Backdoors in Planning and Scheduling Problems

Peter Gregory and **Derek Long** and **Maria Fox**
University of Strathclyde
Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Abstract

A backdoor is a set of assignments to a problem that yield the rest of the problem polynomially determinable. Backdoors have been shown to have interesting relationships with problem hardness, backbones and other properties.

We show deeper insight into the relationship between backdoors and backbones than has previously been shown. The effect of no-good clause learning on backdoors is also discussed.

The aim of this work is to find ways to exploit backdoors to solve problems more efficiently.

Introduction

Boolean Satisfiability (SAT) underlies the optimal STRIPS planner, Blackbox (Kautz & Selman 1999). It gained the first place in the 'optimal track' of the 2004 International Planning Competition (Hoffmann & Edelkamp 2004). The translation is effective because modern SAT-solvers incorporate sophisticated search features such as rapid-restarts and clause learning; these features in combination with efficient data-structures make SAT a high performance general problem solving framework. This work furthers previous studies on the relationship between two structures in SAT problems, the *backbone* and *backdoors*. The backbone of a SAT problem is the variables that are set in the same way in every solution. A backdoor of a SAT problem, is a set of variables that lead to the rest of the problem being polynomially determinable.

The *backdoor* has recently been seen as an important structure in SAT problems (Williams, Gomes, & Selman 2003), as it can explain how a rapid restart policy can improve chronological backtracking search. However, there are many unanswered questions about backdoors.

- How are backdoors distributed across the variable space?
- How *exactly* are backdoors and backbones related, if at all?
- How does the amount of symmetry in a problem relate to backdoor distribution?
- Does no-good clause learning affect the backdoor distribution of a problem?
- Are backdoor properties domain-independent, or are there specific *planning* backdoors, etc.

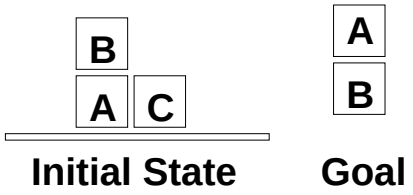

Figure 1: A blocksworld example to illustrate the major points in this paper. Can be solved in two time-steps, there are two different plans that solve the problem at this length. Note, in the goal, it is not specified that any block is on the table.

- Can backdoor variables be predicted with enough accuracy to directly solve problems with them?

The following example will be used throughout the paper as illustration of the concepts discussed. It is intentionally trivial, so that ideas can be clarified; but its structure is not so trivial that it isn't representative of harder problems. Figure 1 is a blocksworld planning problem. The goal is to reach a state where A is on B. This can be achieved in two ways: the first being 'put B on *the table* and then put A on B', the second being 'put B on *C* and then put A on B'.

Blackbox encodes STRIPS planning problems by first representing the problem as a plan-graph. The graph is then translated to a SAT instance where each variable represents either a fact (at some layer), a possible actions (including no-ops). The clauses represent the preconditions, effects, and mutexes between both facts and actions (Kautz, McAllester, & Selman 1996). The example SAT instance is the translation of the plan-graph at the first satisfiable layer.

Motivation

The aim of any academic study into the structure of problems should be to better understand that problem, and to develop novel problem solving techniques that exploit this new understanding. If backdoors can be characterised effectively, then it should be possible to search over small "candidate backdoors" at a much lower cost than searching the entire set of variables.

The challenges involved in this are manifold. A good prediction of the size of backdoor is required. Searching for an

unreasonably small backdoor would never find one, whilst searching for a backdoor much larger than the smallest ones would lead to redundant search. If we are to identify those variables in a problem that are most likely to be backdoor variables, then we need to have a good characterisation of how a backdoor is structurally composed. It is this characterisation that is detailed in this paper, along with some interesting effects that clause learning can have on backdoors. In previous research on backdoors, the backbone and backdoor sets been shown to be typically disjoint. For example, (Kilby *et al.* 2005) shows an empirical distinction between the two structures. In this work, we show analytically that there is a certain part of the backdoor that *necessarily* is not part of the backbone and that even if all of the backbone variables were set correctly, that would not be enough information to solve the problem (unless all variables are in the backbone).

SAT and Backdoors

In this work, we are concerned with the problem of Boolean Satisfiability. Boolean Satisfiability (SAT herein) is a special case of CSP. SAT restricts the domains of every variable to two values, $true$ and $false$. It also restricts the constraints to a set of clauses. A clause is a disjunction of literals. The variables correspond to logical variables, and the clauses disjunctions of logical literals, rather than writing $x_i = true$ and $x_i = false$, we will use the shorthand x_i and $\neg x_i$ instead. We will also occasionally refer to x_i and $\neg x_i$ as being in positive and negative *phase* respectively.

A sub-solver is an algorithm that solves a tractable subproblem of the general problem class. Paraphrasing Garey and Johnson (Garey & Johnson 1979), a subproblem of a general problem is obtained whenever we place additional restrictions on the allowed instances of that problem class. A sub-solver is an algorithm that determines only problem instances of a given subproblem.

The backdoor structure is reliant on the definition of a *sub-solver*, A, that has the following properties (given as input a CSP, C):

Trichotomy A either rejects the input C, or "determines" C correctly.

Efficiency A runs in polynomial time.

Trivial Solvability A can determine if C is trivially true (has no constraints) or trivially false (has a contradictory constraint).

Self-Reducibility If A determines C, then for any variable x, and value v, then A determines $C[v/x]$. (Williams, Gomes, & Selman 2003)

The informal definition of a backdoor is those variables which lead to a solution in polynomial time, when assigned correctly, or prove no solution exists for unsatisfiable problems. A *weak* backdoor can determine if a problem is satisfiable. That is, a set of variables wBD, for which there is at least one assignment such that A returns a satifying assignment. A *strong* backdoor can determine both satisfiability and unsatisfiability. For an unsatisfiable instance, a strong backdoor is defined as a set of variables, such that each assignment to those variables leads to A determining the instance unsatisfiable.

The definition of a backdoor requires a polynomial-time *sub-solver* (A in the definition). In this work, the sub-solver is assumed to be unit-propagation, and the CSP is always a SAT instance. Even when restricted to SAT, the sub-solver need not be unit propagation, it could be an algorithm that solves only 2SAT problems, or an algorithm that solves only horn-SAT problems, for example. The work will be as general as possible, as the concepts discussed (backdoors, backbones, no-good learning) are ubiquitous in CSP, planning, scheduling, and all search problems that can be modelled using constraints. Whenever I refer to *backdoors* from here, I refer to *minimal backdoors* as these are more interesting to study (The entire set of variables is trivially a non-minimal backdoor but it isn't a very interesting one, for example.)

Distribution of Backdoors

If we can find all of the minimal backdoors in a problem, we can easily calculate the total number of backdoors in the problem. As this is computationally prohibitive, we can use a sampling method to find representatives from the global set of minimal backdoors. The algorithm we use to find these is as follows:

```
ALGORITHM: MINIMAL BACKDOORS
-------------------------------------
1. s <- number of variables

2. while (!cutoff_limit){
3.  BD <- pick random variables (size s)
4.  backtrack over BD, if backdoor then{
5.   minBD <- MinimiseBackdoor(BD)
6.   s' <- |minBD|
7.   if (s' < s)
8.     s <- s'
9.   BDlist <- BDlist U {minBD}
    }
   }
-------------------------------------
```

The algorithm seeds the size of the backdoor to be the size of the instance. Then, while a cutoff limit is not reached, new backdoor candidates are selected. The algorithm backtracks over the variables in the candidate, if a solution is found (or unsatisfiability proven), we minimise the backdoor.

Minimisation is achieved by simply removing each variable, in order, and testing if the remaining structure is a backdoor. If it is, then the variable is not part of the minimal backdoor, and is discarded. If not, then the variable is reintroduced into the candidate. This is similar to the MINWEAKBACKDOOR algorithm in (Kilby *et al.* 2005). The difference being, that algorithm used *literals* and not variables as the constituents of the backdoor. This means that different instantiation of the variables in their backdoors could give a smaller *weak* backdoor. It also means that the (Kilby *et al.* 2005) procedure cannot detect (or minimise) *strong* backdoors. It would however have better runtime performance than MINIMAL BACKDOORS.

Problem	#Backdoors	Variables	Median BD Size	Backbone Size	Median BB Overlap
qg1-07	8	343	5.5	189	0
qg2-07	11	343	5	169	0
qg7-09	13	729	2	505	0
bw-medium	30	116	2	97	1
bw-large.a	31	459	3	459	3
bw-huge	38	459	3	459	3
flat30-5	9771	90	5	0	0
flat75-5	141	225	12	0	0

Table 1: Table of statistics for the studied instances. The studied instances are 3 quasi-group completion problems, 3 blocksworld problems, and 2 graph colouring problems, each from the satlib benchmark suite. The results show the number of minimal backdoors found in 10 minutes using the above algorithm.

Let us consider the blocksworld instance. Using unit propagation as the subsolver, there are two minimal backdoors. These relate to the actions (stack B C) and (move-to-table B) (both at timepoint 1). This is because, in two steps, the goal can be achieved by *either* putting B on the table or on block C first, then stacking A on B. Once we have decided which option to take, the mutex between the two actions causes propagation to imply the other one false. Everything else then propagates from this decision.

The Backbone and Backdoors

The backbone of a SAT instance is the set of variables that are implied by the model. More intuitively, it is the set of variables that take *the same* assignments in every solution. There are two variables in the SAT encoding of our example that are not in the backbone. These variables correspond to the actions (stack B C) and (move-to-table B). These have already been identified as the backdoor variables, no backbone variables are backdoor variables in this instance.

It has been previously observed that backdoor variables are not often backbone variables (Kilby *et al.* 2005). There is occasionally an intersection between the two structures, but it appears accidental. So a better question is: what is the reason that backbones and backdoors appear to be (typically) disjoint? Let us start by making some observations.

If all of the backbone variables are set correctly, could this be a backdoor? No. The backbone variables are those whose assignments are implied by the problem. Thus, if setting the backdoor correctly implied another variable/value assignment, this other variable *must be* in the backbone also. Once we have this piece of information, we can see that partial/ full assignments to backbone variables only have the capacity to imply *other backbone variables*. Since a backdoor implies *every* variable's value for a given solution, the backbone variables cannot be a backdoor.

As variables are assigned in search, the sub-spaces that we move into have monotonically growing backbones. Indeed, when the problem is solved using assignment and propagation, all of the variables are trivially in the 'backbone' (as in the final state all variables are set). Since we have shown backbone variables can only imply themselves, it is true that in any sub-space of the search tree, an algorithm would not want to make the choice of next variable one which is in the augmented backbone, as this can't imply any variables other than those in the augmented backbone.

Identifying Unique Solutions

When all variables are in the augmented backbone, then there is a single solution (in that sub-space). This doesn't mean that search is necessarily complete: some problems with single solutions can be hard to solve. But it does mean the problem is simplified to a state where it *might* be possible to solve the problem using propagation, because there is now a single solution in the sub-space.

Therefore, we have a necessary, but not sufficient, property of any backdoor – *assignment of part of the backdoor must identify a unique solution*. The next enquiry naturally concerns the question: how is the remainder of the backdoor composed? In this situation, several variables have been assigned such that, in the current sub-space, there is a unique solution to the studied instance (but the problem is not solved). However, there is not enough information in the current clauses to cause propagation of the remaining variables. One reason this can happen is that there are cyclic relationships in the clauses. Once a unique solution is found, if there remain cycles in the constraint graph, it may be necessary to 'cut' them in order to finish solving the problem. This is equivalent to finding the *cycle cutset* (Dechter 2003) of the remaining problem after a unique solution has been identified.

Another conjecture could be posited: if we had enough inferred knowledge of the problem at hand, then *no* backdoors would contain any backbone variables. This claim may seem unlikely, but some preliminary work has been carried out to suggest that it may not be. The conjecture centres around *inferred knowledge*. What *exactly* has to be inferred from the problem to reduce the size of backdoors? A "perfect" set of clauses can be imagined. With these clauses, any partial assignment that is made would lead to the entire augmented backbone being propagated. This would mean that the problem would have a unit clause for each variable in the backbone. If we actually had the "perfect" set of clauses, we could make any assignment in full confidence that the assignment led to a solution. Clearly we could never find such a model in reasonable time. But there are ways of bringing our model closer to this "perfect" model during search. We

can infer extra knowledge of a SAT instance using conflict-clause learning.

Clause Learning and Backdoors

Many SAT solvers make use of conflict clause learning. What happens to the backdoors in a problem as new clauses are learnt? It appears that as we learn new information, the size of a typical backdoor to a problem reduces. To illustrate how this happens, consider the following tiny SAT instance:

$(1 \vee 2) \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 1 \vee 2 \vee 3) \wedge (\neg 1 \vee 2 \vee \neg 3)$

In this example 2 is the only backbone member ($\neg 2$ implies 1 which in turn implies $3 \wedge \neg 3$, a contradiction, hence 2 is true in all solutions). There are two minimal backdoors to the instance, (1) and $(2, 3)$ (this is because 3 alone cannot cause any propagation).

The effect of choosing $\neg 2$ as the first decision, causes the contradiction on variable 3. Using FirstUIP conflict analysis, the generated conflict clause would be simply the unit-clause (2). With this clause in place, one of the previous minimal backdoors is now *non-minimal*. The backdoor $(2, 3)$ need not have 2 in it, as this is already implied by the conflict clause. Clause learning has reduced the size of the largest minimal backdoor and removed the backdoor values from it.

This one example only shows that clause learning can be used to reduce the size of backdoors.

Results and Discussion

The results in Table 1 were found by running MINIMAL BACKDOORS on each problem instance for 10 minutes. There are three classes of problems studied. The first are quasigroup completion problems. These are partially complete latin squares with additional constraints. These are useful in experimental design, for example scheduling a drugs trial. The second class of problem is the blocksworld planning problem. The third class of problem are graph colouring instances. The problems are benchmarks picked from the satlib web resource (Hoos & Sttzle 2000).

It is interesting to note the fact that the planning instances have very large backbones. This is because of the the fact that at the first satisfiable plan graph layer, the number of valid solutions to these blocksworld problems is extremely limited. The median backdoor sizes for every instance studied is tiny in proportion to the total number of variables in the respective problems. This is what was expected, and further indication that direct exploitation of backdoors is a promising research avenue.

Conclusions and Future Work

Backdoors and backbones are related structures, even though they do not often contain the same variables as each other. A backdoor causes the backbone to "grow" so that it covers all of the variables. The rest of the backdoor is composed of assignments that "fill-in" the missing information that renders the problem easily soluble. It also appears that clause learning can reduce the size of backdoors. Immediate future work includes rigourous empirical analysis of the relationship between backdoors and backbones; and between backdoors and clause learning.

Algorithms that predict which variables are backbone variables (Dubois & Dequen 2001) have been used previously to guide DPLL based search. If the algorithm they use can successfully predict backbone variables, then these variables are unlikely to be backdoor variables. Once the likely backdoor variables are found, then candidate backdoor sets can be generated from these, and tested using backtracking.

In DPLL based search, certain types of learnt clause are not useful. The decision variables are never used to generate conflict clauses as the same assignments will never again be visited in chronological search. However, this changes with restarts occurring, and will be just as important with search using backdoors. Which type of clause learning will be most useful in the development of a solver that exploits backdoors directly isn't clear, and is worthy of further work.

The final goal of this work is to create a SAT Solver that uses analysis of the structure of problems to find backdoors efficiently, so that problems can be solved faster and and problems that are currently out of reach of current solvers can be solved.

References

Dechter, R. 2003. *Constraint Processing*. Morgan Kauffman.

Dubois, O., and Dequen, G. 2001. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In Nebel, B., ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 248–253. Morgan Kaufmann.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Hoffmann, J., and Edelkamp, S. 2004. International planning competition. http://ipc.icaps-conference.org/.

Hoos, H. H., and Sttzle, T. 2000. SATLIB: An Online Resource for Research on SAT. In I.P.Gent; H.v.Maaren; and T.Walsh., eds., *Proceedings of the Third International Conference on the Theory and Applications of Satisfiability Testing*, 283–292. IOS Press.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Kautz, H. A.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, 374–384.

Kilby, P.; Slaney, J.; Thiebaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proceedings of AAAI-2005*.

Williams, R.; Gomes, C.; and Selman, B. 2003. Backdoors to typical case complexity.