Local Search for Unsatisfiability

Steven Prestwich[1] and Inês Lynce[2]

[1] Cork Constraint Computation Centre,
Department of Computer Science, University College, Cork, Ireland
s.prestwich@cs.ucc.ie
[2] IST/INESC-ID, Technical University of Lisbon, Portugal
ines@sat.inesc-id.pt

Abstract. Local search is widely applied to satisfiable SAT problems, and on some classes outperforms backtrack search. An intriguing challenge posed by Selman, Kautz and McAllester in 1997 is to use it instead to prove unsatisfiability. We investigate two distinct approaches. Firstly we apply standard local search to a reformulation of the problem, such that a solution to the reformulation corresponds to a refutation of the original problem. Secondly we design a greedy randomised resolution algorithm that will eventually discover proofs of any size while using bounded memory. We show experimentally that both approaches can refute some problems more quickly than backtrack search.

1 Introduction

Most SAT solvers can be classed either as *complete* or *incomplete*, and the complete algorithms may be based on resolution or backtracking. Resolution provides a complete proof system by refutation [24]. The first resolution algorithm was the Davis-Putnam (DP) procedure [6] which was then modified to the Davis-Putnam-Logemann-Loveland (DPLL) backtracking algorithm [7]. Because of its high space complexity, resolution is often seen as impractical for real-world problems, but there are problems on which general resolution proofs are exponentially smaller than DPLL proofs [4]. Incomplete SAT algorithms are usually based on *local search* following early work by [14, 26], but metaheuristics such as genetic algorithms may also be applied. On some large satisfiable problems, local search finds a solution much more quickly than complete algorithms, though it currently compares rather badly with backtracking algorithms on industrial benchmarks.

An interesting question is: can local search be applied to *unsatisfiable* problems? Such a method might be able to refute (prove unsatisfiable) SAT problems that defy complete algorithms. This was number five of the ten SAT challenges posed by Selman, Kautz and McAllester in 1997: *design a practical stochastic local search procedure for proving unsatisfiability* [25]. While substantial progress has been made on several challenges, this one remains wide open [18], and we explore two distinct ways of attacking it. It was suggested in [18] that local search could be applied to a space of incomplete proof trees, and our first approach uses a related idea: we apply standard local search to a space of (possibly incorrect)

proof graphs each represented by a clause list. In order to exploit current local search technology, this is done via a new reformulation of the original SAT problem. Our second approach is a new local search algorithm that explores a space of resolvent multisets, and aims to derive the empty clause non-systematically.

The paper is organised as follows. In Section (2) we SAT-encode the meta-problem of finding a proof for a given SAT problem. Section (3) describes the new local search algorithm. Section (4) discusses related work. Finally, Section (5) concludes the paper.

2 Local search on a reformulated problem

Our first approach is to apply existing local search algorithms to a reformulation of the original SAT problem: a solution to the reformulation corresponds to a refutation of the original problem. A potential drawback with this approach is the sheer size of the reformulation, and it is noted in [18] that *a key issue is the need to find smaller proof objects*. Much of our effort is therefore devoted to reducing the size of the reformulation.

2.1 Initial model

Suppose we have an unsatisfiable SAT problem with n variables $v_1 \dots v_n$ and m clauses, and we want to prove unsatisfiability using no more than r resolvents. We can represent the proof as an ordered list of $m + r$ clauses, with the first m clauses being those of the problem, and each of the other clauses being a resolvent of two earlier clauses (which we call the *parents* of the resolvent). The final resolvent must be empty. This meta-problem can be SAT-encoded as follows.

Define meta-variables x_{ikp} (where $i < k$ and $p \in \{0, 1\}$) to be true (T) iff clause k in the list is a resolvent of clause i and another unspecified clause in the list, using an unspecified variable occurring in i as either a positive ($p = 1$) or negative ($p = 0$) literal. Define $u_{kv} = T$ iff clause k was the result of a resolution using variable v. Define $o_{ivp} = T$ iff literal v/p occurs in clause i: $v/1$ denotes literal v, and $v/0$ literal $\bar{v}$, and we shall refer to p as the *sign* of the literal. Define $d_{kvq} = T$ iff variable v in resolvent k occurs in a literal v/q in the parent clause. For example if clause 10 ($\bar{v}_{36} \vee v_{37}$) is resolved with clause 12 ($v_{36} \vee v_{38}$) to give clause 17 ($v_{37} \vee v_{38}$) then the following meta-variables are all true: $o_{36,10,0}$, $o_{37,10,1}$, $o_{36,12,1}$, $o_{38,12,1}$, $o_{37,17,1}$, $o_{38,17,1}$, $x_{10,17,0}$, $x_{12,17,1}$, $u_{17,36}$, $d_{17,37,0}$, $d_{17,38,1}$. There are $O(r^2 + rm + rn + mn)$ meta-variables.

The meta-clauses are as follows, with their space complexities in terms of number of literals. We represent the SAT problem by the following unary meta-clauses for all literals v/p [not] occurring in clauses i:

$$o_{ivp} \quad [\bar{o}_{ivp}] \qquad O(mn) \tag{1}$$

Each resolvent must be the resolvent of one earlier clause in the list using a variable positively and one negatively. We use two sets of meta-clauses to ensure

that at least one, and no more than one, earlier clause is used:

$$\bigvee_i x_{ikp} \qquad O(r(m+r)) \tag{2}$$

$$\bar{x}_{ikp} \vee \bar{x}_{jkp} \qquad O(r(m+r)^2) \tag{3}$$

Exactly one variable is used to generate a resolvent:

$$\bigvee_v u_{kv} \qquad O(nr) \tag{4}$$

$$\bar{u}_{kv} \vee \bar{u}_{kw} \qquad O(n^2 r) \tag{5}$$

If k is a resolvent using v then v does not occur in k:

$$\bar{u}_{kv} \vee \bar{o}_{kvp} \qquad O(nr) \tag{6}$$

If k is the resolvent of i and another clause using a variable with sign p, and k is a resolvent using variable v, then v/p occurs in i:

$$\bar{x}_{ikp} \vee \bar{u}_{kv} \vee o_{ivp} \qquad O(nr(m+r)) \tag{7}$$

Every literal in k occurs in a literal of at least one of its parent clauses:

$$\bar{o}_{kvp} \vee d_{kv0} \vee d_{kv1} \qquad O(nr) \tag{8}$$

$$\bar{d}_{kvq} \vee \bar{o}_{kvp} \vee \bar{x}_{ikq} \vee o_{ivp} \qquad O(nr(m+r)) \tag{9}$$

(Variables d_{kvq} were introduced to avoid referring to both parent clauses of k in a single meta-clause, which would increase the space complexity.) If i is a parent clause of k using a variable occurring with sign p in i, v/p occurs in i, and v was not used in the resolution generating k, then v/p occurs in k:

$$\bar{x}_{ikp} \vee \bar{o}_{ivp} \vee u_{kv} \vee o_{kvp} \qquad O(nr(m+r)) \tag{10}$$

If i is a parent clause of k using a variable occurring with sign p in i, and $v/\bar{p}$ occurs in i, then $v/\bar{p}$ occurs in k:

$$\bar{x}_{ikp} \vee \bar{o}_{iv\bar{p}} \vee o_{kv\bar{p}} \qquad O(nr(m+r)) \tag{11}$$

The last resolvent is empty:

$$\bar{o}_{m+r\,vp} \qquad O(n) \tag{12}$$

Tautologous resolvents are excluded (we assume that the original problem contains no tautologies):

$$\bigvee_p \bar{o}_{kvp} \qquad O(nr) \tag{13}$$

Every resolvent is used in a later resolution, so for $m \leq i < k \leq m+r$:

$$\bigvee_k \bigvee_p x_{ikp} \qquad O(r^2) \tag{14}$$

This meta-encoding has $O(nr(m+r)^2 + n^2 r)$ literals which can be reduced in several ways as follows.

2.2 Model reduction by unit resolution

The model can be reduced by observing that many meta-variables appear in unary meta-clauses, and can therefore be eliminated. For every variable v and every original clause i we have a meta-variable o_{ivp} that occurs in a unary meta-clause (either $\bar{o}_{ivp}$ or o_{ivp} depending on whether literal v/p occurs in i). All of these $O(mn)$ meta-variables can be eliminated by resolving on the unary meta-clauses (1). For any such unary clause l: any clause $A \vee l$ is a tautology and can be removed; any clause $A \vee \bar{l}$ can be replaced by A via unit resolution and subsumption; and clause l itself can then be deleted by the pure literal rule. This leaves $O(r^2 + mr + nr)$ meta-variables. If we are searching for a short proof then mn is the dominant term in the number of meta-variables, so we have eliminated most of them. The complexity of some sets of meta-clauses is reduced by unit resolution. Suppose that λ is the mean clause length divided by n, so that $mn\lambda$ is the size in literals of the original problem. Then after unit resolution (7,8,9) become $O(nr(m(1-\lambda)+r)$ and (10,11) become $O(nr(m\lambda+r))$. The total space complexity of (7,8,9,10,11) is still $O(nr(m+r))$ but we will eliminate some of these below.

2.3 Model reduction by weakening rule

We can greatly reduce the space complexity by allowing the proof to use the *weakening rule*, in which any literals may be added to a clause as long as this does not create a tautology. This allows us to remove some of the largest sets of meta-clauses: (7,8,9), removing the $nr(m(1-\lambda)+r)$ terms from the space complexity. This is a significant reduction: in a SAT problem without tautologous clauses $\lambda \leq 0.5$ but a typical problem will have a much smaller value, for example in a random 3-SAT problem $\lambda = \frac{3}{n}$, and the meta-encoding with $r = 10$ of a 600-variable problem from the phase transition is reduced from approximately 92,000,000 to 300,000 clauses (in both cases applying all our other reduction techniques). Removing these meta-clauses allows new literals to be added to a resolvent. For example $A \vee x$ and $B \vee \bar{x}$ may be the parent clauses of $A \vee B \vee C$ for some disjunction C. The d_{kvq} variables no longer occur in any meta-clauses and can be removed. The total space complexity is now $O(r(m+r)^2 + n^2r + nr(m\lambda+r))$ literals.

2.4 Model reduction by allowing multiple premises

We can eliminate the $r(m+r)^2$ complexity term by dropping meta-clauses (3), allowing a resolvent to have more than one parent clause of a given sign. For example $A \vee x$, $A \vee x$ and $B \vee \bar{x}$ may be parent clauses of $A \vee B$. When combined with the use of the weakening rule we obtain the more general: $x \vee A_i$ and $\bar{x} \vee B_j$ may all be parent clauses of $\bigvee_i A_i \vee \bigvee_j B_j$ (assuming that this clause is non-tautologous) because each of the possible resolvents can be extended to this clause via weakening. The total space complexity is now $O(n^2r + nr(m\lambda + r))$ literals.

2.5 Model reduction by ladder encoding

We can eliminate the n^2r term by replacing meta-clauses (5) with a *ladder encoding* of the at-most-one constraint, adapted from [11]. Because we are using local search we are not concerned with the propagation properties of the encoding, so we can omit some clauses from the original ladder encoding. Define $O(nr)$ new variables l_{kv} and add ladder validity clauses $l_{kv} \vee \bar{l}_{k\,v-1}$ and channelling clauses $\bar{u}_{kv} \vee \bar{l}_{k\,v-1}$ and $\bar{u}_{kv} \vee l_{kv}$. These clauses are sufficient to prevent any pair u_{kv}, u_{kw} from both being true. This set of meta-clauses has only $O(nr)$ literals instead of the $O(n^2r)$ of (5), so the total space complexity is now $O(nr(m\lambda + r))$. In summary, the reduced meta-encoding contains $O(r^2 + mr + nr)$ variables and $O(nr^2 + rs)$ literals where $s = mn\lambda$ is the size in literals of the original SAT problem. We conjecture that this cannot be reduced further.

2.6 Discussion of the meta-encoding

A useful property of the model is that we can overestimate r, which we would not normally know precisely in advance. This is because for any proof of length r there exists another proof of length $r + 1$. Suppose a proof contains a clause $i : x \vee A$ and a later clause $j : \bar{x} \vee B$, which are resolved to give $k : A \vee B \vee C$ where A, B, C are (possibly empty) disjunctions of literals and C is introduced by weakening. Then between j and k we can insert a new clause $k' : x \vee A \vee B \vee C$ derived from i, j and derive k from i, k'. If a appears in C then first remove it; we can always remove literals introduced by weakening without affecting the correctness of the proof.

Whereas local search on a SAT problem can prove satisfiability but not unsatisfiability, local search on the meta-encoding can prove unsatisfiability but not satisfiability. We can apply any standard local search algorithm for SAT to a meta-encoded problem. Many such algorithms have a property called *probabilistic approximate completeness* (PAC) [15]: the probability of finding a solution tends to 1 as search time tends to infinity. PAC has been shown to be an important factor in the performance of practical local search algorithms [15], and we expect it to be important also in proving unsatisfiability. If we use a PAC local algorithm then it will eventually refute any unsatisfiable problem, given sufficient time and assuming that we set the proof length r high enough.

2.7 Experiments

The local search algorithm we use is RSAPS [16], a state-of-the-art dynamic local search algorithm that has been shown to be robust over a variety of problem types using default runtime parameters. Its performance can sometimes be improved by parameter tuning but in our experiments the difference was not great, nor did any other local search algorithm we tried perform much better. All experiments in this paper were performed on a 733 MHz Pentium II with Linux.

We will make use of two unsatisfiable problems. One is derived from the well-known pigeon hole problem: place $n + 1$ pigeons in n holes, such that no hole

receives more than one pigeon. The SAT model has variables v_{ij} for pigeons i and holes j. Clauses $\bigvee_j v_{ij}$ place each pigeon in at least one hole, and clauses $\bar{v}_{ij} \vee \bar{v}_{i'j}$ prevent more than one pigeon being placed in any hole. The 2-hole problem, which we denote by HOLE2, has a refutation of size 10. The other problem is one we designed and call HIDER:

$$\begin{array}{llll} a_1 \vee b_1 \vee c_1 & \bar{a}_1 \vee d \vee e & \bar{b}_1 \vee d \vee e & \bar{c}_1 \vee d \vee e \\ a_2 \vee b_2 \vee c_2 & \bar{a}_2 \vee \bar{d} \vee e & \bar{b}_2 \vee \bar{d} \vee e & \bar{c}_2 \vee \bar{d} \vee e \\ a_3 \vee b_3 \vee c_3 & \bar{a}_3 \vee d \vee \bar{e} & \bar{b}_3 \vee d \vee \bar{e} & \bar{c}_3 \vee d \vee \bar{e} \\ a_4 \vee b_4 \vee c_4 & \bar{a}_4 \vee \bar{d} \vee \bar{e} & \bar{b}_4 \vee \bar{d} \vee \bar{e} & \bar{c}_4 \vee \bar{d} \vee \bar{e} \end{array}$$

From these clauses we can derive $d \vee e$, $\bar{d} \vee e$, $d \vee \bar{e}$ or $\bar{d} \vee \bar{e}$ in 3 resolution steps each. For example resolving $(a_1 \vee b_1 \vee c_1)$ with $(\bar{a}_1 \vee d \vee e)$ gives $(b_1 \vee c_1 \vee d \vee e)$; resolving this with $(\bar{b}_1 \vee d \vee e)$ gives $(c_1 \vee d \vee e)$; and resolving this with $(\bar{c}_1 \vee d \vee e)$ gives $(d \vee e)$. From these 4 resolvents we can obtain d and $\bar{d}$ (or e and $\bar{e}$) in 2 resolution steps. Finally, we can obtain the empty clause in 1 more resolution step, so this problem has a refutation of size 15. We designed HIDER to be hidden in random 3-SAT problems as an unsatisfiable sub-problem with a short refutation. All its clauses are ternary and no variable occurs more than 12 times. In a random 3-SAT problem from the phase transition each variable occurs an expected 12.78 times, so these clauses blend well with the problem, and a backtracker has no obvious reason to focus on the new variables. Moreover, a resolution refutation of HIDER requires the generation of quaternary clauses, which SATZ's *compactor* preprocessor [19] does not generate. We combine both HIDER and HOLE2 with a random 3-SAT problem by renumbering their variables so that they are distinct from the 3-SAT variables, then taking the union of the two clause sets. We denote such a combination of two problems A and B by $A + B$, where A's variables are renumbered. We performed experiments to obtain preliminary answers to several questions.

What is the effect of the weakening rule on local search performance? To allow weakening in the refutation we may remove some meta-clauses as described in Section 2.3. Dropping clauses from a SAT problem can increase the solution density, which sometimes helps local search to solve the problem, but here it has a bad effect. RSAPS was able to refute HIDER in a few seconds (it is trivial to refute by DPLL) using the meta-encoding without weakening, but with weakening it did not terminate in a reasonable time. This was surprising for such a tiny problem: perhaps the meta-encoding is missing some vital ingredient such as a good set of implied clauses.

What is the effect of allowing unused resolvents? We tested the effect of dropping meta-clauses (14), which are optional (and do not affect the space complexity). Removing them allows a resolvent to be added to the proof then not used further. In experiments omitting them made HIDER much harder to refute, presumably by not penalising the construction of irrelevant chains of resolvents.

What effect does the allowed proof length have on local search performance? On HIDER under the original meta-encoding (without weakening) RSAPS finds the refutation very hard with r set to its minimum value of 15.

However, as r increases the runtime decreases to a few seconds. The number of flips decreases as r increases, but the increasing size of the model means that using larger r eventually ceases to pay off.

Can local search on a reformulation beat DPLL on an unsatisfiable problem? We combined HOLE2 with f600, a fairly large, satisfiable, random 3-SAT problem from the phase transition region.[1] HOLE2+f600 has a refutation of size 10 and we set $r = 20$. ZChaff [21] aborted the proof after 10 minutes, whereas RSAPS found a refutation in a median of 1,003,246 flips and 112 seconds, over 100 runs. However, SATZ refutes the problem almost instantly.

3 Local search on multisets of resolvents

A drawback with the reformulation approach is that it is only practical for proofs of relatively small size. Our second approach is to design a new local search algorithm that explores multisets of resolvents, and can in principle find a proof of any size while using only bounded memory. To determine how much memory is required we begin by reviewing a theoretical result from [8].

Given an unsatisfiable SAT formula ϕ with n variables and m clauses, a general resolution refutation can be represented by a series of formulae $\phi_1, \ldots, \phi_s$ where ϕ_1 consists of some or all of the clauses in ϕ, and ϕ_s contains the empty clause. Each ϕ_i is obtained from ϕ_{i-1} by (optionally) deleting some clauses in ϕ_{i-1}, adding the resolvent of two remaining clauses in ϕ_{i-1}, and (optionally) adding clauses from ϕ. The *space* of a proof is defined as the minimum k such that each ϕ_i contains no more than k clauses.

Intuitively each ϕ_i represents the set of *active* clauses at step i of the proof. Inactive clauses are not required for future resolution, and after they have been used as needed they can be deleted. It is proved in [8] that the space k need be no larger than $n + 1$: possibly fewer clauses than in ϕ itself.

The *width* of a proof is the length (in literals) of the largest clause in the proof. Any non-tautologous clause must have length no greater than n, so this is a trivial upper bound for the width used for our algorithm. However, short proofs are also narrow [3] so in practice we may succeed even if we restrict resolvent length to some small value. This may be useful for saving memory on large problems.

Thus we can in principle find a large refutation using a modest amount of working memory. But finding such a proof may not be easy. We shall use the above notions as the basis for a novel local search algorithm that performs a randomised but biased search in the space of formulae ϕ_i. Each ϕ_i will be of the same constant size, and derived from ϕ_{i-1} by the application of resolution or the replacement of a clause by one taken from ϕ. We call our algorithm RANGER (RANdomised GEneral Resolution).

[1] Available at http://www.cs.ubc.ca/~hoos/SATLIB/

3.1 The algorithm

The RANGER architecture is shown in Figure 1. It has six parameters: the size k of the ϕ_i, the width w, three probabilities p_i, p_t, p_g and the formula ϕ.

```
1  RANGER(φ, p_i, p_t, p_g, w, k):
2     i ← 1 and φ_1 ← {any k clauses from φ}
3     while φ_i does not contain the empty clause
4        with probability p_i
5           replace a random φ_i clause by a random φ clause
6        otherwise
7           resolve random φ_i clauses c, c' giving r
8           if r is non-tautologous and |r| ≤ w
9              with probability p_g
10                if |r| ≤ max(|c|, |c'|) replace the longer of c, c' by r
11             otherwise
12                replace a random φ_i clause by r
13       with probability p_t
14          apply any satisfiability-preserving transformation to φ, φ_i
15       i ← i + 1 and φ_{i+1} ← {the new formula}
16    return UNSATISFIABLE
```

Fig. 1. The RANGER architecture

RANGER begins with any sub-multiset $\phi_1 \subseteq \phi$ (we shall interpret ϕ, ϕ_i as multisets of clauses). It then performs iterations i, each either replacing a ϕ_i clause by a ϕ clause (with probability p_i), or resolving two ϕ_i clauses and placing the result r into ϕ_{i+1}. In the latter case, if r is tautologous or contains more than w literals then it is discarded and $\phi_{i+1} = \phi_i$. Otherwise a ϕ_i clause must be removed to make room for r: either (with probability p_g) the removed clause is the longer of the two parents of r (breaking ties randomly), or it is randomly chosen. In the former case, if r is longer than the parent then r is discarded and $\phi_{i+1} = \phi_i$. At the end of the iteration, any satisfiability-preserving transformation may (with probability p_t) be applied to ϕ, ϕ_{i+1} or both. If the empty clause has been derived then the algorithm terminates with the message "unsatisfiable". Otherwise the algorithm might not terminate, but a time-out condition (omitted here for brevity) may be added.

Local search algorithms usually use *greedy* local moves that reduce the value of an objective function, or *plateau* moves that leave it unchanged. However, they must also allow non-greedy moves in order to escape from local minima. This is often controlled by a parameter known as *noise* (or *temperature* in simulated annealing). But what is our objective function? Our goal is to derive the empty clause, and a necessary condition for this to occur is that ϕ_i contains at least some small clauses. We will call a local move *greedy* if it does not increase the number of literals in ϕ_i. This is guaranteed on line 10, so increasing p_g increases the

greediness of the search, reducing the proliferation of large resolvents. There may be better forms of greediness but this form is straightforward, and in experiments it significantly improved performance on some problems.

3.2 A convergence property

We show that RANGER has the PAC property, used here to mean that given sufficient time it will refute any unsatisfiable problem:

Theorem. *For any unsatisfiable SAT problem with n variables and m clauses,* RANGER *is PAC if $p_i > 0$, $p_i, p_t, p_g < 1$, $w = n$ and $k \geq n + 1$.*

Proof. Firstly, any proof of the form in [8] can be transformed to one in which (i) all the ϕ_i have exactly k clauses, possibly with some duplications, and (ii) ϕ_{i+1} is derived from ϕ_i by replacing a clause either by a ϕ clause or a resolvent of two ϕ_i clauses. Take any proof and expand each set ϕ_i to a multiset ϕ'_i by adding arbitrary ϕ clauses, allowing duplications. Suppose that ϕ_{i+1} was originally derived from ϕ_i by removing a (possibly non-empty) set S_1 of clauses, adding the resolvent r of two ϕ_i clauses, and adding a (possibly non-empty) set S_2 of clauses. Then ϕ'_{i+1} can be derived from ϕ_i by removing a multiset S'_1 of clauses, adding r, and adding another multiset S'_2. Because all the ϕ'_i are of the same size k it must be true that $|S'_1| + 1 = |S'_2|$. Then we can derive ϕ'_{i+1} from ϕ'_i by replacing one S'_1 clause by r, then then replacing the rest by S'_2 clauses.

Secondly, any transformed proof may be discovered by RANGER from an arbitrary state ϕ'_i. Suppose that the proof begins from a multiset ϕ^*. Then ϕ'_i may be transformed into ϕ^* in at most k moves (they may already have clauses in common), each move being the replacement of a ϕ'_i clause by a ϕ^* clause. From ϕ^* the transformed proof may then be recreated by RANGER, which at each move may perform any resolution or replacement. □

3.3 Subsumption and pure literal elimination

Lines 13–14 provide an opportunity to apply helpful satisfiability-preserving transformations to ϕ or ϕ_i or both. We apply the subsumption and pure literal rules in several ways:

- Randomly choose two ϕ_i clauses c, c' containing a literal in common. If c subsumes c' then replace c' by a random ϕ clause.
- Randomly choose two ϕ clauses c, c' containing a literal in common. If c subsumes c' then delete c'.
- Randomly choose a ϕ clause c and a ϕ_i clause c' containing a literal in common. If c strictly subsumes c' then replace c' by c.
- If a randomly-chosen ϕ_i clause c contains a literal that is pure in ϕ then replace c by a randomly-chosen ϕ clause.
- If a randomly-chosen ϕ clause c contains a literal that is pure in ϕ then delete c from ϕ.

Each of these rules is applied once per RANGER iteration with probability p_t. Using ϕ_i clauses to transform ϕ, a feature we shall call *feedback*, can preserve useful improvements for the rest of the search. (We believe that for these particular transformations we can set $p_t = 1$ without losing completeness, but we defer the proof until a later paper.) Note that if ϕ is reduced then this will soon be reflected in the ϕ_i via line 5 of the algorithm.

The space complexity of RANGER is $O(n + m + kw)$. To guarantee the PAC property we require $w = n$ and $k \geq n + 1$ so the complexity becomes at least $O(m+n^2)$. In practice we may require k to be several times larger, but a smaller value of w is often sufficient. Recall that the meta-encoding has space complexity $O(nr^2 + rs)$ where s is the size of the original problem and r the proof length. Thus for short proofs the meta-encoding may be economical, but RANGER's space complexity has the important advantage of being independent of the length of the proof.

A note on implementation. We maintain a data structure that records the locations in ϕ and ϕ_i of two clauses containing each of the $2n$ possible literals. Two locations in this structure are randomly updated at each iteration and used during the application of resolution, subsumption and the pure literal rule. The implementation could no doubt be improved by applying the pure literal rule and unit resolution as soon as possible, but our prototype relies on these eventually being applied randomly.

3.4 Experiments

Again we performed experiments to answer some questions.

Does RANGER perform empirically like a local search algorithm? Though RANGER has been described as a local search algorithm and has the PAC property, it is very different from more standard local search algorithms. It is therefore interesting to see whether its runtime performance is similar to a standard algorithm. Figure 2 shows run-length distributions of the number of iterations required for RANGER to refute HIDER+f600, with $p_i = 0.1$, $p_t = 0.9$, $k = 10m$, and 250 runs per curve. With no greed ($p_g = 0.00$) there is heavy-tailed behaviour. With maximum greed ($p_g = 1.00$, not shown) a refutation cannot be found because HIDER's refutation contains quaternary resolvents, which require non-greedy moves to derive from the ternary clauses. With high greed ($p_g = 0.99$) the median result is worse but there is no heavy tail. The best results are with a moderate amount of greed (such as $p_g = 0.50$): there is no heavy-tailed behaviour and the median result is improved.

What is the effect of space on RANGER's performance? Though a low-space proof may exist, performance was often improved by allowing more space: $10m$ usually gave far better results than the theoretical minimum of $n + 1$.

What is the effect of feedback on RANGER's performance? Feedback was observed to accelerate refutation on some problems, especially as ϕ is sometimes reduced to a fraction of its original size.

Can RANGER beat a non-trivial complete SAT algorithm on an unsatisfiable problem? It refutes HOLE2+f600 in about 0.15 seconds: recall that

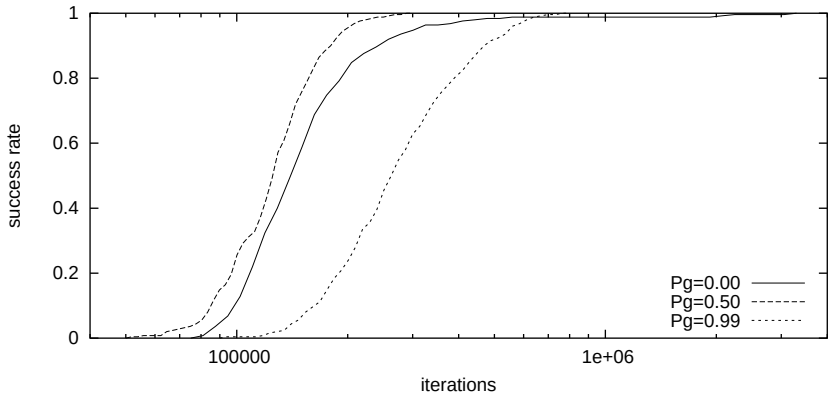

Fig. 2. Three run-length distributions for RANGER

RSAPS on the meta-encoding took over 100 seconds, which beat ZChaff but not SATZ. RANGER also refutes HIDER+f600 in about 1 second, easily beating several backtrackers which take at least tens of minutes: ZChaff, SATZ, Siege, POSIT and Minisat (if we renumber the variables, because Minisat branches on the variables in reverse lexicographic order). On this problem RANGER performs roughly 130,000 iterations per second.

How does RANGER perform on benchmarks? It can refute the automotive product configuration problems of [27] in seconds or minutes, depending on the instance, but these are easier for current backtrackers. It is interesting to note that these problems are reduced to a fraction of their original size (approximately $\frac{1}{20}$) by RANGER's feedback mechanism. RANGER also refutes aim-100-2_0-no-1 in a few seconds, whereas Rish & Dechter's DR resolution algorithm [23] takes tens of minutes, as does the Tableau backtracker. But their resolution/backtrack hybrid algorithms take under 1 second, as does the *compactor* algorithm alone. On unsatisfiable random 3-SAT problems RANGER performs very poorly: an interesting asymmetry, given that local search performs well on *satisfiable* random problems. The DR algorithm refutes the dubois20/21 benchmarks quite quickly while RANGER finds them very hard. RANGER refutes ssa0432-003 in about 40 minutes, backtrackers take seconds, DR takes a long time, and a resolution/backtrack hybrid takes 40 seconds. RANGER, DR and the hybrids take a long time on bf0432-007, but current backtrackers find it easy. These results are mixed, but in future work we hope to find a useful class of SAT problems on which RANGER is the algorithm of choice. These problems should be unsatisfiable, fairly large, not susceptible to backtrack search, and require resolution proof of non-trivial width.

4 Related work

As of 2003 no work had been done on using local search to prove unsatisfiability [18], and we are unaware of any such work since. However, some research may be viewed as moving in this direction. Local search can be made complete by using learning techniques, for example GSAT with dynamic backtracking [12], learnSAT [22] and Complete Local Search [9]. But the aim of these algorithms seems to be to improve performance on satisfiable problems, not to speed up proof of unsatisfiability. Learning algorithms may also require exponential memory in the worst case, though in practice polynomial memory is often sufficient.

Backtracking algorithms have been modified with local search techniques, to improve performance on both satisfiable and unsatisfiable problems. Recently [20] proposed randomly selecting backtrack points within a complete backtrack search algorithm. Search restarts can also be seen as a form of randomization within backtrack search, and have been shown to be effective on hard SAT instances [13]. The search is repeatedly restarted whenever a cutoff value is reached. The algorithm proposed is not complete, since the restart cutoff point is kept constant. But in [2] search restarts were combined with learning for solving hard, real-world instances of SAT. This latter algorithm is complete, since the backtrack cutoff value increases after each restart. Local search has also been used to finding a good variable ordering, which is then used to speed up a DPLL proof of unsatisfiability [5].

Hybrid approaches have also been tried for the more general class of QBF formulas. WalkQSAT [10] has two main components. The first is the QBF engine, which performs a backjumping search based on conflict and solution directed backjumping. The second is the SAT engine, which is a slightly adapted version of the WalkSAT local search algorithm used as an auxiliary search procedure to find satisfying assignments quickly. The resulting solver is incomplete as it can terminate without a definite result. WalkMinQBF [17] has also two main components. The first is a local search algorithm that attempts to find an assignment to the universal variables that is a witness for unsatisfiability. The second is a complete SAT solver that tests the truth or falsity of SAT formulas that result from assigning the universal variables. WalkMinQBF is also incomplete: it outputs *unsatisfiable* if a certificate of unsatisfiability is found, otherwise it outputs *unknown*.

5 Conclusion

We proposed two distinct ways in which local search can be used to prove unsatisfiability, and demonstrated that there exist problems on which they outperform backtracking (and in some cases systematic resolution) algorithms. As far as we know, this is the first work reporting progress on the fifth SAT challenge of [25]. In experiments with both methods we noted an interesting trend: that short and low-space proofs are harder to find by local search. It is therefore advisable to allow greater length and space than theoretically necessary.

The more successful of our two approaches used randomised general resolution with greedy heuristics and other techniques. It is perhaps surprising that this relatively short-sighted algorithm beats a sophisticated dynamic local search algorithm, though they explore different search spaces and cannot be directly compared. In future work we hope to improve the first approach by modifying the reformulation, and the second by finding improved heuristics.

Powerful proof systems such as general resolution can in principle be used to solve harder problems than more simple systems. In practice such systems are rarely used, partly because of their excessive memory consumption, but also because no good strategy is known for applying the inference rules in order to find a small proof. In fact there may be no such strategy [1], and we suggest that a non-systematic approach is an interesting research direction for such proof systems.

Acknowledgements Thanks to Eli Ben-Sasson for advice on resolution, and to the anonymous referees for many helpful comments. This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075, and by Fundação para a Ciência e Tecnologia under research project POSC/EIA/61852/2004.

References

1. M. Alekhnovich, A. Razborov. Resolution is Not Automizable Unless W[P] is Tractable. *Forty-Second IEEE Symposium on FOCS*, 2001, pp. 210–219.
2. L. Baptista, J. P. Marques-Silva. Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. *Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Springer 2000, pp. 489–494.
3. E. Ben-Sasson, A. Wigderson. Short Proofs are Narrow — Resolution Made Simple. *Journal of the ACM* vol. 48 no. 2, 2001, pp. 149–169.
4. E. Ben-Sasson, R. Impagliazzo, A. Wigderson. Near-Optimal Separation of Treelike and General Resolution. *Combinatorica* vol. 24 no. 4, 2004, pp. 585–603.
5. F. Boussemart, F. Hemery, C. Lecoutre, L. Saïs. Boosting Systematic Search by Weighting Constraints. *Sixteenth European Conference on Artificial Intelligence*, IOS Press, 2004, pp. 146–150.
6. M. Davis, H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the Association of Computing Machinery* vol. 7 no. 3, 1960.
7. M. Davis, G. Logemann, D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM* vol. 5, 1962, pp. 394–397.
8. J. L. Esteban, J. Torán. Space Bounds for Resolution. *Information and Computation* vol. 171 no. 1, 2001, pp. 84–97.
9. H. Fang, W. Ruml. Complete Local Search for Propositional Satisfiability. *Nineteenth National Conference on Artificial Intelligence*, AAAI Press, 2004, pp. 161–166.
10. I. P. Gent, H. H. Hoos, A. G. D. Rowley, K. Smyth. Using Stochastic Local Search to Solve Quantified Boolean Formulae. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer, 2003, pp. 348–362.

11. 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.
12. M. L. Ginsberg, D. McAllester. GSAT and Dynamic Backtracking. *International Conference on Principles of Knowledge and Reasoning*, 1994, pp. 226–237.
13. C. P. Gomes, B. Selman, H. Kautz. Boosting Combinatorial Search Through Randomization. *Fifteenth National Conference on Artificial Intelligence*, AAAI Press, 1998, pp. 431–437.
14. J. Gu. Efficient Local Search for Very Large-Scale Satisfiability Problems. *Sigart Bulletin* vol. 3, no. 1, 1992, pp. 8–12.
15. H. H. Hoos. On the Run-Time Behaviour of Stochastic Local Search Algorithms. *Sixteenth National Conference on Artificial Intelligence*, AAAI Press, 1999, pp. 661–666.
16. 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.
17. Y. Interian, G. Corvera, B. Selman, R. Williams. Finding Small Unsatisfiable Cores to Prove Unsatisfiability of QBFs. *Ninth International Symposium on AI and Mathematics*, 2006.
18. H. A. 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.
19. C. M. Li, Anbulagan. Look-Ahead Versus Look-Back for Satisfiability Problems. *Principles and Practice of Constraint Programming, Proceedings of the Third International Conference, Lecture Notes in Computer Science* vol. 1330, Springer-Verlag 1997, pp. 341–355.
20. I. Lynce, J. P. Marques-Silva. Random Backtracking in Backtrack Search Algorithms for Satisfiability. *Discrete Applied Mathematics*, 2006 (in press).
21. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: Engineering an Efficient SAT Solver. *Thirty Eighth Design Automation Conference*, Las Vegas, 2001, pp. 530–535.
22. E. T. Richards, B. Richards. Non-Systematic Search and No-Good Learning. *Journal of Automated Reasoning* vol. 24 no. 4, 2000, pp. 483–533.
23. I. Rish, R. Dechter. Resolution Versus Search: Two Strategies for SAT. *Journal of Automated Reasoning* vol. 24 nos. 1–2, 2000, pp. 225–275.
24. J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery* vol. 12 no. 1, 1965, pp. 23–41.
25. 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.
26. B. Selman, H. Levesque, D. Mitchell. A New Method for Solving Hard Satisfiability Problems. *Tenth National Conference on Artificial Intelligence*, MIT Press, 1992, pp. 440–446.
27. C. Sinz, A. Kaiser, W. Küchlin. Formal Methods for the Validation of Automotive Product Configuration Data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 17 no. 2, special issue on configuration, 2003.