Complete Local Search for Propositional Satisfiability

Hai Fang
Department of Computer Science
Yale University
New Haven, CT 06520-8285
hai.fang@yale.edu

Wheeler Ruml
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
ruml@parc.com

Abstract

Algorithms based on following local gradient information are surprisingly effective for certain classes of constraint satisfaction problems. Unfortunately, previous local search algorithms are notoriously incomplete: They are not guaranteed to find a feasible solution if one exists and they cannot be used to determine unsatisfiability. We present an algorithmic framework for complete local search and discuss in detail an instantiation for the propositional satisfiability problem (SAT). The fundamental idea is to use constraint learning in combination with a novel objective function that converges during search to a surface without local minima. Although the algorithm has worst-case exponential space complexity, we present empirical results on challenging SAT competition benchmarks that suggest that our implementation can perform as well as state-of-the-art solvers based on more mature techniques. Our framework suggests a range of possible algorithms lying between tree-based search and local search.

Introduction

Algorithms for constraint satisfaction problems have traditionally been divided into two classes: those that are based on a systematic tree-structured search and those that use a repair-based local search scheme. Algorithms that explore a tree containing all possible variable assignments can implicitly prove that a problem is unsatisfiable by traversing the entire tree without finding a feasible solution. Methods based on repairing an initially infeasible solution typically follow a local gradient function and do not record which assignments have been visited and which remain untried. Such methods are therefore incomplete: There is no finite time within which they are guaranteed to find a satisfying assignment if one exists. They will fail to terminate on unsatisfiable problems and are not guaranteed to solve a satisfiable problem. Given the surprising effectiveness of local search methods in practice, there is great interest in understanding how tree-based and repair-based methods can be hybridized and in whether it might be possible to design a complete local search algorithm (Selman, Kautz, & McAllester, 1995; Kautz & Selman, 2003).

In this paper, we will show that it is indeed feasible to record information reflecting the past history of a gradient-based search and to use that information to prove the search complete. Although our basic framework applies to any finite-domain constraint satisfaction problem, we will focus in this paper on propositional satisfiability (SAT). In this problem, one must determine whether it is possible to find a satisfying assignment to the variables of a Boolean formula that is presented in conjunctive normal form—a conjunction of clauses, each of which is the disjunction of one or more literals, where a literal is an instance of a variable or its negation. Each clause thus forms a constraint that disallows certain assignments. Many other problems can be translated into SAT, including STRIPS planning, graph coloring, test pattern generation, logic synthesis, equivalence checking, and model checking.

Our approach is based on using a novel objective function to compute the local gradient. When a local minimum is reached, we generate new implied clauses from the current formula. This can be done in many different ways, including resolution. As we discuss in detail below, the new clauses dynamically change the objective function. This process incrementally smooths the search space. We provide a guarantee that when all possible implied clauses have been generated there will be no local minima left. Although this approach suffers from worst-case exponential space complexity, in practice our implementation usually either finds a satisfying assignment or generates the empty clause well before all implied clauses have been generated.

After presenting our framework for complete local search, we will discuss our instantiation of the scheme for SAT. We then demonstrate the performance of our prototype implementation on challenging benchmarks from the 2003 SAT competition. The results indicate that the guarantee of completeness costs little in overall performance: Our solver surpasses all local search solvers from the competition and performs comparably to the best local search solver known.

The Framework

An outline of our approach is given in Figure 1. As in most local search algorithms for SAT, the search proceeds by iteratively toggling the value of a single variable in the current assignment. Unlike many other local search schemes, ours does not fundamentally depend on randomness. The search moves to a neighboring assignment only if it is strictly better according to the current objective function (steps 3–4). If no

CompleteLocalSearch(ϕ)
1. $\alpha \leftarrow$ generate-initial-assignment(ϕ)
2. while α does not satisfy ϕ do
3. if the best neighbor α' is better than α
4. $\alpha \leftarrow \alpha'$
5. else
6. $\gamma \leftarrow$ generate-implied-clause(ϕ,α)
7. if γ is the empty clause
8. return unsatisfiable
9. else
10. $\phi \leftarrow \phi \wedge \gamma$
11. return α

Figure 1: A framework for complete local search.

adjacent assignment is better, we have reached a local minimum (step 5). A new implied clause is then derived from the current formula (step 6). The clause generator must always generate a new implied clause in finite time, but any procedure that meets this criterion can be used. The generator need not use the current assignment when constructing a new clause, although doing so may improve performance. (We will discuss clause generation at length below.) If the empty clause can be derived, the instance is unsatisfiable, otherwise the formula is augmented with the new clause, thereby implicitly changing the objective function.

The objective function compares two assignments on the basis of the clauses that they each violate. It depends crucially on the lengths of the violated clauses. Although this novel formulation may seem a little strange, it will be important in ensuring the completeness of the search. More formally, let ϕ_α denote those clauses of formula ϕ that are violated by assignment α. If ϕ is over n variables, let $\mathcal{D}(\phi_\alpha)$ be the tuple $\langle d_n, \ldots, d_0 \rangle$, where d_i is the number of clauses of length i that are violated by α. Given the current formula ϕ, an assignment α' is better than another assignment α if $\mathcal{D}(\phi_{\alpha'})$ is lexicographically earlier than $\mathcal{D}(\phi_\alpha)$. That is to say $\mathcal{D}(\phi_{\alpha'})$ must have a smaller entry at the leftmost position where the two tuples differ. This might occur, for instance, because the longest clause violated by α' is shorter than the longest clause violated by α. Note that as new clauses of various lengths are added to the formula during the search process, the relative superiority of different assignments can change.

The algorithm is clearly sound: Because the new clauses that are added to the formula do not change the set of satisfying assignments, any assignment that satisfies the current formula and is returned (steps 2 and 11) would have satisfied the original formula. Assuming the clause generator terminates, the search will terminate because there are only a finite number of possible implied clauses ($\leq 3^n$, in fact) and there are only a finite number of strictly improving steps the search can make with a fixed objective function, as would be obtained once all clauses were generated. Of course, SAT is NP-complete (Garey & Johnson, 1991) and our scheme has worst-case exponential time complexity. What is perhaps less obvious is that the algorithm is complete.

Completeness

To show that the algorithm is complete, we need only consider the situation in which all possible implied clauses have already been generated. (Of course, in practice we hope to find a model or generate the empty clause well before this.) We will use ϕ^* to denote this implication closure of the original formula ϕ. If the formula were unsatisfiable, the empty clause would have been generated, so we need only consider the case of a satisfiable formula. We need to show that there are no local minima when using the objective function specified above on ϕ^*. In other words, a neighboring assignment α' that is closer to a solution than the current assignment α will necessarily appear more attractive.

To prove this, we will first need a few helpful supporting ideas. In the traditional view of SAT, each clause excludes a family of states from the solution set. We explore an alternative view of this phenomenon: Each solution excludes the clauses that are incompatible with it from ϕ^*.

Let $\mathcal{U}$ denote the set of all clauses that can be formed using variables appearing in ϕ and let ϕ^- denote those clauses in $\mathcal{U}$ that do not appear in ϕ^*. We can describe the relationship between ϕ^- and partial solutions of ϕ precisely:

Lemma 1 *A clause $\gamma = l_1 \vee \cdots \vee l_k$ appears in ϕ^- if and only if the corresponding negated partial assignment $\overline{\gamma} = \{\overline{l}_1, \ldots, \overline{l}_k\}$ is part of some solution of ϕ.*

Proof: A clause γ appears in the closure ϕ^* if and only if $\overline{\gamma}$ is not part of any solution of ϕ. Therefore, γ is not in ϕ^* iff $\overline{\gamma}$ is part of some solution. If γ is not in ϕ^*, it must be in ϕ^-. □

Next, we need to note a special property of the objective function:

Lemma 2 *For any assignment α, the componentwise addition $\mathcal{D}({\phi^*}_\alpha) + \mathcal{D}({\phi^-}_\alpha)$ will yield the same constant tuple.*

Proof: Note that $\mathcal{D}({\phi^*}_\alpha) + \mathcal{D}({\phi^-}_\alpha) = \mathcal{D}(\mathcal{U}_\alpha)$. But $\mathcal{D}(\mathcal{U}_\alpha)$ does not depend on α. There are $2^i\binom{n}{i}$ possible clauses of length i on n variables. Any complete assignment to n variables will satisfy exactly one of the two possible literals for each variable and thus will violate exactly $1/2^i$ of all possible clauses of length i. So the number of possible clauses of length i that will be violated will be exactly $\binom{n}{i}$ and the tuple $\mathcal{D}({\phi^*}_\alpha) + \mathcal{D}({\phi^-}_\alpha)$ will always equal the binomial distribution. □

Now we can consider the objective function on neighboring assignments and show that there are no local optima when ϕ has expanded to its closure:

Theorem 1 *For any unsatisfying assignment α and satisfiable formula ϕ, any neighbor α' that is closer to the nearest solution will be better than α according to the objective function on ϕ^*.*

Proof: The intuition here is that as we step closer and closer to the nearest solution, larger and larger implied clauses will always become satisfied. For the proof, ϕ^- is simpler to deal with than ϕ^*, so we will proceed by showing that α' is strictly worse than α with respect to ϕ^-. Lemma 2 will then give us the opposite result for ϕ^*.

Without loss of generality, assume that α is of the form $\{l_1, \ldots, l_n\}$, the nearest solution α^* is of the form

```
NeighborhoodResolution(φ,α)
foreach violated clause γ1, oldest first
    foreach literal l in γ1, in random order
        foreach clause γ2 critically satisfied by l̄, oldest first
            γ3 ← resolve γ1 and γ2
            if γ3 ∉ φ then return γ3
return ᾱ
```

Figure 2: A resolution-based implicate generator.

$\{l_1, \ldots, l_k, \bar{l}_{k+1}, \ldots, \bar{l}_n\}$, and α' is $\{l_1, \ldots, l_{n-1}, \bar{l}_n\}$. We first examine $\mathcal{D}({\phi^-}_\alpha)$. Each clause in ${\phi^-}_\alpha$ has the form $\bar{l}_{i_1} \vee \ldots \vee \bar{l}_{i_r}$, since it is violated by α. By Lemma 1, we know that the corresponding $\{l_{i_1}, \ldots, l_{i_r}\}$ are partial solutions. Since α^* is the nearest solution, $\{l_1, \ldots, l_k\}$ must be the longest such partial solution, and therefore all clauses in ${\phi^-}_\alpha$ must have length less than or equal to k. So $\mathcal{D}({\phi^-}_\alpha) = \langle 0, \ldots, 0, d_k, \ldots, d_0 \rangle$.

We will now show that $\mathcal{D}({\phi^-}_{\alpha'})$ is worse. Note that $\bar{l}_1 \vee \ldots \vee \bar{l}_k \vee l_n$ is a clause that will be violated by α'. Because its corresponding negated partial assignment $\{l_1, \ldots, l_k, \bar{l}_n\}$ is a subset of α^*, by Lemma 1 this clause is also in ϕ^-. Therefore $\mathcal{D}({\phi^-}_{\alpha'})$ will be non-zero at the entry for clauses of length $k+1$, making it worse than $\mathcal{D}({\phi^-}_\alpha)$. Since $\mathcal{D}({\phi^*}_\alpha) + \mathcal{D}({\phi^-}_\alpha)$ is a constant (Lemma 2), this means that $\mathcal{D}({\phi^*}_{\alpha'})$ is better than $\mathcal{D}({\phi^*}_\alpha)$. □

We have shown that our complete local search framework is indeed complete provided that the full implication closure of the original formula is eventually generated. At that point, the fitness landscape will have no local minima and the search procedure will head directly to the nearest solution. (If two neighbors are equally promising, random tie-breaking can be used to choose between them.) We might hope, however, that by choosing new clauses carefully, we can often avoid generating most of the closure while still benefiting from the smoothing effect.

An Instantiation

Within this complete local search framework, many different algorithms can be developed. The most important choice is probably the method of generating new clauses. As long as it eventually generates all implied clauses, the search is guaranteed to be complete (Theorem 1), but the choice of method can greatly influence the efficiency of the resulting algorithm. For example, generating long clauses violated by the current assignment can quickly cause changes in the objective function, allowing escape from a local minimum. However, such specific clauses may not be useful during subsequent search. For unsatisfiable instances, a bias toward short clauses may be preferred, in an attempt to generate the empty clause quickly. This also minimizes the size of the expanding formula. The most promising technique we have evaluated so far is based on resolution.

Neighborhood Resolution

In resolution, clauses such as $x \vee C_1$ and $\bar{x} \vee C_2$ are combined to yield $C_1 \vee C_2$. Neighborhood resolution (Cha & Iwama, 1996) refers to the special case in which there is exactly one variable that appears positively in one clause and negatively in the other. This guarantees that the result will not be rendered useless by the presence of $x \vee \bar{x}$. The particular variable and clauses that are used can be decided in many ways. In our current implementation (Figure 2), we select a random literal from the oldest violated clause (that is, whose status changed longest ago). We then select the oldest satisfied neighboring clause that depends crucially on the negation of that literal for its satisfaction. In the exceedingly rare case in which a new resolvent cannot be generated, the clause corresponding to the negation of the current assignment is returned. In either case, the new clause is guaranteed to be violated by the current assignment and is therefore potentially helpful for escaping from the current minimum.

To prove that this procedure can always generate a new clause, and thus will preserve the completeness of the search, we need only check its behavior at local minima:

Theorem 2 *Let α be a local minimum on a formula ϕ which already contains the negation of α (call it $\bar{\alpha}$). If ϕ does not already contain the empty clause, then neighborhood resolution will generate a new resolvent.*

Proof: We will proceed via the contrapositive, showing that if additional resolvents cannot be generated, the empty clause is present. Let $\mathcal{D}_i(\phi_\alpha)$ be the number of clauses of length i in ϕ that are violated by α. Note that the clause $\bar{\alpha}$ has length n, so $\mathcal{D}_n(\phi_\alpha) = 1$, its maximum possible value. Because α is a local minimum, $\mathcal{D}_n(\phi_{\alpha'}) = 1$ for all neighboring assignments α'. This means that the clauses $\overline{\alpha'}$ must also be in ϕ. A chain reaction is thus initiated. If neighborhood resolution cannot generate any new resolvents, then ϕ must already contain the n clauses of length $n-1$ that neighborhood resolution can generate from resolving $\bar{\alpha}$ with the various $\overline{\alpha'}$. These shorter clauses are subsets of $\bar{\alpha}$, so $\mathcal{D}_{n-1}(\phi_\alpha) = n$. Again, because we are at a local minimum, $\mathcal{D}_{n-1}(\phi_{\alpha'}) = n$ for all α'. For each α', all n clauses resulting from dropping a single literal from $\overline{\alpha'}$ must be in ϕ. The chain reaction continues, with neighborhood resolution providing all $\binom{n}{i}$ subsets of $\bar{\alpha}$ of length i and the local minimum property implying the presence of the additional clauses necessary for producing still shorter clauses. Continuing in this manner, we see that the empty clause must already be present in ϕ. □

Other Features

Although the algorithm described above yields a respectable SAT solver, additional techniques can be incorporated into the algorithm to improve performance. To allow a meaningful comparison to existing, highly-tuned solvers, the implementation evaluated below included these standard enhancements:

unit propagation Whenever a unit clause is generated, the corresponding variable can be set and eliminated. Adding this feature improved performance on handmade and industrial instances (these benchmarks are described below).

equivalent literal detection If $l_1 \vee \overline{l_2}$ and $\overline{l_1} \vee l_2$ are both present, l_2 can be replaced everywhere by l_1. This limited form of equivalency reasoning seemed to improve performance on parity checking and many industrial instances.

resolution between similar clauses If two clauses differ only in the polarity of a single literal, their resolvent is generated immediately. In our implementation, it is easy to check for similar clauses in a hashtable, and obtaining these shortened clauses seemed to provide a slight improvement.

appropriate data structures Violated and critical clauses are held in doubly-linked lists, allowing constant-time removal as well as tracking clause age (for clause selection during resolution). The critical clause lists are indexed by variable. For each variable, two hashtables hold the clauses in which it occurs positively and negatively. To choose the best neighbor during hill-climbing, the improvement resulting from flipping each variable is calculated explicitly. For efficiency, variables whose attractiveness can only have decreased are not checked. After each resolution, only the variables involved are checked.

Empirical Evaluation

To assess whether complete local search is of practical as well as theoretical interest, we tested our implementation using the instances and protocol of the 2003 SAT Competition (Simon, 2003; Hoos & Stützle, 2000). This allows us to compare against many state-of-the-art solvers using a broad collection of challenging and unbiased instances. The 996 competition instances are divided into three categories: random problems (produced by a parameterized stochastic generator), handmade problems (manually crafted to be both small and difficult), and industrial problems (encodings of problems from planning, model checking, and other applications). Within each category, closely related instances are grouped into a single 'series'. To minimize the effect of implementation details and emphasize breadth of ability, solvers are ranked according to how many different series they can solve, where a series is considered solved if at least one of its instances is solved. Solvers incorporating randomness are given three separate tries on each instance. Prizes are awarded for best performance in each category, with separate awards for performance on all instances and performance on satisfiable instances only. Prize-winning solvers from the 2002 competition were automatically entered in 2003. In the competition, each solver was allocated 900 seconds on an Athlon 1800+ with 1Gb RAM (a 2.4Ghz Pentium 4 was used for instances in the random family). We attempted to match these conditions in our tests, normalizing across machines using the DIMACS dfmax utility (and SPECint2000 results for the Athlon) and imposing a 900Mb memory limit.[1]

Table 1 presents the performance of complete local search (CLS) together with many state-of-the-art solvers, both in terms of the number of series solved and the number of instances. All data except those for CLS and RSAPS were copied from the published competition results. The first four rows represent all of the local search solvers from the competition. Unitwalk (Hirsh & Kojevnikov, 2003) was the best 2003 entrant on satisfiable random instances. Results of CLS are presented both with and without unsatisfiable instances. RSAPS (Hutter, Tompkins, & Hoos, 2002), while not entered in the competition, represents one of the best current local search solvers, if not the best.[2] It uses a clause weighting scheme that is carefully engineered for efficiency. The remaining seven rows represent complete tree search-based solvers, and include the other prize winners from 2003 and 2002.[3] In several cases, the same solver was best both overall and on only satisfiable instances.

The results convincingly demonstrate that complete local search is of practical interest. CLS surpasses all of the local search-based solvers from the 2003 competition on the primary competition measure, number of series solved, both overall and in each category, as well as on the number of instances solved overall. Even considering satisfiable instances alone, CLS solved more random series and a similar number of series overall. Had it entered the competition, CLS would have garnered best solver on satisfiable random instances. Its ability to prove unsatisfiability allows it to substantially surpass other local search solvers on industrial instances. Compared with RSAPS, CLS solved fewer random series but more series overall and a comparable number of instances overall.

Overall, CLS behaves more like a local search solver than a tree-based solver. It does better on satisfiable random instances than any tree-based solver, but it does not exhibit the overall performance of the best tree-based solvers.

The time CLS takes per flip increases as the size of the clause database grows during search. Techniques developed for other solvers, such as subsumption testing and forgetting irrelevant clauses, may be useful for managing this growth and reducing the time per flip.

Randomness is exploited by the algorithm in three places: to construct the initial assignment, to break ties between equally promising neighbors during hill-climbing, and to select variables to resolve on.

Related Work

Constraint learning is a popular technique in tree-based search (Bayardo Jr. & Schrag, 1996), where it has been shown to increase the formal power of the proof system (Beame, Kautz, & Sabharwal, 2003). Usually, only clauses that are short or relevant to the current portion of the search tree are retained (Moskewicz *et al.*, 2001). In

[1] In cases in which we did not use a 2.4Ghz Pentium 4, we used 724 seconds on a 2.0 GHz Xeon, and 557 seconds on a 2.6Ghz Xeon.

[2] Tests on smaller instances (Hutter, Tompkins, & Hoos, 2002) suggest that RSAPS surpasses the performance of Novelty+ (Hoos, 1999) and ESG (Schuurmans, Southey, & Holte, 2001), and therefore also WalkSAT (Selman, Kautz, & Cohen, 1994) and DLM (Wu & Wah, 2000). We ran `saps-1.0` ourselves with default parameters and the `-reactive` (self-tuning) flag.

[3] Oksolver was not run on handmade or industrial problems in the 2003 competition.

	Random		Handmade		Industrial		Total		
	ser.	inst.	ser.	inst.	ser.	inst.	ser.	inst.	
amvo	0	0	1	10	1	2	2	12	
qingting	9	73	3	12	4	17	16	102	
saturn	14	127	7	29	3	22	24	178	
unitwalk	15	139	5	20	4	15	24	174	best sat. random ’03
cls	16	121	9	22	14	66	39	209	
cls (sat only)	16	121	5	17	2	15	23	153	
rsaps	21	177	8	26	1	9	30	212	
kcnf	25	163	16	51	6	17	47	231	best random ’03
satzoo1	11	67	21	102	31	124	63	293	best handmade ’03
forklift	11	53	17	63	34	143	62	259	best industrial ’03
oksolver	16	96	na	na	na	na	na	na	best random ’02
berkmin561	10	54	16	65	32	136	58	255	best sat. handmade ’02
zchaff	10	53	17	67	29	115	56	235	best industrial, handmade ’02
limmat	5	24	14	52	28	114	47	190	best sat. industrial ’02

Table 1: Performance on the 2003 SAT Competition instances: number of series and instances solved.

local search, techniques that associate a weight with each clause and change those weights over time can simulate the effect of adding duplicate clauses (Morris, 1993; Shang & Wah, 1997). The RSAPS solver takes this approach. Such methods can be seen as attempting to approximate the ideal search surface that complete local search builds. Explicitly adding implied clauses was suggested by Cha & Iwama (1996). They showed how implied clauses could provide improvements over duplicate clauses (see also Yokoo, 1997), although their method does not guarantee completeness. Morris (1993) noted that explicitly recording and increasing the cost of visited states can force an algorithm to eventually solve a satisfiable instance, although his scheme cannot easily detect unsatisfiability.

There have been several recent proposals for hybrids of local and tree-based search. To our knowledge, none achieves completeness without embedding the local search in a tree-like framework. For example, both Mazure, Saïs, & Grégoire (1998) and Eisenberg & Faltings (2003) propose schemes in which local search is used to determine a variable ordering for subsequent tree search.

The weak-commitment search strategy of Yokoo (1994) and the Learn-SAT algorithm of Richards & Richards (2000) are perhaps the previous proposals most similar to complete local search. In these closely related procedures, only the first branch of a search tree is explored. Variable choice is guided in part by a current infeasible assignment. At the first dead-end, a new clause is learned and the search starts again from the root. As in our approach, this nogood learning confers completeness. However, both weak-commitment search and Learn-SAT are fundamentally constructive, tree-like, search methods. Many variables can change their values between one construction episode and the next, and unit propagation lets one variable's value directly dictate the values of several others. In contrast, complete local search is based on hill-climbing with gradient information, always changing exactly one variable assignment at a time. In this sense, the steps of the search are restricted to 'local' changes to one part of the assignment. As in our experiments, Richards & Richards did not find the worst-case exponential space complexity of their method a hindrance in practice. Unfortunately, we cannot easily compare Learn-SAT with complete local search because all published results are in terms of constraint checks and node counts, which are meaningless for CLS, and its source code does not seem to be available on-line.

Possible Extensions

There are many ways in which this work might be extended. Other strategies to generate new clauses may be useful. General resolution is powerful, but by itself doesn't suggest which clauses should be resolved. Two interesting alternatives to neighborhood resolution are conflict analysis and clause splitting. By conflict analysis, we mean a procedure that incrementally constructs a new assignment, exploiting unit propagation as in a tree search. When the partial assignment cannot be extended, the corresponding clause can be learned (Bayardo Jr. & Schrag, 1996). Such a procedure can perform the equivalent of many resolution steps, and can take advantage of variable choice heuristics developed for tree search algorithms. While this is the major search step in weak-commitment search and Learn-SAT, it can also be used simply as a clause generation technique in support of local search. Our preliminary experience with such a procedure is promising, but not yet as successful as with neighborhood resolution.

In clause splitting, a violated clause C gives rise to $C \vee x_i$ and $C \vee \overline{x}_i$, where x_i doesn't already appear in the clause. Whichever new clause is violated can be usefully learned, and provides a succinct way to make the current minimum less attractive. This method can be made complete when used with resolution between similar clauses, which we mentioned above.

If extended effort in one part of the search space does not yield a solution, there is no reason why the search cannot be restarted at another assignment. As long as the learned clauses are retained, completeness will not be sacrificed.

Preliminary evidence suggests that this approach could result in substantial gains. Using a greedy heuristic to generate the initial assignment might also improve performance, especially when used with a restarting strategy.

It should be possible to use a variation of complete local search to find all possible models of a formula. When a solution α is found, the negated clause $\overline{\alpha}$ can be added to the formula to force the search to another nearby solution, until eventually the formula becomes unsatisfiable.

Our framework applies directly to other constraint satisfaction problems. The size of the closure is increased, and constraint generation may become more complex, but the completeness proof still holds.

Conclusions

We have presented the first local search algorithm that has been proved complete. This answers one of the major outstanding problems in propositional reasoning and search (Selman, Kautz, & McAllester, 1995; Kautz & Selman, 2003). The algorithm is fundamentally based on following gradient information, rather than relying on an external tree search for completeness. Constraint learning ensures completeness, but does not restrict the motion of the search. Although the completeness proof assumes that all implied clauses have been generated, this is far from necessary in practice. By testing an implementation of the method on challenging SAT instances, we have shown that the completeness guarantee does not hinder the empirical performance of the algorithm compared to other local search solvers. On the contrary, due in part to its ability to prove unsatisfiability, complete local search surpassed all the local search solvers from the 2003 SAT competition and performed comparably to RSAPS, one of the best local search solvers known.

Complex resolution and constraint learning techniques can easily be incorporated into our framework, raising the possibility of mimicking much of the reasoning done by current tree-based solvers. Complete local search suggests that the division between tree-based search and local search may be much more porous than commonly believed.

Acknowledgments

We thank Jimmy H. M. Lee, Valeria de Paiva, and Johan de Kleer for helpful discussions and comments on a preliminary draft of this work, Wen Xu for help improving the C implementation of CLS, and Dave Tompkins for providing the source code for RSAPS. Much of this research was done while the first author was an intern at PARC. This work was supported in part by the DARPA NEST program under contract F33615-01-C-1904.

References

Bayardo Jr., R. J., and Schrag, R. 1996. Using CSP look-back techniques to solve exceptionally hard SAT instances. In *Proceedings of CP-96*, 46–60.

Beame, P.; Kautz, H.; and Sabharwal, A. 2003. Understanding the power of clause learning. In *Proceedings of IJCAI-03*, 1194–1201.

Cha, B., and Iwama, K. 1996. Adding new clauses for faster local search. In *Proceedings of AAAI-96*, 332–337.

Eisenberg, C., and Faltings, B. 2003. Making the breakout algorithm complete using systematic search. In *Proceedings of IJCAI-03*, 1374–1375.

Garey, M. R., and Johnson, D. S. 1991. *Computers and Intractability*. New York: W.H. Freeman and Co.

Hirsh, E. A., and Kojevnikov, A. 2003. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. In *Proceedings of SAT-02*, 35–42.

Hoos, H. H., and Stützle, T. 2000. SATLIB: An online resource for resesarch on SAT. In Gent, I.; van Maaren, H.; and Walsh, T., eds., *SAT 2000*. IOS Press. 283–292.

Hoos, H. H. 1999. On the run-time behavior of stochastic local search algorithms for SAT. In *AAAI-99*, 661–666.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of CP '02*, 233–248.

Kautz, H., and Selman, B. 2003. Ten challenges redux: Recent progress in propositional reasoning and search. In *Proceedings of CP '03*, 1–18.

Mazure, B.; Saïs, L.; and Grégoire, E. 1998. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence* 22:319–331.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings of AAAI-93*, 40–45.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: engineering an efficient SAT solver. In *Proceedings of DAC-01*, 530–535.

Richards, T., and Richards, B. 2000. Nonsystematic search and no-good learning. *J. Auto. Reasoning* 24(4):483–533.

Schuurmans, D.; Southey, F.; and Holte, R. C. 2001. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of IJCAI-01*, 334–341.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *AAAI-94*, 337–343.

Selman, B.; Kautz, H.; and McAllester, D. 1995. Ten challenges in propositional reasoning and search. In *Proceedings of IJCAI-95*, 50–54.

Shang, Y., and Wah, B. W. 1997. A discrete Lagrangian-based global-search method for solving satisfiability problems. *Journal of Global Optimization* 10:1–40.

Simon, L. 2003. The SAT-03 contest results site. http://www.lri.fr/~simon/contest03/results/.

Wu, Z., and Wah, B. 2000. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of AAAI-00*, 310–315.

Yokoo, M. 1994. Weak-commitment search for solving constraint satisfaction problems. In *Proc. AAAI-94*, 313–318.

Yokoo, M. 1997. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Proceedings of CP-97*, 356–370.