3k-2} \land v_{3k-1} \rightarrow v_{3k}.
\]

Many variables are dependent on two other variables, as in \(B_k\). The difference is that the implications do not form a long chain as in \(B_k\), which consists of a single, deep structure; this problem consists of many smaller, unrelated structures. WSAT solves this problem in \(k/2\) flips under both the novelty and SKC heuristics, indicating that from a random truth assignment in which half the variables match the solution, any sequence of flips on variables occurring in violated clauses leads directly to a solution.

Our explanation is as follows. In conjunctive normal form the \(B_k\) clauses are of the form \((\neg v_i \lor \neg v_{i+1} \lor v_{i+2})\). If such a clause is violated then the three variables must be \(T, T\) and \(F\), respectively. There are three ways to remove the violation but only one of them moves closer to the solution (in which all variables are \(T\)): flipping \(v_{i+2}\) from \(F\) to \(T\). Flipping either of the other two variables from \(T\) to \(F\) moves further away from the solution in terms of Hamming distance (the number of variables with different truth value). If there is no reason to choose the correct flip then WSAT is twice as likely to choose an incorrect one. Nor do these incorrect flips necessarily create further violations, which might cause WSAT to avoid them, because each \(o\) variable occurs negatively more often than positively. The same argument applies to \(C_k\) and explains why WSAT has difficulty solving problems containing long chains of variable dependency, in which each variable depends on more than one other variable.
As a final experiment, we construct a problem in which each variable is dependent on two earlier variables, but not the two immediately preceding it. Denote by $D_k$ the conjunction of the following formulae:

$$v_1 \land v_2,$$

$$v_{i/2} \land v_{i-1} \rightarrow v_i \quad (i = 3, \ldots, k).$$

Each variable is dependent on two variables that occur earlier in the series $1, \ldots, k$; one is the variable immediately preceding it, and the other is half-way between the current variable and the first one. Surprisingly, novelty cannot solve these problems without frequent restarts, so we plot results only for SKC. In Fig. 3 it can be seen that SKC scales monomially, and a best fit gives the formula $o(k^{3.854})$. That is, the exponential scaling does depend to some extent on the exact form of the chain. However, $D_k$ has a much higher scaling factor than $A_k$, showing that dependency on more than one earlier variable is still more harmful than dependency on one.

A reasonable question is: do these patterns of variable dependency naturally occur in more realistic SAT problems, and do they cause poor scaling for WSAT? We shall present evidence that they do.

3. Hamiltonian path problems

The Hamiltonian path problem (HPP) is an NP-complete problem defined as follows. A graph $G = (V, E)$ consists of a set $V$ of vertices ($V = \{v_1, \ldots, v_n\}$) and a set $E$ of (directed or undirected) edges $(v_i, v_j)$ between vertices. We assume undirected edges. Two vertices connected by an edge are said to be adjacent. A Hamiltonian path is an ordering of vertices such that adjacent vertices in the path are also adjacent in the
graph, and such that each vertex occurs in the path exactly once. A path is closed if its first and last vertices are adjacent, in which case the path is called a Hamiltonian cycle. It is more common to search for cycles than paths, in which case we are trying to establish whether the graph is Hamiltonian. We shall consider SAT encodings of the more general HPP, though the paths we construct can all be extended to cycles.

The Knight’s tour problem can be formulated as an HPP or HCP (Hamiltonian cycle problem). The HCP is also closely related to the travelling salesperson problem (TSP), which is to find a cycle of minimal cost. The HPP and HCP have been comparatively neglected in artificial intelligence research, compared with more popular problems such as graph colouring, TSP and SAT. Furthermore, the HPP does not seem to have been previously encoded as SAT, though Hoos [20] experimented with two SAT encodings of the (directed) HCP. We shall use two SAT encodings of the HPP, one using a well-known approach and the other a new approach. In both HPP-SAT encodings we specify the first and last vertices in the path, \( v_{\text{first}} \) and \( v_{\text{last}} \).

3.1. The absolute encoding

Firstly, an encoding specifying the absolute positions of vertices in the path, which we shall call the absolute encoding. This is a standard approach, called the sparse encoding by Hoos [20], whereby each possible assignment of a value (position) to a variable (vertex) is represented by a Boolean variable. We use \( n^2 \) variables \( p_{i,j} \) (\( i,j = 1,\ldots,n \)) to denote that vertex \( i \) occurs \( j \)th in the path. The SAT clauses are as follows:

- each vertex must occur at least once in the path: \( p_{1,1} \lor \cdots \lor p_{i,n} \);
- no vertex can occur more than once in the path: \( \neg p_{i,j} \lor \neg p_{i,k} \);
- each location in the path must have at least one vertex assigned to it: \( p_{1,i} \lor \cdots \lor p_{n,i} \);
- no two vertices can occur at the same place in the path: \( \neg p_{j,i} \lor \neg p_{k,i} \);
- two vertices cannot be contiguous in the path if they are not adjacent in the graph:
  \( \neg p_{i,k} \lor \neg p_{j,k+1} \) and \( \neg p_{j,k} \lor \neg p_{i,k+1} \);
- the first vertex occurs at the start of the path: \( p_{\text{first},1} \);
- the last vertex occurs at the end of the path: \( p_{\text{last},n} \).

This is arguably the most natural SAT encoding. Another encoding, described by Iwama and Miyazaki [21], is referred to by Hoos [20] as the compact encoding. This is based on a binary representation of values and uses \( n \log_2(n) \) SAT variables. Both the sparse and compact encodings are derived from the same constraint satisfaction problem (CSP) encoding of the HCP, which constructs permutations of the vertices. In this paper, we will not investigate the compact encoding, but instead introduce a new approach.

3.2. The relative encoding

We now describe a new SAT encoding of HPP based on the relative positions of vertices in the path, which we shall refer to as the relative encoding. The hope is that such an encoding will improve the performance of SAT algorithms. Imagine that
we have constructed a partial Hamiltonian path but omitted a necessary vertex. In the absolute encoding, to insert the vertex we must move all subsequent vertices. If we represent only relative positions then insertion becomes much easier, because by changing the relative positions of two vertices we implicitly change the positions of their neighbours in the sequence.

We require $2n^2$ variables, of two types denoted by $s$ and $o$. The $s_{i,j}$ ($i, j = 1, \ldots, n$) denote a successor relation between vertices $v_i$ and $v_j$: $s_{i,j} = T$ if and only if $v_j$ appears immediately after $v_i$ in the path. The SAT clauses for $s$ are:

- a vertex $v_j$ cannot be the successor of another vertex $v_i$ if they are not adjacent in the graph, nor if $v_i$ is the last vertex or $v_j$ the first vertex or $i = j$: $\neg s_{i,j}$;
- each vertex except $v_{\text{last}}$ has at least one successor: $s_{i,j} \lor s_{i,k} \lor \cdots$ for edges $(v_i, v_j), (v_i, v_k), \ldots$;
- no vertex has more than one successor: $\neg s_{i,j} \lor \neg s_{i,k}$ for edges $(v_i, v_j)$ and $(v_i, v_k)$;
- each vertex except $v_{\text{first}}$ is the successor of at least one vertex: $s_{j,i} \lor s_{k,i} \lor \cdots$ for edges $(v_j, v_i), (v_k, v_i), \ldots$;
- no vertex is the successor of more than one vertex: $\neg s_{j,i} \lor \neg s_{k,i}$ for edges $(v_i, v_j)$ and $(v_i, v_k)$.

We must also ensure that each vertex occurs no more than once in the path, and that the path is connected (does not consist of two or more disjoint paths). Both are true if the path vertices form a total ordering. The $o_{i,j}$ ($i, j = 1, \ldots, n$) denote this total ordering, $o_{i,j} = T$ if and only if $v_i$ appears before $v_j$ in the path:

- transitivity: $\neg o_{i,j} \lor \neg o_{j,k} \lor o_{i,k}$;
- antisymmetry: $\neg o_{i,j} \lor \neg o_{j,i}$;
- irreflexivity: $\neg o_{i,i}$;
- the ordering relation must apply to all pairs of vertices: $o_{i,j} \lor o_{j,i}$;
- the first vertex precedes all others: $o_{\text{first},i}$;
- the last vertex succeeds all others: $o_{i,\text{last}}$.

Finally, the relationship between the successor and ordering relations is simply:

- $s_{i,j} \rightarrow o_{i,j}$.

### 3.3. Constrained local search

Before evaluating SAT algorithms on HPPs we describe the constrained local search (CLS) hybrid SAT algorithm. It begins like any DLL procedure, selecting a variable using some heuristic, assigning a value to it selected by another heuristic, and removing domain values by unit resolution. A dead-end occurs when the selected variable cannot be assigned either truth value, because in both cases unit propagation causes the domain of some other variable to become empty (sometimes called domain wipe-out). On reaching a dead-end the algorithm backtracks by unassigning one or more variables, then tries again to assign values to selected variables. The novel feature of CLS is
its choice of backtracking variable, which is done using another heuristic with no attempt to maintain completeness. As in dynamic backtracking [16] only the selected variable is unassigned, and assignments made since are not undone. Because the search is incomplete we may sometimes need to unassign two or more variables, to avoid becoming trapped in a state from which no variable can change value. We do this by adding an integer parameter $B \geq 1$ to the algorithm, and unassign $B$ variables at each dead end. The value of $B$ is determined by trial and error, and tuned to given SAT problems.

In what sense is this a hybrid algorithm? Though it is a modified backtracker we claim that it performs local search in a novel space: the partial variable assignments that are consistent under unit resolution. In other words, it combines the search strategy of local search with the search space of DLL procedures. To complete the local search analogy: the objective function to be minimised is the number of unassigned variables, forward local moves are variable assignments, backward moves are the randomised backtracks, and $B$ plays the role of a tunable noise parameter. In contrast, most local search algorithms search a space of total variable assignments, and minimise the number of false clauses. It is outside the scope of this paper to argue in detail whether CLS is local search, but it has been shown [27] that it scales almost precisely as standard local search on hard random 3-SAT problems.

A complication arises in the maintenance of variable domains. Suppose we reach a dead-end after assigning variables $v_1, \ldots, v_k$ with $v_{k+1}, \ldots, v_n$ unassigned. We would like to backtrack by unassigning some arbitrary variable $v_u$ ($1 \leq u \leq k$), leaving the domains in the state they would have been in had we assigned only $v_1, \ldots, v_{u-1}, v_{u+1}, \ldots, v_k$. How can we do this efficiently? One way to characterise unit resolution is as follows: a truth value $x$ is in the domain of a currently unassigned variable $v$ if and only if assigning $v = x$ would not falsify any clause. In DLL procedures this principle is used to update the domains of unassigned variables as assignments are added and removed. We generalise the idea slightly by associating with each truth value $x$ in the domain of each variable $v$ a conflict count $C_{v,x}$. The value of $C_{v,x}$ for an unassigned variable $v$ is the number of clauses that would become false if the assignment $v = x$ were added. It follows that a value $x$ is currently in $\text{dom}(v)$ if and only if $C_{v,x} = 0$. Now on assigning or unassigning a variable $v_i$, we incrementally update conflict counts in all other relevant variable domains. For example, on assigning $v_i = T$ we check all clauses in which $v_i$ occurs. Suppose one such clause is $\neg v_i \lor v_j \lor v_k$ with $v_k$ currently assigned to $F$ and $v_j$ unassigned. Then $v_j = F$ would make the clause false so we increment $C_{v_j,F}$. Now suppose we unassign $v_k$; then the assignment $v_j = F$ would no longer make the clause false and we decrement $C_{v_j,F}$. This technique allows us to unassign variables in any order, giving complete backtracking flexibility while efficiently exploiting unit propagation.

It remains to describe the specific heuristics used. Simpler heuristics were previously reported [27,28] but the following often give better results:

- The heuristic for selecting a variable to assign is a weakened form of the TABLEAU variable selection rule. It randomly selects a variable with domain size 1 if possible. If not, it selects an unassigned variable $v$ with maximum value of $1024P_v N_v + P_v + N_v$.
where $P_v [N_v]$ is the number of positive [negative] occurrences of $v$ in binary clauses (those currently having 2 free variables) containing no true literals; ties are broken by selecting the variable that most recently could not be assigned.

- The heuristic for selecting for a variable to be unassigned chooses a least-constrained variable $v_i$, that is with minimum $C_{v_i,T} + C_{v_i,F}$; ties are broken randomly.
- The value ordering heuristic is based not on number of constraint violations as usual, but on previous assignments. For a given variable, the heuristic first tries to select a different value than its previous assignment (previous assignments are remembered, and initialised to random values). On achieving this, the heuristic flips to preferring the same values as in previous assignments. At the next dead-end it reverts to preferring different values, and so on. The heuristic aims ideally to change one assignment per dead-end, though it may be unable to change any assignments, or forced to change more than one.

### 3.4. Results on random graphs

We now present empirical results for various graphs types, using the two encodings described above and three SAT algorithms: the WSAT local search algorithm, the SATZ systematic backtracker and the CLS hybrid. WSAT and SATZ are among the most powerful representatives of their types.

Because most previous HCP work has been done on random graphs, we begin with these. A simple form of random graph is defined by the number of its vertices $n$ and an edge probability $p$: each pair of vertices is adjacent with probability $p$. The mean degree of a graph at the phase transition has been shown theoretically [23] and empirically [4] to be $\log n + \log \log n$. The degree of a vertex is the number of its adjacent vertices, and the mean degree of a graph is the mean of its vertex degrees. Cheeseman et al. [4] and Frank and Martel [10] encoded the HCP as a CSP and found that, as with several other combinatorial problems including random 3-SAT, there is a phase transition in solvability for random graphs as a graph parameter is varied. Both applied backtracking algorithms to small graphs (Frank and Martel up to 14 vertices, Cheeseman et al. up to 24 vertices) and found these problems hard. Frank et al. [9] found that most problems from the phase transition are easy, but that average performance was strongly affected by a few hard instances. Vandegriend and Culberson [39] found no significant peak in problem hardness at the transition; specialised HCP algorithms easily solve graphs of up to 1500 vertices. However, the problems are asymptotically easy, and they did find some small ($n < 100$) graphs hard. We shall generate a few small graphs to evaluate SAT algorithms under different encodings.

Because our formulation is in terms of HPP we must specify start and end vertices. This makes it harder to construct solvable HPPs, because as well as constructing graphs of the appropriate mean degree, we must also select a pair of adjacent vertices whose connecting edge appears in a Hamiltonian cycle. We construct graphs with 24 vertices using Culberson’s graph generator, adjusting the edge probability until the mean degree is close to the phase transition value (4.334 for 24 vertices) and selecting the

first vertex pair in the generated graph as the start and end vertices. We accept the resulting graph if some algorithm finds a Hamiltonian circuit within a reasonable time. We generated three Hamiltonian graphs in this way and, by the results cited above, these are unlikely to represent hard HCP problems. For the sake of reproducibility we describe the graph parameters. The graphs denoted RAND0888, RAND3344 and RAND8200 have (respectively) random seeds 888, 3344 and 8200, edge probabilities 0.177, 0.1917 and 0.1615, and each have mean degree 4.333.

Here and in subsequent experiments, various parameter settings were tried for WSAT and CLS and the reported results are the best. To do both algorithms justice, it was found necessary to tune them to the two encodings separately; there was often no parameter setting that worked well on the same problem under both encodings. On the random graphs, the CLS noise parameter $B$ was set to 150 under the relative encoding and 30 under the absolute encoding. WSAT used the novelty heuristic with the following parameters: for the absolute encoding 10% noise, with cutoffs of 10,000; and for the relative encoding 65% noise with a cutoff of 1,000,000. The version of SATZ used is SATZ-RAND with noise set to zero and cutoff to infinity. Both SATZ-RAND and WSAT were obtained from the SATLIB web page. All algorithms were executed on a 300 MHz DEC Alphaserver 1000A 5/300 under Unix, and means were taken over 100 runs.

The results are shown in Fig. 4, back denoting number of backtracks, flip the number of WSAT flips and sec seconds of CPU time. SATZ solves the problems in very few backtracks under both encodings, whereas the CSP backtracking algorithm of Cheese- man et al. took tens of thousands. The algorithm of Vandegriend and Culberson did not backtrack at all on random graphs, but it used specialised techniques such as graph pruning. For a general-purpose SAT backtracker SATZ does remarkably well, incidentally showing that the three graphs are not intrinsically hard as HCPs and are therefore typical. Under the relative encoding the CLS results are similar to those of SATZ in terms of execution times, though it took tens of thousands of backtracks. The same applies to WSAT under the absolute encoding, using even more flips but slightly smaller execution times. However, under the absolute encoding CLS takes significantly longer.

---

2 [http://aida.intellektik.informatik.th-darmstadt.de/~hoos/SATLIB](http://aida.intellektik.informatik.th-darmstadt.de/~hoos/SATLIB)
in terms of both backtracks and time, and under the relative encoding WSAT takes even longer. So although all three algorithms perform similarly under their best encoding (in terms of time), the two non-systematic algorithms perform poorly under their worst encoding. SATZ is the most robust algorithm on these problems.

3.5. Results on a knight’s tour graph

Vandegriend and Culberson [39] found no classes of random graph with a phase transition in problem hardness, so we consider structured graphs in the remainder of this section. A classic combinatorial problem is the knight’s tour problem (KTP). Given a chess board, not necessarily of standard size or shape, we must find a sequence of moves by a knight chess piece such that each square of the board is visited exactly once. A knight’s move can be defined as one square in the X- or Y-direction followed by two squares in the Y- or X-direction, and an example of a tour on a standard $8 \times 8$ chess board is shown in Fig. 5. The knight moves from the square labelled 1 to the square labelled 2, then 3 and so on. The KTP is a solved problem in the sense that linear-time algorithms are known for its solution, but as a benchmark problem it still attracts considerable interest. Moreover, a generalisation of the problem with rectangular boards and non-standard knight’s moves has been shown to be a source of HCPs that are hard for some algorithms [39].

By assigning a vertex to each chess board square and an edge between each pair of vertices representing squares separated by a single knight’s move, the KTP can be formulated as an HCP or HPP. Because the SAT encodings are quite large (the $8 \times 8$ board under the absolute encoding gives almost half a million clauses) we use a $6 \times 6$ board, whose graph we refer to as KT6. The results are shown in Fig. 6. The noise levels used for CLS were 800 for the relative encoding and 60 for the absolute encoding. For WSAT 5% noise and a cutoff of 200,000 flips were used under the absolute encoding, and several parameter settings were tried without success for the relative encoding.

Interestingly, the results are qualitatively similar to those of the random graphs: again, SATZ is robust and solves both encodings with few backtracks; CLS performs similarly.
under the relative encoding in terms of time, though using many more backtracks; WSAT performs similarly under the absolute encoding in terms of time, but with even more flips; CLS is significantly slower under the absolute encoding; and WSAT even slower under the relative encoding (failing to solve the problem at all within a reasonable time).

3.6. Results on linked cliques

So far SATZ has given the most robust performance, solving problems efficiently under both encodings. However, it seems likely that CLS and WSAT, as local search algorithms, will scale better than SATZ on large HCPs. Unfortunately, the KTP is too large to test this prediction, and random graphs yield easy HCPs. We therefore construct a simple parameterised HCP problem: a graph consisting of a collection of cliques linked to form a ring as in Fig. 7. A c-clique is a complete subgraph with c vertices, in which every vertex is adjacent to every other vertex. The idea behind using this form of graph is that omitting a vertex from the path under construction causes backtracking much later, because there is no way to get back to the omitted vertex. These graphs are easy for any good backtracking HCP algorithm, which usually prefer vertices of least degree (the vertices used to link the cliques have higher degree than the rest, so these will be selected last).

We use cliques of five vertices and denote a graph with k 5-cliques by CRk. The CLS noise level was set to $\frac{1}{8}$ the number of SAT variables for both encodings. WSAT under the absolute encoding used 5% noise and cutoffs of 100 times the number of SAT variables, and under the relative encoding 98% and 50 times the number of SAT variables (though unusual, this parameter setting gave the best results). SATZ on the six cliques problem under the relative encoding was halted after 4 h without finding a solution.
The results are shown in Fig. 8 and are quite different to the previous results. SATZ uses few backtracks under the absolute encoding, showing that its heuristics enable it to discover the best vertex with few mistakes. WSAT uses far more flips but is even faster in execution time. CLS is the worst algorithm under this encoding. Under the relative encoding SATZ handles the small graphs well but has even worse scalability than CLS does under the absolute encoding. CLS has much better scalability, and surprisingly WSAT’s performance is quite close to that of CLS. The most likely explanation for WSAT’s good performance here is that the problems have relatively high solution density. Entering a 5-clique, there are six ways of moving through it such that all vertices are visited before moving to the next clique. There are 10 incorrect ways, that is omitting at least one vertex. Each clique must be traversed correctly to find a path, so for \( k \) 5-cliques the probability of a random path being correct is \((\frac{1}{8})^k\). For six cliques this is 0.00278, which is much higher than for the Knight’s Tour and random graphs.

### 3.7. Results on ring graphs

Finally, we consider a very simple graph type consisting only of a Hamiltonian cycle. This is shown in Fig. 9 and is sometimes called a ring graph. Ring graphs are trivial for specialised HCP algorithms, and we would hope that they are also trivial for SAT algorithms. This turns out to be the case for SATZ and CLS which solve such problems without backtracking, but not always for WSAT.

We evaluate WSAT on ring graphs of various sizes, RING\(k\) denoting a graph with \(k\) vertices. The best results under the absolute encoding were found with novelty, zero noise and low cutoff: \(4n\) was used, where \(n\) is the number of SAT variables. Under the

<table>
<thead>
<tr>
<th>problem</th>
<th>CLS</th>
<th>SATZ</th>
<th>WSAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>enc</td>
<td>varclauses</td>
<td>back</td>
</tr>
<tr>
<td>CR1</td>
<td>abs</td>
<td>25</td>
<td>112</td>
</tr>
<tr>
<td>CR2</td>
<td>abs</td>
<td>100</td>
<td>1336</td>
</tr>
<tr>
<td>CR3</td>
<td>abs</td>
<td>225</td>
<td>5198</td>
</tr>
<tr>
<td>CR4</td>
<td>abs</td>
<td>400</td>
<td>13190</td>
</tr>
<tr>
<td>CR5</td>
<td>abs</td>
<td>625</td>
<td>26812</td>
</tr>
<tr>
<td>CR6</td>
<td>abs</td>
<td>900</td>
<td>47564</td>
</tr>
<tr>
<td>CR1</td>
<td>rel</td>
<td>50</td>
<td>156</td>
</tr>
<tr>
<td>CR2</td>
<td>rel</td>
<td>200</td>
<td>1068</td>
</tr>
<tr>
<td>CR3</td>
<td>rel</td>
<td>450</td>
<td>3424</td>
</tr>
<tr>
<td>CR4</td>
<td>rel</td>
<td>800</td>
<td>7980</td>
</tr>
<tr>
<td>CR5</td>
<td>rel</td>
<td>1250</td>
<td>15486</td>
</tr>
<tr>
<td>CR6</td>
<td>rel</td>
<td>1800</td>
<td>26692</td>
</tr>
</tbody>
</table>

Fig. 8. Results on linked 5-cliques.
relative encoding 99% noise and a cutoff of $200n$ were used. The results in Fig. 10 show that under the absolute encoding local search is quite efficient, applying only about $3n$ flips for all problems (where $n$ is the number of SAT variables). However, under the relative encoding it is very inefficient. Plotting number of flips against number of SAT variables, we did not obtain unambiguously exponential or monomial behaviour. To obtain a better idea of WSAT scaling we executed it with infinite cutoff and the SKC heuristic, for ring graphs with 3–21 vertices. Again the scaling appeared to be less than exponential but more than monomial. If we assume monomial scaling then the closest fit is an inefficient $o(n^{2.6})$ where $n$ is the number of SAT variables. This is further evidence that the relative encoding is highly unsuitable for conventional local search.

4. Discussion of the results

The results of Section 3 show that HPPs yield interesting SAT problems, with performance strongly dependent on graph type, SAT encoding and search algorithm. Comparing a local, a systematic and a hybrid SAT algorithm, none dominated the others over all problems and encodings. CLS usually performed competitively under the relative encoding but less well under the absolute encoding, except on the ring graphs. Conversely, WSAT performed well under the absolute encoding but sometimes very poorly under the relative encoding, except on the linked clique graphs. SATZ was
relatively insensitive to encoding, except on the linked clique graphs under the relative encoding where it scaled poorly.

The results of Section 3 are also interesting from the viewpoint of SAT encoding. It is usually considered desirable to minimise the number of variables in a SAT encoding [21]. As noted by Hoos [20], an encoding with fewer variables has a higher solution density, and therefore might be expected to be easier for local search algorithms. However, Hoos showed that the compact encoding is harder for local search algorithms, because it introduces search space features detrimental to local search. In contrast to Hoos’s results, the relative encoding is even more sparse than the absolute encoding, with twice as many variables and roughly half as many clauses, yet it is very poor for local search. There seems to be no clear pattern in the behaviour of local search on different SAT encodings as a function of encoding size. This is further evidence that it is important to try more than one encoding when solving a new SAT problem.

The results are both positive and negative for our hybrid approach. CLS avoided SATZ’s scaling problems on the relative encoding of the linked cliques, and WSAT’s pathological behaviour on the relative encoding. On the other hand, under the absolute encoding it is sometimes the worst algorithm. It would have been gratifying to find HCP problems on which CLS conclusively outperforms other approaches. However, the $B_k$ problems from Section 2 can be used to construct SAT problems that are hard for (say) SATZ and WSAT but not CLS. We use an approach related to morphing [14], in which two or more problems are combined to give a hybrid problem combining structure and randomness in a controllable ratio. A “type A” morph of two SAT problems includes clauses from one problem with some probability $p$, and clauses from the other with probability $1 - p$. The aim of morphing is to create problems with a controllable combination of structure and randomness. Our aim is slightly different—to construct a problem that is at least as hard as two subproblems—so we simply include all the clauses from two SAT problems. The first subproblem is hard for SATZ but easy for WSAT: $f600$, a large (600 variable) satisfiable random 3-SAT problem from the phase transition, taken from the SATLIB web page. The second subproblem is easy for SATZ but hard for WSAT: $B_{100}$ (see Section 2). Then we form a composite problem denoted $f600 \land B_{100}$ consisting of the union of the clauses from the two problems, renumbering variables so that they are distinct. Adding the $B_{100}$ clauses to those of $f600$ cannot help SATZ to solve $f600$; similarly, adding the $f600$ clauses to those of $B_{100}$ help WSAT to solve $B_{100}$; therefore $f600 \land B_{100}$ should be hard for both SATZ and WSAT. As expected, we were unable to solve $f600 \land B_{100}$ at all using either SATZ or WSAT, but CLS solves it in a few seconds. Though highly artificial, this example shows the existence of problems best solved by a hybrid such as CLS. Of course, given a problem $P$ that is hard for CLS, the same technique can be used to construct a problem best solved by SATZ ($P \land B_{100}$) or WSAT ($P \land f600$). Perhaps the best conclusion is that no search algorithm can be expected to be best on all NP-complete problems: an obvious point, but it is useful to find concrete examples that illustrate it, and which may lead to improvements in current SAT algorithms.

We now relate the HPP results of Section 3 to the artificial examples of Section 2. Do HPPs under the relative encoding contain chains of dependency like those of $B_k$ or $D_k$, which affect local search performance in the same way? We can say that
(for example) $o_{1,10}$ depends on $o_{1,3}$ and $o_{3,10}$ because there is a transitivity clause $o_{1,3} \land o_{3,10} \rightarrow o_{1,10}$; alternatively we could say that it depends on $o_{1,9}$ and $o_{9,10}$; and so on. For each alternative, the two variables on which it depends each depend in turn on two other variables, until we reach a variable $o_{i,i+1}$. So each variable occurs in several chains, and in each chain it depends on two variables occurring earlier. In other words, embedded in the transitivity clauses of the relative encoding are many linked $D_k$-like problems (though the form of the chains is not exactly that of $D_k$). We believe that this explains the poor scaling of WSAT on HPPs as easy as the ring graphs. Note that the same effects do not occur in the absolute encoding, which has no $D_k$-like chains. If this explanation is correct, then when applying conventional local search we should avoid SAT encodings in which such chains of dependency occur (unless solutions are common, as in the linked clique graphs). Unfortunately, for some problems there may not be a choice of encodings. It is therefore valuable to have alternative algorithms such as CLS, which search different spaces and minimise different objective functions.

5. Related work

Related work on Hamiltonian cycles and their encodings has already been mentioned. The problem of handling variable dependencies in local search has previously been studied by Kautz, McAllester and Selman, who show that some of a problem’s structure can be extracted from its syntactic form [22]. In their approach each problem variable is classed as either dependent or independent: a dependent variable is a simple Boolean function of other variables. For a given problem there may be more than one possible classification, but there is often a natural division. Local search can then be applied to the smaller set of independent variables, and the values of the dependent variables handled separately. This algorithm is called Dagsat, and on a set of highly structured problems its performance is roughly quadratic in the number of independent variables. In contrast, WSAT has performance roughly quadratic in the total number of variables. However, Dagsat uses static analysis and cannot exploit dynamic variable dependencies which occur only in certain search contexts. A backtracker may treat a Boolean variable as both independent and dependent during a single search. The status of a variable is effectively decided by dynamic variable ordering heuristics and exploited by unit propagation. CLS handles variable dependencies in the same way.

There are other proposals besides CLS for improving the scalability of backtrackers. Gomes et al. [18] periodically restart chronological or intelligent backtrackers with slightly randomised heuristics. This often improves scalability, and has the advantage over CLS that no special implementation techniques are required so that an existing backtracker can easily be adapted. However, on some problems the approach does not scale as well as local search or CLS [28], and it is unable to solve the composite problem $f_{600} \land B_{100}$ discussed in Section 4 (because it cannot solve $f_{600}$). Another possibility is the execution of two or more search algorithms in parallel, or in a round-robin fashion on a single processor, a scheme referred to as an algorithm portfolio [17]. For example we could form a portfolio consisting of SATZ and WSAT, which would be able to solve many problems that are easy for systematic or for
local search. SATZ would handle dependent variables efficiently while WSAT would handle large problems. The advantage of a portfolio is that it solves a problem in a time approximately that taken by its best algorithm: it is more robust than any of the algorithms it contains. However, a portfolio can do no better than its best algorithm on a given problem, so the composite problem \( f_{600} \land B_{100} \) will also be hard for this portfolio. A further possibility is the modification of local search to handle the relative encoding, simply by changing the objective function. Clause weighting schemes (for example [3,25,32]) do this by dynamically changing the objective function in order to escape local minima. It would be interesting to apply a clause weighting algorithm to the HPPs. However, weighting handles variable dependencies indirectly. On one class of problems (the single-solution AIM benchmarks with low clause/variable ratio) a clause weighting scheme was much faster than local search without weighting, but CLS was much faster again [28].

Several researchers have attempted to analyse why local search performs badly on certain problems. Yokoo [40] shows that some problems yield search spaces with many local minima. Frank et al. [8] analyse local minima and plateaus in objective functions for combinatorial problems. Hoos [20] investigates search space properties such as ruggedness and local minima branching. Gent and Walsh [15] suggest a combination of factors for GENSAT (a generalised local search procedure related to WSAT) on certain problems: scarcity of solutions; the indirectness of the SAT encoding which allows problem constraints to be easily violated; the non-constructive nature of most local search algorithms, which allows clauses themselves to be violated, causing the generation of states that constraint propagation would immediately rule out; and the smallness of the flip operation used to change truth values. Clark et al. [5] show that the scarcity of solutions is a contributing factor. Singer et al.’s work on large, fragile backbones in random 3-SAT [38] has already been mentioned. We have shown that certain variable dependencies cause even worse local search performance than previously supposed, and attempted to explain why.

6. Conclusion

This paper makes several contributions to the use of SAT technology for solving combinatorial problems:

- A simple class of SAT problems with long chains of dependent variables was described, and shown to cause very poor scaling in two local search algorithms. Such pathological benchmarks are useful for guiding algorithm development.
- It was shown that on some Hamiltonian path problems the new encoding causes similar scaling in local search, and provided evidence that the cause is similar patterns of variable dependency.
- A hybrid local search algorithm was described that handles dependencies differently, and does not exhibit the pathological behaviour.
- A new SAT encoding for the Hamiltonian path problem was presented, adding to the known number of such encodings. Access to a variety of problem encodings allows.
better analysis of the applicability and robustness of search algorithms. Moreover, constraint modelling has recently been identified as a key challenge for constraint programming [12], yet there are few available guidelines. New models are therefore of theoretical and practical interest.

- SAT encodings with fewer variables have been shown to hinder local search [20]. Similarly, it has been found that increasing the number of variables improves the performance of genetic algorithms on highly constrained problems [2]. However, the new encoding uses at least twice as many variables as previous encodings, yet it can drastically degrade both local search and backtracking performance. On the other hand, it appears better suited to hybrid search than a standard encoding. This complicates the picture of what makes an appropriate SAT encoding.

- Hamiltonian cycles in random graphs from the phase transition have been found hard for CSP and SAT backtrackers. It was shown that they can be solved in a few backtracks by a SAT backtracker.

It seems likely that other deeply structured combinatorial problems will give rise to similar variable dependency chains. Future work will include a search for, and better characterisation of, such problems. We will also attempt to cure the weaknesses of our hybrid algorithm exposed in this paper, specifically by the addition of more powerful variable ordering heuristics from algorithms such as SATZ, or by adapting random walk ideas from WSAT. It would also be interesting to modify WSAT or a related algorithm to overcome the pathological behaviour described in the paper, by the integration of propagation techniques.

References


