Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems*

James M. Crawford and Andrew B. Baker
Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
jc@cs.uoregon.edu

Abstract
Considerable progress has been made in recent years in understanding and solving propositional satisfiability problems. Much of this work has been based on experiments on randomly generated 3SAT problems. One generally accepted shortcoming of this work is that it is not clear how the results and algorithms developed will carry over to "real" constraint-satisfaction problems. This paper reports on a series of experiments applying satisfiability algorithms to scheduling problems. We have found that scheduling problems bear fairly little resemblance to the previously studied hard randomly generated 3SAT problems. In particular, scheduling problems tend to be quite large but under-constrained. Further, forward checking (e.g., unit propagation) seems to be much more important on these problems than on hard random 3SAT problems. We have also found that the domain-specific heuristics developed to solve scheduling problems make surprisingly little difference in the time required to solve the problems. We suggest that the best algorithms for this problem class will probably be hill-climbing algorithms that incorporate some sort of forward checking.

Introduction
Many classes of problems in knowledge representation, learning, planning, and other areas of AI are known to be NP-complete. In the worst case, all known algorithms for solving such problems require run time exponential in the size of the problem. Propositional satisfiability (SAT) is, in a sense, the prototypical example of an NP-complete problem. It is simply formalized, yet has an amazingly complex structure.

Paradoxically, one perennial problem with work on SAT has been the difficulty of finding hard instances on which to test algorithms; it turns out to be surprisingly hard to collect a sufficiently large body of reasonably sized "real" problems. Randomly generated problems, on the other hand, tend to end up being quite easily solved.

One important advance in recent years has been the discovery that the difficulty of randomly generated problems depends critically on whether they are under-constrained, over-constrained, or critically-constrained (Cheeseman, Kanefsky, & Taylor 1991; Mitchell, Selman, & Levesque 1992; Crawford & Auton 1993). Consider a randomly generated constraint satisfaction problem. Intuitively, if there are very few constraints, it should be easy to find a solution (since there will generally be many solutions). Similarly, if there are very many constraints then an intelligent algorithm will generally be able to quickly close off most or all of the branches in the search tree. The hardest problems are thus those which are critically constrained: these problems have relatively few solutions, but most branches in the search tree go fairly deep before reaching a dead end.

Critically constrained randomly generated satisfiability problems provide a ready supply of hard test cases of arbitrary size. This discovery has lead to work on understanding (Crawford & Auton 1993; Williams & Hogg 1992), and solving (Selman, Levesque, & Mitchell 1992) these problems. GSAT in particular appears to be well suited to solving large randomly generated critically constrained problems. However, concerns have been raised that randomly generated problems are bad test cases because they have no structure and thus may bear little resemblance to "real" problems.

This paper reports on a series of experiments applying satisfiability algorithms to scheduling problems. In these experiments we have used Sadeh's job shop scheduling problems (both because they are readily accessible and because they have been well studied in the scheduling community). These problems do have a random component, namely the ready times and deadlines for the jobs. However, these times have been chosen according to a variety of distributions in an effort to mimic various types of scheduling problems encountered in the field.

Our main result from these experiments is that

---

*This work has been supported by the Air Force Office of Scientific Research under grant number 92-0693 and by ARPA/Rome Labs under grant numbers F30602-91-C-0036 and F30602-93-C-00031.
Scheduling

The scheduling problem is ubiquitous. One may, for example, have a set of machine tools and be told to schedule a series of jobs so as to maximize the efficiency of the use of the tools. Alternatively, one may have a collection of transport ships in various locations and be told to transport some number of divisions to a variety of locations as reliably and cheaply as possible. Or, one may have to assemble some number of surgical teams using a variety of specialists subject to a set of constraints on consecutive numbers of hours worked, availability of operating rooms, etc. In all cases, the general form of the problem is that one is given a set of tasks to achieve, and a collection of resources to use.

One important type of scheduling problems is machine shop scheduling (Xiong, Sadeh, & Sycara 1992; Smith & Cheng 1993). Sadeh has developed a test suite of machine shop scheduling problems that are intended to represent a range of the types of machine shop scheduling problems encountered in the field.

Machine shop scheduling problems are usually taken to consist of a number of operations 1, . . . , n to be scheduled subject to a collection of constraints. Each operation requires processing time p_i (given as part of the problem). A solution is a schedule giving the start time, s_i, for each operation.

The constraints are usually taken to consist of sequencing restrictions, resource capacity constraints, and ready times and deadlines (Smith & Cheng 1993). Sequencing restrictions, written i → j state that operation i must complete before j can begin. The restriction i → j is thus equivalent to s_i + p_i ≤ s_j ("the start time of i plus processing time for i is less than or equal to the start time of j"). Resource capacity constraints, written c_i,j, state that operations i and j conflict (usually because both require the same resource) and thus cannot be scheduled concurrently. c_i,j is equivalent to the disjunction (s_i + p_i ≤ s_j) ∨ (s_j + p_j ≤ s_i) ("i completes before j begins or j completes before i begins"). Ready times, represented by r_i, are the earliest time at which operation i can start. A deadline d_i is the time by which operation i must be completed. Ready times thus just state s_i ≥ r_i and deadlines that s_i + p_i ≤ d_i.

We now discuss propositional satisfiability (SAT) and then show how scheduling problems can be converted into SAT problems.
Propositional Satisfiability

The propositional satisfiability problem is the following (Garey & Johnson 1979): Given a set of clauses $C$ on a finite set $U$ of variables, find a truth assignment for $U$ that satisfies all the clauses in $C$.

Clearly one can determine whether a satisfying assignment exists by trying all possible assignments. Unfortunately, if the set $U$ is of size $n$ then there are $2^n$ such assignments. SAT algorithms thus typically either (1) walk through the space of assignments following some set of heuristics and hope to run into a solution, or (2) work with partial assignments and use some sort of forward checking to compute forced values for other variables. Algorithms in this second class generally use depth-first search to systematically search the space of assignments. In section on satisfiability algorithms below we discuss examples of both types of algorithms.

Encoding Scheduling Problems as SAT

At first glance there seems to be an "obvious" translation of scheduling problems into SAT: create variables to represent the start times of the operations (e.g., $s_i$ true means operation $s_i$ starts at time $t$), and create clauses to represent the necessary inequalities. However, the search space in SAT problems so generated turns out to be much larger than necessary.

To see this, note that the key decisions to be made in scheduling problems concern the orderings of conflicting operations. Thus, for example, if $i$ and $j$ share a resource then we have to decide whether to schedule $i$ then $j$, or $j$ then $i$. We do not necessarily have to specify the exact start times of operations $i$ and $j$, as long as we can be sure that there is some way to do so that is consistent with our ordering decisions (this observation, and the essence of the encoding we use here, is due to Smith and Cheng (1993)).

More formally, for each pair of operations $i$ and $j$, we introduce a boolean variable $pr_{i,j}$ meaning "$i$ precedes $j$", and for each operation $i$ and each time $t$ we introduce a boolean variable $sa_{i,t}$ meaning "$i$ starts at time $t$ or later", and a boolean variable $eb_{i,t}$ meaning "$i$ ends by time $t".4

Scheduling constraints are then translate by:

- $i \rightarrow j$ becomes $pr_{i,j} = true$
- $c_{i,j}$ becomes $pr_{i,j} \lor pr_{j,i}$
- $r_i$ becomes $sa_{i,r_i} = true$
- $d_i$ becomes $eb_{i,d_i} = true$

We refer to the set of constraints generated by this mapping as $C$.

It is also necessary to add a collection of "coherence conditions" on the introduced variables. In all conditions below, $i$ and $j$ are quantified over all relevant operations and $t$ is quantified over all relevant times.

1. $sa_{i,t} \rightarrow sa_{i,t-1}$ (coherence of $sa$). This ensures that if $i$ starts at or after time $t$ then it starts at or after time $t-1$.
2. $eb_{i,t} \rightarrow eb_{i,t+1}$ (coherence of $eb$). This ensures that if $i$ ends by $t$ then it ends by $t + 1$.
3. $sa_{i,t} \rightarrow \neg eb_{i,t+p_i-1}$ (job $i$ requires time $p_i$). This ensures that if $i$ starts at or after time $t$ then it cannot end before time $t + p_i$.
4. $sa_{i,t} \land pr_{i,j} \rightarrow sa_{j,t+p_j}$ (coherence of $pr_{i,j}$). This ensures that if $i$ starts at or after $t$ and $j$ follows $i$ then $j$ cannot start until $i$ is finished.

We refer to the set of coherence conditions as $S$. Any mapping of the variables $pr_{i,j}$ to $\{T, F\}$ that extends to a model of $C \land S$ is then a template describing a set of solutions to the scheduling problem.

There are several advantages to this translation from scheduling problems to SAT:

1. $A$ set of legal values for the $pr$ variables corresponds to a collection of feasible schedules. If there are additional optimization conditions (e.g., robustness), one can then use these to select a particular schedule.
2. The constraints in $S$ apply to all scheduling problems and thus need be computed and stored only once (they might therefore be compiled into a procedure rather than being stored explicitly). Further, $S$ is symmetric under any permutation of the operations. In order to check for symmetries among operations it is thus only necessary to consider $C$.
3. Without loss of generality, one can omit the variable $pr_{i,j}$ (and all clauses containing it) if there are no sequencing restrictions or resource capacity constraints for $i$ and $j$. This means that the size of the SAT problem is of order $nd + c$ (where $n$ is the number of operations, $d$ is the number of distinct time points, and $c$ is the number of constraints in the scheduling problem). If $S$ is represented as a compiled procedure then the number of clauses in the SAT problem is further reduced to order $c$ (plus the size of the compiled procedure).

Three Satisfiability Algorithms

Tableau

TABLEAU is a Davis-Putnam algorithm that does a depth-first search of possible assignments using unit propagation for forward checking. This basic algorithm dates back to the work of Davis, Logemann, and Loveland (Davis, Logeman, & Loveland 1962). TABLEAU adds efficient data-structures for fast unit propagation and a series of heuristics for selecting branch variables...
(these are discussed in Crawford & Auton 1993). Most of these heuristics do not seem particularly helpful for scheduling problems (see discussion of experimental results below).

Unit propagation consists of the repeated application of the inference rule:

\[ z \lor y_1 \lor \cdots \lor y_n \]

(similarly for \( \neg z \)). Unit propagation is a special case of resolution (the singleton \( z \) is resolved against the \( \neg z \) in the clause). It is a particularly useful case, however, since it can always be performed to completion in time linear in the size of the theory, and since, in practice, it greatly reduces the number of nodes in the search tree (by propagating variable values through the theory).

The basic depth-first search algorithm is then the following:

\[
\text{tableau}(\text{theory}) \equiv \\
\begin{aligned}
&\text{unit-propagate}(\text{theory}); \\
&\text{if contradiction discovered return(false);} \\
&\text{else if all variables are valued} \\
&\quad \text{return(current assignment);} \\
&\text{else} \\
&\quad x = \text{some unvalued variable; } \\
&\quad \text{return(tableau(\text{theory AND } x) OR} \\
&\quad \text{tableau(\text{theory AND NOT } x));}
\end{aligned}
\]

Gsat

Gsat is the most successful hill-climbing search algorithm for SAT to date. A complete assignment of variables to values is always kept. Variables are "flipped" (their value is changed) so as to increase the number of satisfied clauses (if possible). In our experiments we found the use of the "walk" strategy (Selman & Kautz 1993) to be critical. The basic WSAT ("walk sat") algorithm is the following:

\[
\text{WSAT}(\text{theory}) \\
\begin{aligned}
&\text{for } i := 1 \text{ to MAX-TRIES } \{ \\
&\quad A := \text{a randomly generated truth assignment; } \\
&\quad \text{for } j := 1 \text{ to MAX-FLIPS } \{ \\
&\quad \quad \text{if } A \text{ is a solution return it;} \\
&\quad \quad \text{else } \\
&\quad \quad \quad C := \text{randomly chosen unsatisfied clause;} \\
&\quad \quad \quad \text{With probability } P, \\
&\quad \quad \quad \quad \text{Flip a random variable in } C; \\
&\quad \quad \quad \text{Otherwise (that is, with probability } 1-P \text{) } \\
&\quad \quad \quad \quad \text{Flip a variable in } C \text{ resulting in the greatest decrease in the number of unsat clauses;} \\
&\quad \quad \}\text{; } \\
&\quad \}\text{; } \\
&\text{return failure}
\end{aligned}
\]

Obviously this approach will only work on problems with a large number of models. One of the surprises in our work on scheduling problems has been that this algorithm outperforms both tableau and gsat.

Experimental Results

Our experiments were designed to assess the performance of each of these three algorithms on scheduling problems. We used the sixty scheduling problems produced by Sadeh (Sadeh 1992). Each of these problems consists of fifty operations to be scheduled subject to sequencing restrictions and resource capacity constraints. The operations are grouped into ten jobs of five operations each. Operations within each job must be performed in order. Further, each job requires one of five resources and each resource can be used by at most one job at a time.

Ready times and deadlines were generated randomly using several distributions. The distributions were defined by two parameters: (1) degree of constraint: (w) wide, (n) narrow, and (t) tight, and (2) number of bottlenecks: none, one, or two. These two parameters yield the six classes shown in Figure 1. Sadeh produced ten sample problems from each class.

The results for isamp and gsat are shown in Figure 1. The gsat results are for our version of the wsat ("walk sat") variant of gsat (Selman & Kautz 1993). These results are an average over ten runs on each problem. In each run gsat was given ten tries of four million flips each which corresponds to about
**TABLEAU:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Success Rate</th>
<th>Branches</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/l</td>
<td>90</td>
<td>3947.1</td>
<td>255.4</td>
</tr>
<tr>
<td>w/2</td>
<td>100</td>
<td>221.2</td>
<td>104.8</td>
</tr>
<tr>
<td>n/1</td>
<td>70</td>
<td>1719.7</td>
<td>79.2</td>
</tr>
<tr>
<td>n/2</td>
<td>100</td>
<td>93.8</td>
<td>90.6</td>
</tr>
<tr>
<td>t/1</td>
<td>80</td>
<td>1160.4</td>
<td>66.5</td>
</tr>
<tr>
<td>t/2</td>
<td>100</td>
<td>119.4</td>
<td>81.7</td>
</tr>
</tbody>
</table>

**GSAT:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Success Rate</th>
<th>Flips (millions)</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/l</td>
<td>99</td>
<td>0.7</td>
<td>500</td>
</tr>
<tr>
<td>w/2</td>
<td>99</td>
<td>7.0</td>
<td>599</td>
</tr>
<tr>
<td>n/1</td>
<td>100</td>
<td>3.2</td>
<td>258</td>
</tr>
<tr>
<td>n/2</td>
<td>100</td>
<td>3.7</td>
<td>312</td>
</tr>
<tr>
<td>t/1</td>
<td>96</td>
<td>3.9</td>
<td>274</td>
</tr>
<tr>
<td>t/2</td>
<td>88</td>
<td>5.0</td>
<td>354</td>
</tr>
</tbody>
</table>

**ISAMP:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Success Rate</th>
<th>Tries</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/l</td>
<td>100</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>w/2</td>
<td>100</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>n/1</td>
<td>100</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>n/2</td>
<td>100</td>
<td>48</td>
<td>21</td>
</tr>
<tr>
<td>t/1</td>
<td>100</td>
<td>52</td>
<td>19</td>
</tr>
<tr>
<td>t/2</td>
<td>100</td>
<td>252</td>
<td>68</td>
</tr>
</tbody>
</table>

Figure 1: Experimental results for scheduling problems.

Forty-five minutes of computation time. The mean flip and time data is for the successful cases only (these counts would be higher if we "punished" GSAT for the cases on which it ran out of time). For ISAMP the results are an average over 100 runs. ISAMP runs were given twenty-thousand tries. TABLEAU was run with only the non-horn heuristic ("branch first on variables appearing in non-horn clauses"). This has the effect of forcing TABLEAU to branch on the $p_r$ variables. Since TABLEAU currently has no random component, the results are for one run on each problem. TABLEAU was interrupted after forty-five minutes. The averages are over the successful cases only. All algorithms are implemented in C, and all experiments were run on a SPARC 10/51.

In order to test the hypothesis that the lack of unit propagation was hurting GSAT, we performed a linear time simplification on the propositionally encoded scheduling problems and ran the experiments again. The simplification consisted of running unit propagation to completion on the initial theory (the encoding is such that the initial theories contain a number of unit clauses). We then deleted any clauses that were subsumed by a unit literal (e.g., if the theory contained $x$ and $x \lor y \lor z$ then we deleted the clause). Finally we "compacted" the encoding — our encoding uses integers to represent variables so this step ensured that if the largest variable in the theory is $n$ then every integer less than $n$ is also used for some variable in the theory. The data for the simplified theories is shown in Figure 2. As expected, the GSAT run times are much lower. The ISAMP run times are also lower. We believe that this is primarily due to the fact that the simplified theory is smaller (and thus faster to read in from disk). We did not rerun TABLEAU since the simplification should have minimal effect on its run time. In this experiment GSAT and ISAMP were each run ten times on each problem.

<table>
<thead>
<tr>
<th>Class</th>
<th>Success Rate</th>
<th>Flips (millions)</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/l</td>
<td>100</td>
<td>0.39</td>
<td>27</td>
</tr>
<tr>
<td>w/2</td>
<td>100</td>
<td>0.29</td>
<td>23</td>
</tr>
<tr>
<td>n/1</td>
<td>100</td>
<td>0.31</td>
<td>23</td>
</tr>
<tr>
<td>n/2</td>
<td>100</td>
<td>0.55</td>
<td>43</td>
</tr>
<tr>
<td>t/1</td>
<td>100</td>
<td>1.1</td>
<td>77</td>
</tr>
<tr>
<td>t/2</td>
<td>97</td>
<td>2.9</td>
<td>211</td>
</tr>
</tbody>
</table>

Figure 2: Experimental results for scheduling problems after simplification.

**Discussion**

The success of ISAMP indicates that, given the right encoding, Sadeh's scheduling problems are not that difficult. The encoding we use is domain specific only in that it is implicitly based on the observation that the key choices to be made are the relative orders of the conflicting operations. This is much less domain-specific than the slack-based heuristics used by Smith and Cheng (Smith & Cheng 1993). Of course Smith and Cheng do solve these problems almost two orders of magnitude faster than ISAMP. However, most of this difference is probably due to our use of propositional logic as a representation language (just to read in a propositional version of these theories takes about thirty times as long as Smith and Cheng take to solve them). The obvious experiment of implementing ISAMP using a more natural representation language (e.g., a constraint-satisfaction language with integral-valued variables) is underway.

A great deal of recent work has been done on analysis and solution methods for randomly generated satisfiability problems. An important open question in this body of work has been its relevance to "real" problems. Our work attempts to begin providing an answer to
this question by studying the performance of a variety of satisfiability algorithms on propositional encodings of scheduling problems.

We have found that scheduling problems generate propositional theories that are much larger and much less constrained than the randomly generated theories generally studied. Further, the variables in scheduling problems can be partitioned into control variables that define a solution (e.g., the pr variables in scheduling problems) and dependent variables whose values are derived from the control variables (e.g., all the rest of the variables in the scheduling problems).

The fact that TABLEAU generally performs poorly on these problems while ISAMP performs well indicates that there are a large number of solutions but that these solutions are not uniformly distributed throughout the search space. Rather there seem to be large "deserts" containing no solutions. TABLEAU sometimes wanders into one of these deserts by making an unlucky choice at some early branch in the tree. It then has no way to recover. We refer to this as the early mistake problem. Since ISAMP restarts on every contradiction, it is sensitive only to the number of solutions, not their uniformity.

The existence of a large number of dependent variables appears to be hobbling GSAT relative to ISAMP. One can see this by the following analysis. Assume that we can divide the variables in a problem into c control variables and d dependent variables, such that a polynomial time procedure will always determine whether an assignment to the control variables extends to a model. This will be the case, for example, if we choose the control variables to be variables appearing in non-horn clauses (e.g., the pr variables in scheduling problems) and choose unit propagation as the polynomial time procedure. If there are any constraints on the dependent variables the density of solutions in "control variable space" will then be higher than density of solutions in the original search space (to see this note that the density could be equal only if any satisfying assignment to the control variables extended to $2^d$ models). GSAT clearly searches in the full space (since it uses no forward checking). We hypothesize that ISAMP effectively searches in a smaller space (even though it makes no explicit distinction between control and dependent variables) because of its use of unit propagation. TABLEAU also used unit propagation but it appears to fail because of the early mistake problem discussed above. Work on GSAT suggests that hill-climbing is a useful technique for satisfiability problems and in general one would expect hill-climbing to be superior to the random probing of ISAMP. This clearly suggests that the next step in this line of work is to develop algorithms that hill-climb in control space.

The existence of a large number of dependent variables appears to be hobbling GSAT relative to ISAMP. One can see this by the following analysis. Assume that we can divide the variables in a problem into c control variables and d dependent variables, such that a polynomial time procedure will always determine whether an assignment to the control variables extends to a model. This will be the case, for example, if we choose the control variables to be variables appearing in non-horn clauses (e.g., the pr variables in scheduling problems) and choose unit propagation as the polynomial time procedure. If there are any constraints on the dependent variables the density of solutions in "control variable space" will then be higher than density of solutions in the original search space (to see this note that the density could be equal only if any satisfying assignment to the control variables extended to $2^d$ models). GSAT clearly searches in the full space (since it uses no forward checking). We hypothesize that ISAMP effectively searches in a smaller space (even though it makes no explicit distinction between control and dependent variables) because of its use of unit propagation. TABLEAU also used unit propagation but it appears to fail because of the early mistake problem discussed above. Work on GSAT suggests that hill-climbing is a useful technique for satisfiability problems and in general one would expect hill-climbing to be superior to the random probing of ISAMP. This clearly suggests that the next step in this line of work is to develop algorithms that hill-climb in control space.

Acknowledgements

We would like to thank the members of CIRL, particularly Matt Ginsberg, for useful discussions of this material.

References


