Backbones and Backdoors in Satisfiability

Philip Kilby
ANU
Canberra, Australia
Philip.Kilby@anu.edu.au

John Slaney
NICTA* and ANU
Canberra, Australia
John.Slaney@anu.edu.au

Sylvie Thiébaux
NICTA* and ANU
Canberra, Australia
Sylvie.Thiebaux@anu.edu.au

Toby Walsh
NICTA* and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

We study the backbone and the backdoors of propositional satisfiability problems. We make a number of theoretical, algorithmic and experimental contributions. From a theoretical perspective, we prove that backbones are hard even to approximate. From an algorithmic perspective, we present a number of different procedures for computing backdoors. From an empirical perspective, we study the correlation between being in the backbone and in a backdoor. Experiments show that there tends to be very little overlap between backbones and backdoors. We also study problem hardness for the Davis Putnam procedure. Problem hardness appears to be correlated with the size of strong backdoors, and weakly correlated with the size of the backbone, but does not appear to be correlated to the size of weak backdoors nor their number. Finally, to isolate the effect of backdoors, we look at problems with no backbone.

Introduction

Many problems in AI like constraint solving, planning and learning are intractable in general. Propositional satisfiability (or SAT) is typical of such problems. It is a problem of considerable practical and theoretical importance. SAT was the first problem shown to be NP-complete (Cook 1971). It therefore lies at the heart of the theory of computational complexity. In addition, many real world problems like planning have been encoded into SAT. Highly optimized SAT solvers are then used to find solutions.

Due in part to its simplicity, SAT has become a problem class in which to study search and the causes of intractability. A number of fundamental notions have been identified to explain why search problems are hard. Two such notions are the backbone of a search problem and a backdoor into a search problem (see next section for their formal definitions). The first identifies those decisions which are fixed in all solutions (and so need to be made correctly), whilst the second identifies those decisions which result in a polynomial subproblem. Both these notions have been connected to problem hardness (Monasson *et al.* 1998; Williams, Gomes, & Selman 2003).

In this paper, we look at the connections between backbones, backdoors, and problem hardness. We are interested as to whether some commonality between backbones and backdoors explains their connection to problem hardness. Are backbone literals more or less likely to also likely to form backdoors? From a practical point of view, we are also interested in approximations of the backbone set.

Backbones and backdoors

The **backbone** of a satisfiable SAT problem is the set of literals which are true in every satisfying truth assignment. There are a number of different definitions for the backbone of an unsatisfiable formula (e.g. the set of literals fixed in every truth assignment maximizing the number of satisfied clauses (Monasson *et al.* 1998)). We will avoid this complication by focusing on satisfiable only formulae. Backbone size has been associated with problem hardness (Parkes 1997; Monasson *et al.* 1998; Achlioptas *et al.* 2000). If a SAT problem has a large backbone, there are many opportunities to assign variable incorrectly. Such problems tend to be hard therefore for systematic methods like Davis-Putnam. A large backbone also means that solutions are clustered. Such problems therefore can be hard to solve with local search methods like WalkSAT.

A backdoor into a SAT problem is a (hopefully small) set of variables which provide a "short cut" into solving the problem (Williams, Gomes, & Selman 2003). A **weak backdoor** of a satisfiable SAT problem is the set of literals which give a simplified formula which is satisfiable and can be solved in polynomial time. A **strong backdoor** of a satisfiable or unsatisfiable SAT problem is the set of variables which, however they are assigned, give a simplified formula which can be solved in polynomial time.

The definition of backdoors is inherently algorithm dependent - a backdoor set for one algorithm is not necessarily a backdoor for another. We use as the definition of nacdoor branch-free search in *satz* version 2.15 (Li 1999). That is, once the backdoor literals have been assigned, the problem can be solved essentially through unit propagation.

We will typically consider backdoors that are *minimal*; that is, no strict subset of the backdoor is itself a backdoor. Note also that the concept of weak and strong backdoor is

*National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council

parameterized by a polynomial class of subformulae. This class may be defined syntactically (e.g. Horn formulae) or algorithmically (e.g. those formulae polynomially decided by a Davis Putnam procedure). Empirical studies have shown that many structured SAT problems have small backdoors, whilst random 3-SAT problems do not. Gomes et al. argue this may explain why we can typically solve random 3-SAT problems with a only few hundred variables but can often solve structured problems with thousands of variables (Williams, Gomes, & Selman 2003). Small backdoors also help to explain the heavy-tailed behaviour of backtracking search algorithms (Williams, Gomes, & Selman 2003).

Computational complexity

Computing the backbone or a backdoor of a SAT problem is intractable in general. To be more precise, finding the backbone of a SAT problem is both NP-hard and NP-easy. It is NP-hard as we can determine the satisfiability of a formula with a polynomial number of calls to a procedure to find the backbone (as argued in the proof of Theorem 1). It is NP-easy as deciding if a literal is in the backbone can be solved with a single call to a SAT decision procedure. Garey and Johnson suggest that problems which are both NP-hard and NP-easy might be called NP-equivalent (Garey & Johnson 1979). Although this class contains problems which do not belong to NP, the class has the property of NP-complete decision problems that: unless P=NP, no problem in the class can be solved in polynomial time, and if P=NP then all problems in the class can be solved in polynomial time. In other words, the problem of deciding the SAT backbone is polynomial if and only if P=NP.

Finding the backdoor into a SAT problem is also intractable in general (assuming P $\neq$ NP). However, if we can bound the size of the backdoor, it can be tractable in certain cases. In particular, finding a strong backdoor set (into either Horn or 2-cnf polynomial subformulae) is tractable if the size of the backdoor is bounded, but finding a weak backdoor set is not (Nishimura, Ragde, & Szeider 2004).

Approximation

We now show that even approximating the backbone is intractable in general. Suppose we have a sound approximation procedure that returns some subset of the backbone. That is, any literal returned by the procedure is guaranteed to be in the backbone, but it may not return all of the backbone.

Theorem 1 *If P $\neq$ NP then no sound approximation procedure can return a fixed fraction α or greater of the SAT backbone in polynomial time.*

Proof: Suppose there was such a polynomial time approximation procedure. Since the approximation procedure returns a fixed fraction of the backbone (rounded up) it must return at least one backbone literal if the backbone is non-empty. We set this literal to true and simplify the formula. If the backbone is empty, we set an arbitrary literal to true and simplify. We then call the approximation procedure and repeat. This procedure will find a satisfying assignment if one exists in polynomial time, contradicting the assumption that P $\neq$ NP. $\Box$

The same argument shows that, if P $\neq$ NP then no sound approximation procedure can exist that is guaranteed to return at least one backbone literal when the backbone is non-empty in polynomial time.

Suppose instead that we have an unsound approximation procedure. That is, literals returned by the procedure are not guaranteed to be in the backbone. If we do not limit the number of literals incorrectly assigned to the backbone, then there exists a polynomial time approximation that meets any approximation ratio. For example, consider the procedure that returns all literals. We therefore consider unsound approximation procedures which limit the number of literals falsely assigned to the backbone. An approximation procedure is a "majority-approximation" iff, when the backbone is non-empty, the ratio of the number of literals falsely assigned to the backbone compared to the number returned is strictly less than 1/2. If the backbone is empty, any number of literals can be falsely returned as being in the backbone.

Theorem 2 *If P $\neq$ NP then no majority-approximation procedure can be guaranteed to return a fixed fraction α or greater of the literals in the backbone in polynomial time.*

Proof: Suppose there was such a polynomial time approximation procedure. We show how such a procedure can be used to decide the satisfiability of a set of clauses Σ in polynomial time, contradicting the assumption that P $\neq$ NP. Let k be $\lceil \frac{1}{\alpha} \rceil$. We construct k copies of Σ augmenting the clauses as follows. In the ith copy, we add the disjunct x_i to each clause, where x_i is a new variable not in Σ. We denote these modified clauses by Σ_i. Even if Σ is unsatisfiable, we can always satisfy Σ_i by setting x_i to true. We now consider the satisfiable set of clauses: $\{z\} \cup (\bigcup_{1 \leq i \leq k} \Sigma_i)$ where z is again a new variable. Note that, as α is fixed, this set of clauses is polynomial in the size of Σ. If Σ is satisfiable, then z is the unique backbone literal. If Σ is unsatisfiable, then the backbone is $\{x_1, \dots, x_k, z\}$. We now use our approximation procedure to compute the backbone of the constructed formula. If the formula Σ is satisfiable then the majority approximation procedure must return just z. If the formula Σ is unsatisfiable then the majority approximation procedure must return at least one literal, x_j from the backbone. Hence, the backbone literals returned can be used to decide the satisfiability of Σ. $\Box$

Note that in the proof, we just computed the backbone of satisfiable formulae. Hence, the same result holds however we define the backbone of unsatisfiable formulae.

Algorithms

A series of algorithms was developed to conduct empirical tests on backdoors and backbones. Algorithms and tests are based on a modified version of *satz* version 2.15. These modifications were required to ensure backdoors were robust to renaming of variables. That is, if the same problem is presented with variables in a different order, the backdoors discovered remains the same.

Algorithm MINWEAKBACKDOOR

This is a simple routine used by all the other algorithms to reduce an initial weak backdoor into a minimal weak backdoor. It maintains a set of variables (W) which must form part of a minimal weak backdoor. It selects literals from the initial set I and tests them for inclusion in W. The algorithm also returns a model consistent with the backdoor set returned. The backdoor set is minimal in that no proper subset is also a weak backdoor. However, the algorithm is not guaranteed to return the backdoor set of minimal cardinality from a given input. A sequential version of this algorithm treats I as a list rather than a set, and chooses the literal l in step 3 sequentially.

Algorithm 1 MinWeakBackdoor (F, I)

Input: Formula F, Initial weak backdoor set I - i.e. running *satz* on $F \cup I$ requires no branching.

Output: A set of literals W forming a minimal backdoor, and a model M consistent with the backdoor

1. $W \leftarrow \emptyset$; $M \leftarrow \emptyset$
2. **while** $I \neq \emptyset$
3.   Choose literal $l \in I$ randomly
4. $I \leftarrow I \setminus \{l\}$
5.   Run *satz* on $F \cup W \cup I$
6.   **if** *satz* requires branching,
7.   **then** $W \leftarrow W \cup \{l\}$; $M \leftarrow$ *satz* solution
8. **endwhile**
9. **return** W, M

Algorithm SATZWEAK

This algorithm creates a weak backdoor set using branching variables selected by *satz*. It then reduces the set to a minimal backdoor using MINWEAKBACKDOOR. It returns a minimal weak backdoor, and a consistent model.

Algorithm 2 SatzWeak (F)

Input: Formula F

Output: A minimal weak backdoor W and a consistent model M

1. Solve F using *satz* , saving branching literals in B
2. $W, M \leftarrow$ MINWEAKBACKDOOR (F, B)
3. **return** W, M

Algorithm SATZLS1

We are interested in weak backdoors of minimum cardinality. SATZWEAK does not always find the smallest backdoor, so we use local search to explore "neighbouring" backdoors.

The algorithm maintains an incumbent backdoor W. It adds literals chosen randomly to that set, and then reduces it to a minimal backdoor again. The algorithm periodically restarts with the smallest backdoor found so far. Using the sequential version of MINWEAKBACKDOOR, the literals in the current incumbent are tested first for exclusion from the set. This helps to drive the algorithm to discover new weak backdoors. The algorithm returns $\mathcal{S}$ – all distinct minimal weak backdoors discovered.

Three constants are used in the algorithm: Iteration-limit (the number of iterations per restart); Restart-limit (the number of restarts); and Card-mult (the multiplier for the number of literals added to the incumbent backdoor). For the runs reported here the following values were chosen after some initial experimentation: Iteration-limit is $\sqrt{n} * 3$, Restart-limit is 2. and Card-mult is 2.

Algorithm 3 SatzLS1 (F, W, M)

Input: Formula F, Initial backdoor W, Model M

Output: A set of minimal weak backdoors $\mathcal{S}$

1. $\mathcal{S} \leftarrow \emptyset$; $B \leftarrow W$
   \# W is current backdoor,
   \# B is an example of smallest backdoor seen.
2. Restart-count $\leftarrow 0$
3. **while** Restart-count $<$ Restart-limit
4.   Restart-count $\leftarrow$ Restart-count + 1
5. $W \leftarrow B$
6.   Iteration-count $\leftarrow 0$;
7.   **while** Iteration-count $<$ Iteration-limit
8.   Iteration-count $\leftarrow$ Iteration-count + 1
9. $Z \leftarrow \mid W \mid \times$ Card-mult literals chosen randomly from $M \setminus W$
10. $W \leftarrow$ MINWEAKBACKDOOR $(F, W \cup Z)$ (sequential version)
11. $\mathcal{S} \leftarrow \mathcal{S} \cup W$
12.   **if** $\mid W \mid < \mid B \mid$
13.   **then** $B \leftarrow W$; Restart-count $\leftarrow 0$
14.   **endwhile**
15. **endwhile**
16. **return** $\mathcal{S}$

When a new, smaller backdoor is found at line 12 the restart counter is reset to give the procedure an opportunity to find more examples of backdoors of this size. The procedure is called with W and M returned by SATZWEAK(F).

Algorithm SATZLS2

SATZLS1 tends to generate backdoor sets from a single model. As there are often many models for a formula, the procedure needs to be forced to explore new models. SATZLS2 was developed to accomplish this.

SATZLS2 uses repeated calls of SATZLS1. The neighbourhood of the backdoor created from the *satz* branching variables is explored first. The algorithm then explores backdoors generated using a number of randomly chosen models. An advantage of the algorithm is that initial weak backdoors for second and subsequent models are created in a way that does not rely on the *satz* algorithm. This helps to ensure the backdoors are not biased by the *satz* branching rules.

The initial backdoor for a model is created by simply adding literals chosen at random from the model until the set forms a weak backdoor. It is reduced using MINWEAKBACKDOOR.

In tests reported here, the list of models L is 9 models chosen randomly from all possible models of F, giving a total of 10 models tested.

Algorithm 4 SatzLS2(F, L)

Input: Formula F, and a list of models L.

Output: A set of minimal weak backdoors $\mathcal{S}$

```
S ← SatzLS1 (F, SatzWeak(F)).
while L ≠ ∅
   Choose M from L
   L ← L \ M
   W ← ∅
   do
      Choose l randomly from M \ W
      W ← W ∪ {l}
      Solve F ∪ W using satz
   until no branching required
   W ← MinWeakBackdoor (F, W)
   S ← S ∪ SatzLS1 (F, W, M)
endwhile
return S
```

Algorithm StrongBackdoor

A simple algorithm to calculate strong backdoors was written which simply tests every combination of literals up to a fixed cardinality. The algorithm has the advantage that for small problems, every weak and strong backdoor up to the given size is generated. However, this procedure can only be used on small problems. The search for a strong backdoor in a satisfiable formula can be sped up considerably using frequency information gathered *a priori* during local search. A score representing the number of times the variable appears in small weak backdoors is calculated in the following way. First, run SatzLS2 or similar procedure to generate a set $\mathcal{S}$ of weak backdoors. Examine each backdoor in $\mathcal{S}$. For a backdoor of size w, and for each variable v corresponding to a literal in the backdoor, add $1/w$ to the score for v. During the search for a strong backdoor, the variables are examined at line 2 above in order of decreasing score.

Algorithm 5 StrongBackdoor (F, Max-card)

Input: Formula F, the maximum cardinality Max-card.

Output: A set of strong backdoors $\mathcal{S}$, and a set of weak backdoors $\mathcal{W}$.

```
1. S ← W ← ∅
2. for each subset X of the problem variables of size
      up to Max-card
3.    for each distinct set of literals L corresponding
         to the variables in X
4.       Run satz on F ∪ L
5.       if branching is not required,
            and the formula is satisfiable
6.          then W ← W ∪ L
7.    if no literal set required branching
8.       then S ← S ∪ X
9. return S, W
```

We tested the ordering algorithm on problems with 20 variables, using as $\mathcal{S}$ the list of all weak backdoors up to size 3. Compared to the default ordering $(1, 2, 3, \ldots)$, the improved order found a strong backdoor in 40% fewer iterations per problem on average. In problems of size 50 we used the list of weak backdoors found by SatzLS2. The new ordering required 25% fewer iterations per problem.

Empirical comparisons

To study the connection between backbones, backdoors and problem hardness, we ran a number of experiments. For strong backdoors, we were computationally limited to problems with up to 50 variables. For weak backdoors, we were able to study larger problems with up to 225 variables, both random and more structured. The problem sets used are listed in Table 1. All problems in all sets are satisfiable, except uuf50. Correlation between statistics is measured using Pearson's r-value, and the corresponding coefficient of determination (c.o.d.).

Abbrev	Description	n	m	Number of Inst.
RTI	Random 3-sat	100	429	500
uf20	Random 3-sat	20	91	1000
uf50	Random 3-sat	50	218	1000
uuf50	Random 3-sat (unsatisfiable)	50	218	1000
uf100	Random 3-sat	100	430	1000
uf125	Random 3-sat	125	538	100
uf150	Random 3-sat	150	645	100
uf175	Random 3-sat	175	753	100
uf200	Random 3-sat	200	860	100
uf225	Random 3-sat	225	960	100
flat30	SAT-encoded graph colouring	90	300	100
flat50	SAT-encoded graph colouring	150	545	100

Table 1: Problem Sets

Strong and weak backdoors

Of the 1000 random 3-Sat problems with 20 variables from SATLIB, 275 can be solved by *satz* in pre-processing, and hence can be said to have a zero length strong and weak backdoors. We ran the systematic procedure StrongBackdoor that looks for backdoors of size up to 3. All problems have a weak backdoor of length 3 or less. Of the 725 non-trivial problems, 214 have a strong backdoor of length greater than 3.

Each formula F can have a large number of backdoors, both strong and weak, of various sizes. The cardinality off the smallest strong and weak backdoor sets are of particular interest. Let the cardinality of the smallest weak (resp. strong) backdoor for a formula F be w_F (resp. s_F). Strong backdoors in the test set were only slightly larger than weak ones. Over the 1000 instances, w_F averaged 0.76 literals, whilst s_F averaged 1.1 variables. The set of literals that are members of smallest weak backdoors are also of interest. In the following, we use W_F^* to denote the set of literals that are members of at least one weak backdoor of size w_F. Similarly S_F^* is the set of variables that appear in at least one strong backdoor of size s_F.

Backbone *versus* backdoor

It is not hard to show that there is no logical connection between backbones and backdoors. There are problem classes which are NP-complete in which the backbone and backdoor variables are disjoint. We can, however, see if there is a statistical connection. Are backbones likely to be backdoors, and vice versa? The answer appears to be, not very likely. In the 1000 tests on problems of size 20, *no* literals in W_F^* were also backbone literals. That is, no backbone literal was also in a smallest weak backdoor. Results are presented in Table 2.

Weak backdoors

Problem set	Mean backbone size	Mean w_F	Mean $\mid W_F^* \mid$	Mean Overlap
uf20[1]	13.7	0.76	4.4	0
uf50[1]	30.8	1.6	18.8	6.5
uf100[2]	53.6	4.6	10.2	2.7
RTI[2]	53.8	4.5	9.8	2.5

Strong backdoors

Problem set	Mean backbone size	Mean s_F	Mean $\mid S_F^* \mid$	Mean Overlap
uf20[1,3]	15.5	1.1	3.1	0
uf50[1,3]	43.4	2.0	16.4	13.0

[1] These entries are based on systematic search
[2] These entries are based on local search
[3] Only problems where a strong backdoor of size < 4 was found. (Hence mean backbone size differs from weak backdoor table)

Table 2: Overlap between backbones and backdoors

In larger problems, the smallest weak backdoor and the backbone do overlap, but not to a great extent. Strong backdoors cannot be compared directly to backbones, as strong backdoors are expressed in terms of variables, while backbones are sets of literals. However, we can say a backbone literal of F is in the overlap set if its corresponding variable is in S_F^*. Even with this fairly loose definition, there was no overlap between backbone and smallest strong backdoors in the size 20 problems. Again, as problem size grew, more backbone variables appear in the strong backdoor.

Problem hardness

Problem hardness is taken to be the log of the number of search nodes required by *satz*. We present the most interesting correlations with problem hardness in Table 3, and discuss them in the text following.

The strongest correlation with problem hardness we found was the size of the smallest strong backdoors. The size of S_F^* correlated with problem hardness for the problems with strong backdoors of size < 4. Guessing that the remaining problems had a strong backdoor length of 4 increased the r-value to 0.78. The corresponding c.o.d. tells us that about 60% of the variation in problem hardness is accounted for by

Problem set	Statistic	r-value	c.o.d.
uf20	s_F	0.71	0.50
uf20	s_F + guess	0.78	0.61
uf20	w_F	0.58	0.00
uf20	Backbone size	-0.88	0.78
uuf50	s_F	0.74	0.54
uf50	s_F	0.37	0.14
uf50	s_F + guess	0.46	0.21
uf50	w_F	0.26	0.07
uf50	Backbone size	-0.46	0.21
RTI	w_F	0.03	0.00
uf100	w_F	-0.01	0.00
uf125	w_F	-0.03	0.00
uf150	w_F	0.07	0.01
uf175	w_F	0.01	0.00
uf200	w_F	-0.13	0.02
uf225	w_F	-0.05	0.00
RTI	Backbone size	0.16	0.03
uf100	Backbone size	0.25	0.06
uf125	Backbone size	0.32	0.10
uf150	Backbone size	0.28	0.08
uf175	Backbone size	0.25	0.06
uf200	Backbone size	0.34	0.12
uf225	Backbone size	0.48	0.23
flat30	Num weak backdoors overall	-0.31	0.09
flat50	Num weak backdoors overall	-0.37	0.14
flat30	Num models	0.42	0.18
flat50	Num models	0.54	0.

Table 3: Correlations with problem hardness

variation in strong backdoor length. This strong correlation was also present in the unsatisfiable, size 50 problems in uuf50 (all of which had a strong backdoor of size < 4).

The correlation with strong backdoor size was not as strong with satisfiable problems of size 50. Only 395 of these had a strong backdoor of size < 4. The correlation with problem hardness was not as marked as for the previous problem sets. We have been able to examine only a few larger problems to see if the effect is evident. The pigeon-hole problems from SATLIB are very hard for *satz*. The number of search nodes required is typically one or two orders of magnitude larger than similar-sized random problems. For example the 6-pigeon-hole problem has 42 variables and requires 14,604 search nodes, and has a strong backdoor of size at least 5. The average number of search nodes for random 50-node problems is 380, and the average strong backdoor size for non-trivial problems is 2. The hard problem therefore has a comparatively large strong backdoor. Unfortunately, computation cost prohibited us from examining any other larger problems.

For the smallest problems, the size of the weak backdoors is weakly correlated with problem hardness. However, the effect reduces for the larger problems sizes, so that the statistic is uncorrelated for problems of size 100 and above. Surprisingly, problem hardness is *negatively* correlated with backbone size in the smallest problems. The larger problems exhibit the positive correlation observed elsewhere (Slaney & Walsh 2001).

Number of backdoors

The number of distinct smallest strong backdoors was not a significant predictor of hardness. Basically all the predictive power was in the 0/1 test "How many 0-length strong backdoors are there?", which is equivalent to "Is this problem trivial?". Adding length 1, 2, or 3 strong backdoors did not increase the correlation. The number of distinct smallest weak backdoors, and the number of distinct weak backdoors seen overall, were also not well correlated with problem hardness. This is surprising, as the existence of a small backdoor, or even many small backdoors, would seem to suggest the problem may not be hard.

Backbone free problems

What if we eliminate the influence of the backbone by ensuring problems do not have any backbone? The usual SAT encoding of a graph coloring problem lacks any backbone since we can permute any coloring of the graph. Such encodings showed interesting results. There was a slight, negative correlation between problem hardness and the number of weak backdoors (i.e., weak backdoors of size 1, 2 and 3 combined). Approximately 10% of the variation in hardness is explained by the number of weak backdoors overall. The negative correlation is as expected, but the effect is not very strong. Surprisingly, there was a positive correlation between problem hardness and the total number of models. As the number of models increases, one would expect the problem to become easier. For random problems, the correlation was negative in all problem groups, but very weak.

Related Work

Beacham has considered the complexity of computing the backbone for a range of decision problems like the satisfiability and Hamiltonian path problem (Beacham 2000). He considers a slightly modified definition of backbone: the set of decisions whose negation give an unsatisfiable subproblem. This definition is equivalent to the usual one for satisfiable problems but gives every decision for unsatisfiable problems. He shows that determining if the backbone is empty is NP-complete (Beacham 2000).

Zhang has demonstrated experimentally that there is a sharp transition in the size of the backbone of random MAX 3SAT problems (Zhang 2001). This appears to be correlated with the transition in the random 3SAT decision problem.

Ruan *et al.* (Ruan, Kautz, & Horvitz 2004) look at the *backdoor key*, the set of dependent variables within a weak backdoor. They find that the ratio of the size of backdoor key to the size of whole backdoor set is strongly correlated with problem hardness. An interesting open question is the relationship between the backdoor key and the few variables we observed that are both in the backbone and the backdoor.

Conclusion

We have studied the backbone and the backdoors of propositional satisfiability problems. We proved that backbones are hard even to approximate, and gave a number of procedures for computing backdoors. Our experiments showed that there is very little overlap between backbones and backdoors. In addition, they demonstrated that problem hardness appears to be correlated with the size of strong backdoors, and weakly correlated with the size of the backbone, but does not appear to be correlated to the size of weak backdoors nor their number.

Probably the most significant finding of this study is that backbones and backdoors do not overlap to a great extent. Backbone-guided heuristics have been demonstrated to be effective in solving SAT problems (Zhang 2004). However, our results show that such algorithms are probably not identifying backdoor sets. Heuristics based on identifying backdoor literals will likely identify different literals, and have the potential to be very effective. Second, no one general statistic appears able to predict problem hardness well. Even the size of the strong backdoor, which showed the best correlation, only explained about 60% of the variation in problem hardness.

References

Achlioptas, D.; Gomes, C.; Kautz, H.; and Selman, B. 2000. Generating satisfiable problem instances. In *Proc. of 17th Nat. Conf. on AI.*

Beacham, A. 2000. The complexity of problems without backbones. Master's thesis, Dept. of Computing Science, University of Alberta.

Cook, S. 1971. The complexity of theorem proving procedures. In *Proc. 3rd Annual ACM Symposium on the Theory of Computation*, 151–158.

Garey, M., and Johnson, D. 1979. *Computers and intractability : a guide to the theory of NP-completeness.* W.H. Freeman.

Li, C. M. 1999. A constrained-based approach to narrow search trees for satisfiability. *Information processing letters* 71:75–80.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1998. Determining computational complexity for characteristic 'phase transitions'. *Nature* 400:133–137.

Nishimura, N.; Ragde, P.; and Szeider, S. 2004. Detecting backdoor sets with respect to horn and binary clauses. In *Proc. of 7th Int. Conf. on Theory and Applications of Satisfiability Testing*.

Parkes, A. 1997. Clustering at the phase transition. In *Proc. of the 14th Nat. Conf. on AI*, 340–345.

Ruan, Y.; Kautz, H.; and Horvitz, E. 2004. The backdoor key: A path to understanding problem hardness. In *Proc. of the 19th Nat. Conf. on AI.*

Slaney, J., and Walsh, T. 2001. Backbones in optimization and approximation. In *Proc. of 17th IJCAI.*

Williams, R.; Gomes, C.; and Selman, B. 2003. Backdoors to typical case complexity. In *Proc. of the 18th IJCAI.*

Zhang, W. 2001. Phase transitions and backbones of 3-SAT and Maximum 3-SAT. In *Proc. of 7th Int. Conf. on Principles and Practice of Constraint Programming (CP2001).* Springer.

Zhang, W. 2004. Configuration landscape analysis and backbone guided local search for satisfiability and maximum satisfiability. *Artificial Intelligence* 158(1):1–26.