Local Search for Unsatisfiable Propositional Formulae

David Pereira

Instituto Superior Técnico, Universidade Técnica de Lisboa, Portugal
david.pereira@tagus.ist.utl.pt

Abstract. Stochastic local search (SLS) has been an active field of research in the last few years, with new techniques and procedures being developed at an astonishing rate. SLS has been traditionally associated with satisfiability solving, that is, finding solutions for a set of problems. However, their intrinsic nature does not allow them to address unsatisfiable problems. Selman, Kautz and McAllester proposed a challenge to use local search instead to prove unsatisfiability. In this document we give an extensive overview on how to improve the state of the art SLS unsatisfiability solver RANGER using GUNSAT’s techniques, namely unit propagation look-ahead and extended resolution.

1 Introduction

Suppose you have to plan the assignment of some rooms in a conference center, in a very busy week, where a lot of guest speakers will make their speeches. You must plan the hours; who speaks in which room; some speakers have tight schedules and can only stay for a short time; the audience cannot be made to change rooms every hour, so the speakers of a particular issue must be grouped in the same room in an interval of time; and so on. All these constraints can be easily solved for a small number of variables: 2-3 speakers, 2 rooms, 1 day, among many others. But if the number of variables increase, even in a small scale, the problem easily becomes hard to solve.

This kind of problems can be encoded using propositional logic. Propositional logic encodes/represents propositions and their corresponding relations. We want to assume that a proposition (or condition) is either true or false, and so this logic can also be called Boolean logic because the propositions can be assigned truth values.

There are many problems that can be formulated as SAT (from SATisfiability), such as schedule planning [6] (the example given above), graph coloring [4], circuit planning and testing [2], blocks world [5], quasigroup problems [12], tower of Hanoi [6], etc.

Combinatorial problems can be found in various areas of computer science and other fields, such as artificial intelligence, bioinformatics, schedule planning, microprocessor planning and testing, etc. These problems usually involve finding groupings or assignments of a finite and discrete set of objects that satisfy certain
constraints or conditions. The solution to combinatorial problems is formed by combinations of these solution components. The example above, about scheduling, can be viewed as a combinatorial problem in which the components that form the solution are the events to be scheduled, and the values assigned to the events represent the time at which they occur. We can then define candidate solution as a set of assignments of Boolean values to the variables; this candidate solution must then be evaluated by some function to see if it is indeed a solution of the problem, such that all conditions are satisfied.

Many combinatorial problems can be said to be decision problems, in which the solutions to a given instance of the problem are specified by a logical set of conditions. This particular kind of combinatorial problems will be explored in more detail in the following section.

Another kind of problems is optimization problems. Many of the more practically relevant combinatorial problems are optimization problems rather than decision problems. For each optimization problem, there is a corresponding decision problem that asks whether there is a feasible solution for some particular set of assignments. The goal is to find the best possible solution, and an objective function is used to measure and evaluate all candidate solutions. Some examples of optimization problems are the minimization problem and the maximization problem.

There are two very important complexity classes: $P$ and $NP$. The first is the class of problems that can be solved by a deterministic machine in polynomial time, whereas the second is the class of problems that can be solved by a non-deterministic machine in polynomial time. Every problem in $P$ is contained in $NP$. Many hard problems from $NP$ are closely related and can be translated into each other in polynomial deterministic time. A problem that is at least as hard as any other problem in $NP$ is called $NP$-hard. But these problems do not have to belong to the $NP$ class themselves, because their complexity may be higher. $NP$-hard problems that are contained in $NP$ are called $NP$-complete.

SAT was the first decision problem to be proved to be a $NP$-complete problem. This was done by Stephen Cook in 1971 [3].

This document is organized as follows. In chapter 2 we discuss the two most prominent local search algorithms for proving unsatisfiability: RANGER and GUNSAT. At the end of this chapter we focus on the advantages and disadvantages of both of them.

We then follow, in chapter 3, with a detailed explanation of the techniques and methods used to improve the original RANGER, namely unit propagation look-ahead and extended resolution.

In the fourth chapter, we present the results achieved with the various methods experimented on RANGER and draw some conclusions about the usefulness of such procedures in a solver such as RANGER.

Finally in chapter 5 we draw conclusions about this thesis and suggest some future work.
Local Search for Unsatisfiability

Selman, Kautz and McAllester posed an intriguing challenge in 1997 to use local search to prove unsatisfiability instead of satisfiability [10]. In 2006 and 2007 two different approaches were proposed in response to that challenge [8, 1]. These two algorithms use local search to prove unsatisfiability but not satisfiability, still being incomplete algorithms.

2.1 RANGER

RANGER [8] stands for RANdomized GEneral Resolution and was presented in 2006 as the first SLS algorithm that can prove unsatisfiability instead of satisfiability. It explores a space of multisets of resolvents using general resolution and aims at deriving the empty clause non-systematically but greedily, and thus proving unsatisfiability. RANGER will eventually refute any unsatisfiable instance while using only bounded memory.

Intuitively, each \( \varphi_i \) 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 (such are the clauses that are subsumed by some other clauses, for example).

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 in RANGER. However, in practice, it may succeed even if the resolvent length is restricted to a smaller value, which will save memory on large problems.

Each \( \varphi_i \) will be of the same constant size, and derived from \( \varphi_{i-1} \) by the application of resolution or the replacement of a clause by one taken from \( \varphi \).

The RANGER algorithm begins by choosing any \( k \) clauses from the formula \( \varphi \) into \( \varphi_1 \). It then performs iterations \( i \), either replacing a \( \varphi_i \) clause with a \( \varphi \) clause (with probability \( p_i \)) or resolving two \( \varphi_i \) clauses and placing the result \( r \) into \( \varphi_i \). In the latter case, if \( r \) is a tautology or contains more than \( w \) literals then it is discarded and \( \varphi_{i+1} = \varphi_i \). Otherwise a \( \varphi_i \) 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 \( r \) or it is randomly chosen. In the former case, if \( r \) is longer than the parent then \( r \) is discarded and \( \varphi_{i+1} = \varphi_i \). With probability \( p_r \), any satisfiability-preserving transformation may be applied to \( \varphi, \varphi_i \) or both. One can apply subsumption and the pure literal rule in several ways as satisfiability-preserving transformations. If the empty clause has been derived then the algorithm returns the message UNSATISFIABLE, otherwise it may not terminate. A time-out condition may be added to restrict the CPU time that the algorithm is allowed to run.

In this algorithm the goal is to derive the empty clause, and as such \( \varphi_i \) must contain some small clauses. This is controlled by the level of greediness (probability \( p_g \)). A greedy local move is one that does not increase the number of literals in \( \varphi_i \). So, increasing \( p_g \) will increase the greediness of the search, reducing the proliferation of large resolvents.
2.2 GUNSAT

GUNSAT [1] proposes to make a greedy walk through the resolution search space in which, at each iteration of the algorithm, it tries to compute a better neighbouring set of clauses (a set of clauses similar to the previous one, with only minor variations on a few clauses), differing from the previous one by at most two clauses: one added by resolution and one that may have been removed. A score is given to all pairs of literals based on their frequency appearance in the formula. GUNSAT also makes use of higher reasoning mechanisms, based on extended resolution [9] and unit propagation look-ahead [7], that are key to make this approach effective.

It was very important to add three powerful mechanisms to GUNSAT in order to make it competitive with other resolution-based reasoning systems: binary clause saturation, unit propagation look-ahead with pairs of literals and extended resolution. The first but essential refinement concerns subsumptions. Before adding a new clause, the algorithm performs a forward/backward subsumption detection [13].

The extended resolution [11] is used when the algorithm has tried to increase the score of a given pair of literals too many times without success, and it uses extended resolution to artificially increase that score. The extended rule \( e \iff l_1 \lor l_2 \) is encoded by the three clauses \( (\neg e \lor l_1 \lor l_2) \), \( (e \lor \neg l_1) \) and \( (e \lor \neg l_2) \). These three clauses and the new variable \( e \) can then be added to the formula.

Look-ahead techniques are used to detect equivalences between literals until an inconsistency is found. It uses a reformulation of the initial formula, based on a set of triplets \( (p \iff q \iff r) \) and \( (p \iff q \Rightarrow r) \) only. GUNSAT uses look-ahead unit propagation on pairs of literals, such that the four possible pairs of values are propagated in \( \varphi \), searching for more unit propagations.

2.3 RANGER vs. GUNSAT

There are some few important differences between these two local search algorithms for proving unsatisfiability. RANGER generates a large number of the shortest possible clauses as fast as possible, using non intelligent local moves, whereas GUNSAT takes longer to make more intelligent moves based on a more complex objective function. GUNSAT also uses higher reasoning techniques like extended resolution and unit propagation look-ahead (RANGER uses only general resolution). As such, GUNSAT has a more powerful reasoning mechanism and a more refined heuristic to guide moves, whereas RANGER is simpler but performs many more moves per second. Also, unlike GUNSAT, RANGER uses a mechanism to ensure bounded memory.

3 Improving RANGER

The algorithm developed as a consequence of the investigation work done for this dissertation is based on the original solver by S. Prestwich and I. Lynce and dubbed ranger_release.
It was kindly yielded to serve as a starting point for the achievement of the objectives proposed in this thesis: the development of a stochastic local search solver that could prove unsatisfiability and that combined the most prominent elements of both RANGER [8] and GUNSAT [1], the current state-of-the-art SLS solvers for unsatisfiability.

3.1 RANGER with Unit Propagation Look-Ahead

The solver GUNSAT successfully uses a method dubbed Unit Propagation Look-Ahead [1] to improve its basic algorithm. The literals of the formula under consideration are extensively checked to see if there are any conflicts arising from assignments.

GUNSAT implements a version of the unit propagation look-ahead which uses two variables. These two variables are then assigned a value such that the four possible combinations are covered. If \( v_1 \) and \( v_2 \) are our variables, then the four possible combinations are:

- \( v_1 = 0 \) and \( v_2 = 0 \);
- \( v_1 = 0 \) and \( v_2 = 1 \);
- \( v_1 = 1 \) and \( v_2 = 0 \);
- \( v_1 = 1 \) and \( v_2 = 1 \).

Let us define a conflict as a situation where, given an assignment for both variables, the formula evaluates to false. For each iteration of this look-ahead method, i.e., for each combination of variable assignments, we store the value of each variable in the formula, only if that variable is forced to be assigned (note that these assignments are only temporary, done for each iteration of the look-ahead and stored only for the duration of the look-ahead for the two variables).

Let us define the intersection over a variable as the set of assignments to that variable, after all the combinations of the look-ahead method are executed, but before the unit propagation phase begins.

According to the number of conflicts after the application of the look-ahead technique, we have five possible scenarios for the unit propagation phase:

- **Zero conflicts**: if there are no conflicts, we will consider all four combinations when deciding the intersections; if a variable is assigned the same value through all combinations, then that variable will be propagated and the unit clause rule will be applied; this result is applied to all the following cases excluding the last one;
- **One conflict**:
  - the intersections will be calculated, but only considering the combinations that did not yield a conflict (three in this case);
  - a binary clause is added to the formula: this clause results from the negation of the assignments that yielded a conflict;
- **Two conflicts**:
  - as above, the intersections will be calculated, but only considering the combinations that did not yield a conflict (two in this case);
two binary clauses are added to the formula, resulting each one from the negation of the assignments that yielded a conflict; there is a special case where only a unit clause is added, which happens when a variable assignment is repeated in both conflicts (the two binary clauses are resolved to yield the unit clause);

- Three conflicts:
  - the variables that were assigned in the only combination that did not yield a conflict will be propagated;
  - two unit clauses are added, each one with each variable assignment that did not yield a conflict;
- Four conflicts: the formula is unsatisfiable for all combinations, which means the formula is unsatisfiable.

One of the objectives of this dissertation is to successfully integrate features of the GUNSAT algorithm into the RANGER algorithm. We did not want to modify the most important properties of the original RANGER, nor alter its flow. The unit propagation look-ahead was, thus, added to the section of satisfiability-preserving transformations. The probability, $P_t$, to execute these transformations is 90%, and like the other transformations, unit propagation will be executed, on average, in 90% of the iterations of the algorithm.

This addition is divided into two parts: unit propagation look-ahead and unit propagation only. The first is run only once, during the first time that satisfiability-transformations are executed, due to the weight it has on the performance. The look-ahead with two variables, over all variables in the formula, can be very heavy and deteriorate the performance of the algorithm for very little gain in return.

The second part of this addition is unit propagation without look-ahead, and this is executed in every satisfiability-preserving transformation. The details of both methods will be discussed shortly.

Let it be noted that these methods can prove the unsatisfiability of a formula by themselves: if, during the look-ahead method, four conflicts arise, then the formula is unsatisfiable; likewise, if during the unit propagation phase, a conflict is detected, the function also returns UNSAT.

3.2 RANGER with Extended Resolution

According to the authors of GUNSAT [1], extended resolution is a method which gives very good results in practice when added to their solver. When the algorithm has tried to increase the heuristic score of a given pair of literals too many times without any success, extended resolution is used to artificially increase the score of this pair of literals.

But the way extended resolution is used in GUNSAT is intrinsically related to the algorithm itself, built to take advantage of its scoring system. Note that extended resolution is only used when the algorithm has tried to increase the score of a pair of literals too many times without success. It is very different from the way RANGER operates, where no scoring system for literals is used. Thus,
we have to add GUNSAT’s scoring system to RANGER if we want to implement extended resolution in the same way it was successfully used in GUNSAT.

But there is a problem with this approach. GUNSAT’s scoring system for pairs of literals is part of the main heuristic of the solver. It was developed to be the backbone of the algorithm and to be a highly refined heuristic of scoring and choosing the best literals and clauses to resolve. Methods like extended resolution, unit propagation look-ahead or binary clause saturation are only meant to improve this heuristic.

As has been said before, RANGER does not have a scoring scheme for literals like GUNSAT does, so that extended resolution could be applied directly to that scheme and be integrated seamlessly with the algorithm. Instead, we chose to adapt the scoring method of GUNSAT to RANGER and thus apply the extended resolution in the same way GUNSAT does.

Any instance of extended resolution is executed only during the satisfiability-preserving transformations phase of the RANGER algorithm, in the same way as unit propagation look-ahead, and for the same reasons. We did not want to alter RANGER’s base concept and program flow. Furthermore, besides the probability \( P_t \) of this phase of the algorithm, we inserted another probability \( P_{cr} \), and the steps of extended resolution will only be executed according to this probability, set to \( P_{cr} = 5\% \), which means it will be executed in about 4.5% of the iterations (calculated by multiplying the 90% chance that the satisfiability-preserving transformations phase will be run and the 5% chance that extended resolution will be executed within that phase).

4 Experimental Evaluation

Our results were obtained in an Intel Xeon 5160 server (3.0GHz, 1333Mhz, 4GB) running Red Hat Enterprise Linux WS 4. Each problem instance was run with 10 seeds and for a timeout of 1000 seconds.

To illustrate the behaviour of RANGER, a set of experimental results was compiled in the following tables. We utilized the number of seconds each instance takes to be solved by a given tool as a measurement (time) as well as the number of iterations it took to solve the problem. To better illustrate the usefulness of each component added to RANGER, each problem instance was run with different versions of RANGER: the original RANGER code as a basis for comparisons; the modified RANGER without unit propagation look-ahead or extended resolution; RANGER with unit propagation look-ahead; and RANGER with both unit propagation look-ahead and extended resolution. Results for GUNSAT were also collected.

Tables 1 and 2 show the results of, respectively, the CPU time taken to solve each instance and the number of iterations needed by each solver.

From these tables, we conclude that the best solver to solve easier instances with few variables and a low ratio of variables/clauses is the original RANGER due to its simplicity. In all the other instances of aim, RANGER with unit propagation look-ahead beats the other two solvers in both time taken and iterations.
Table 1. Comparison of time results for the solvers tested

<table>
<thead>
<tr>
<th>Instance</th>
<th>original</th>
<th>modified</th>
<th>UPLA</th>
<th>UPLA+ER</th>
<th>GUNSA T</th>
</tr>
</thead>
<tbody>
<tr>
<td>aim – 50 – no – 1.6</td>
<td>0.015</td>
<td>0.126</td>
<td>0.096</td>
<td>0.281</td>
<td>1.21</td>
</tr>
<tr>
<td>aim – 50 – no – 2.0</td>
<td>250.026</td>
<td>250.271</td>
<td>0.114</td>
<td>0.384</td>
<td>1.41</td>
</tr>
<tr>
<td>aim – 100 – no – 1.6</td>
<td>0.834</td>
<td>5.75</td>
<td>0.91</td>
<td>13.57</td>
<td>36.96</td>
</tr>
<tr>
<td>aim – 100 – no – 2.0</td>
<td>11.35</td>
<td>11.03</td>
<td>1.74</td>
<td>24.09</td>
<td>3.27</td>
</tr>
<tr>
<td>aim – 200 – no – 1.6</td>
<td>259.227</td>
<td>285</td>
<td>24.56</td>
<td>317.1</td>
<td>16.72</td>
</tr>
<tr>
<td>aim – 200 – no – 2.0</td>
<td>512.156</td>
<td>567.01</td>
<td>238.7</td>
<td>818.2</td>
<td>379.8</td>
</tr>
<tr>
<td>uuf.50</td>
<td>845.6</td>
<td>-</td>
<td>521.6</td>
<td>740.1</td>
<td>67.39</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the number of iterations for the solvers tested

<table>
<thead>
<tr>
<th>Instance</th>
<th>original</th>
<th>modified</th>
<th>UPLA</th>
<th>UPLA+ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>aim – 50 – no – 1.6</td>
<td>39223</td>
<td>24076</td>
<td>4201</td>
<td>4958</td>
</tr>
<tr>
<td>aim – 50 – no – 2.0</td>
<td>67105</td>
<td>74452</td>
<td>6240</td>
<td>7026</td>
</tr>
<tr>
<td>aim – 100 – no – 1.6</td>
<td>433398</td>
<td>414130</td>
<td>134572</td>
<td>120122</td>
</tr>
<tr>
<td>aim – 100 – no – 2.0</td>
<td>3268605</td>
<td>3717396</td>
<td>160971</td>
<td>174131</td>
</tr>
<tr>
<td>aim – 200 – no – 1.6</td>
<td>5902875</td>
<td>5320123</td>
<td>1390580</td>
<td>1788612</td>
</tr>
<tr>
<td>aim – 200 – no – 2.0</td>
<td>9802581</td>
<td>9420952</td>
<td>3066713</td>
<td>3762236</td>
</tr>
<tr>
<td>uuf.50</td>
<td>174752209</td>
<td>-</td>
<td>43620471</td>
<td>51390176</td>
</tr>
</tbody>
</table>

(For those that provide us with such information). For the uuf50 set of instances, GUNSAT is far superior to RANGER in any of its forms.

5 Conclusions and Future Work

Propositional Satisfiability is fundamental in solving many application problems in artificial intelligence and also in other fields of computer science and engineering. Recent years have have seen the proposal of several effective approaches for solving SAT instances. In particular, it is important to know whether a problem instance is satisfiable or not. In this thesis, we have presented new methods which draw on the strengths of two state of the art SLS solvers which prove unsatisfiability instead of satisfiability. Our approach was to use the RANGER solver as a base and extend it with the most effective methods used by the GUNSAT solver, and thus produce a new solver that is more than the sum of RANGER and GUNSAT.

To preserve RANGER’s simplicity we did not alter its algorithmic flow and instead added the new techniques to a phase of the algorithm that already supported the so called satisfiability-preserving transformations. These new methods, which were used by GUNSAT to achieve very good results and improvements, are unit propagation look-ahead and extended resolution. Their implementation in RANGER produced interesting results.
We started by modifying the original RANGER’s data structures and some functions to better prepare the solver for the implementation of the new techniques. This turned RANGER into a slightly slower algorithm, mostly due to the fact that the location of all the literals need to be stored, instead of only two randomly clauses where each literal appears. In the original RANGER there was no need to know extensively where each literal is contained, but it is a pre-condition for unit propagation look-ahead and extended resolution.

Next, we implemented unit propagation look-ahead into RANGER. It was divided into two parts: a pre-processing part which performed unit propagation look-ahead with every pair of variables, and another part with only unit propagation, which ran in almost every iteration of the algorithm (it ran in 95% of its iterations). We decided to follow this direction because running unit propagation look-ahead with two pairs of variables in almost every iteration was very heavy and degraded the performance of the algorithm without adding any real usefulness: in most of the iterations no look-ahead was needed and only unit literals were found. Thus, a pre-processing phase prepares the search space of the problem instance for the algorithm by performing look-ahead, and unit propagation is then performed in almost every iteration from then on.

Unit propagation look-ahead achieved very good results in practice, beating both the original RANGER and GUNSAT in most of the instances tested (GUNSAT was only better in the test-set of unif50 problem instances). We thus recommend adding unit propagation look-ahead to future versions of RANGER, and deem it an interesting addition to future SLS algorithms that delve deeply into proving unsatisfiability.

The second technique did not work quite as well. GUNSAT utilizes extended resolution to improve its scoring system, and according to its authors it is an important part of the solver, which improves its results in practice. To implement such feature in RANGER in the same way that it is used in GUNSAT, we need to add a scoring system to RANGER. This goes against RANGER’s policies of simplicity with fast but not so intelligent moves as GUNSAT’s, which produces good results in practice. A scoring system similar to GUNSAT’s was added to RANGER (we chose a similar scoring system because it is argued by GUNSAT’s authors that extended resolution works well with that system, so there is no point in applying another one). As before, and with the purpose of not altering RANGER’s flow, extended resolution was added to the satisfiability-preserving transformations phase in the same way as unit propagation.

The scoring system added computes the scores of each pair of literals, each clause in the formula and each quadruplet (every possible combination of two variables). It then tried to improve the formula by adding a low scored clause. If the algorithm tried to improve the score of a pair of literals too many times without any success, extended resolution was applied: a new variable and three new clauses were added to the formula. This worked well in GUNSAT because the algorithm depends on the scoring system to produce good results: it is its main heuristic. In RANGER, this scoring system was only added to implement extended resolution, and thus it can be regarded as a waste of resources, not
achieving its true potential like it has been achieved in GUNSAT. RANGER’s simplicity principles prevent that.

This step of extended resolution ran only in 4.5% of the iterations, but the overhead of computing and trying to improve scores proved to be too expensive. RANGER with UPLA+ER was consistently slower than RANGER with only UPLA, and it only beat GUNSAT on easier instances.

We can conclude that extended resolution is not a viable option to be added to RANGER in future versions, or at least not in the same way it is used in GUNSAT.

Nonetheless, we recognize that the developed tool is mostly a prototype, and a re-implementation of the tool should be considered. Moreover, considering the promising results obtained, several interesting topics can be suggested for future work. Apart from implementation details, which can be decisive for reducing the expected computational effort, and therefore decisive for solving an instance in a reasonable amount of time, other aspects should be considered. Looking at extended resolution outside of GUNSAT’s use of it could prove to be interesting, as it is a simple technique that could be applied to RANGER’s own heuristic. We also expect to integrate into RANGER more simplification techniques, which are used successfully by other SLS solvers, mostly those that do not require high computations and therefore fit into RANGER’s simplicity principles.

Extended resolution could be added to RANGER in a simpler manner. It could, for example, be executed after \( x \) iterations of the algorithm where no improvement could be done. It would be interesting to test this scenario of adding a new variable and three new clauses to the formula in such a simple manner, which better reflects RANGER’s simplicity principles. The algorithm can sometimes have many consecutive non-improving steps, usually because no new clauses can be generated either by replacing or resolving clauses from the original formula into the working formula, nor by the unit propagation look-ahead and satisfiability-preserving transformations. Using extended resolution to add fresh clauses in such scenarios can prove to be a very useful technique, while keeping the methods used simple and fast (only a counter for the number of non-improving steps is used, as well as a simple method to add new clauses to the formula).

References


