Structure and Problem Hardness: Asymmetry and DPLL Proofs in SAT-Based Planning

Jörg Hoffmann\(^1\), Carla Gomes\(^2\), and Bart Selman\(^2\)

\(^1\) Max-Planck-Institute for CS, Saarbrücken, Germany
\(^2\) Cornell University, Ithaca, NY, USA

Abstract. In applications from AI Planning and Model-Checking, a successful method is to compile the application task into boolean satisfiability (SAT), and solve it with state-of-the-art DPLL-based procedures. There is a lack of formal understanding why this works so well. Focussing on the AI Planning context, we identify a structural parameter, called AsymRatio, that measures a kind of subgoal asymmetry in planning tasks. AsymRatio ranges between 0 and 1, and we show empirically that it correlates strongly with SAT solver performance in a broad range of AI Planning benchmarks, namely the domains used in the 3rd International Planning Competition. We then examine carefully crafted synthetic planning domains that allow to control the value of AsymRatio, and that are clean enough to allow a rigorous analysis of the combinatorial search space, while meaningful enough to allow conclusions about more practical domains. The domains are parameterized by size \(n\), and by a structure parameter \(k\), so that AsymRatio is asymptotic to \(k/n\). We investigate the best (smallest) possible sets of branching variables for DPLL, as a function of \(n\), for different settings of \(k\). With minimum \(k\), we identify minimal sets of branching variables linear in the total number of variables, \(\Theta(n^2)\). With maximum \(k\), we identify sets of size \(O(\log_2 n)\), and thus size \(O(n)\) DPLL proofs.

1 Introduction

There has been a long interest in a better understanding of what makes combinatorial problems hard or easy. The most successful work in this area involves random instance distributions with phase transition characterizations (e.g., [1, 2]). However, the link of these results to more structured instances is less direct. A random unsatisfiable 3-SAT instance from the phase transition region with 1,000 variables is beyond the reach of any current solver. But many unsatisfiable formulas from verification and planning contain well over 100,000 variables and can be proved to be unsatisfiable within a few minutes (e.g., with Chaff [3]). This raises the question as to whether one can obtain general measures of structure in SAT encodings, and use them to characterize typical case complexity. To this end, our overall goal in this paper is to identify general problem features that characterize problem hardness in practice. We focus on formulas from AI planning. We view this as an entry point to similar studies in other areas.

The main spirit of our work is a two-step approach: first, identify a measure of “structure” that, empirically, correlates with CSP/SAT solver performance in practical benchmarks; then, design synthetic domains that capture this structure.
in a clean form, and analyze the behavior of DPLL (or any other search algorithm of interest), within these synthetic domains, in detail. The latter step serves to obtain a deeper understanding of what causes the empirical correlation observed in the first step. For this to make sense, the synthetic domains have to be simple enough to be rigorously analyzed, yet meaningful enough to allow conclusions about more practical domains. We remark that, while under development, the two research steps may well be – and have been, in our case – intermingled: increasingly accurate intuitions are obtained in a trial-and-error fashion.

Note that our approach is very different from identifying tractable classes. Generally, our research is aimed at understanding the behavior of existing algorithms, not at identifying new algorithms. More technically, the first research step outlined above establishes an empirical correlation between structure and performance. The second research step may, or may not, yield results on polynomial best-case or worst-case behavior. But even if so, these results hold only for the specific examples (synthetic domains) considered. In that sense, the analytical step is merely a case-study, aimed at obtaining more accurate intuitions.

We focus on showing infeasibility. Precisely, we consider the difficulty of showing the non-existence of a plan with one step less than the shortest possible (optimal) plan. SAT-based search for a plan works by iteratively incrementing a plan length bound $b$, and testing in each iteration a formula that is satisfiable iff there exists a plan with $b$ steps (this was first implemented in the Blackbox system [4]). So, our focus is on the last unsuccessful iteration in a SAT-based plan search. This is typically the hardest iteration in practice. SAT-based planning is currently state-of-the-art for finding optimal plans: e.g., Blackbox won the 1st prize for optimal planners in the 4th International Planning Competition [5].

We consider SAT encodings of our synthetic domains and investigate the best possible sets of branching variables for DPLL proof trees. Such variable sets were recently coined “backdoors” [6]. In our context, a backdoor is a subset of the variables so that, for every value assignment to these variables, unit propagation (UP) yields an empty clause.\(^3\) That is, a smallest possible backdoor encapsulates the best possible branching variables for DPLL, a question of huge practical interest. Also, the size of the backdoor provides an upper bound on the size of the DPLL search tree: if the backdoor contains $l$ variables, then the maximum number of nodes in the proof tree is $2^l$. In particular, if $l$ is logarithmic in the formula size, then there exists a polynomial size DPLL proof. In all considered formula classes, we determine a backdoor subset of variables. We prove that the backdoors are minimal: no variable can be removed without losing the backdoor property. In small enough instances, we prove empirically that the backdoors are in fact optimal - of minimal size. We conjecture that the latter is true in general.

Our synthetic planning domains are (1) a logistics planning domain (MAP) and (2) a stacking domain (SBW). We also consider a third synthetic domain

---

\(^3\) In general, a backdoor is defined relative to an arbitrary polynomial time “subsolver” procedure. The subsolver can solve some class of formulas that does not necessarily have a syntactic characterization. Our definition here instantiates the subsolver with the widely used unit propagation procedure.
called SPH, a structured version of the Pigeon Hole problem. The domains are characterized by a size parameter, called \( n \), and by a structure parameter, called \( k \). The structure parameter controls the amount of an intuitive “asymmetry” in the underlying task: as the value of \( k \) increases, one part of the task becomes more and more difficult to achieve, while the other parts become relatively easier. Concretely, in Planning, we define the parameter \( \text{AsymRatio} \) as the ratio between maximum sub-goal difficulty – the maximum number of steps needed to achieve any single sub-goal – and the overall difficulty, i.e., the number of steps needed to achieve the conjunction of all goals. \( \text{AsymRatio} \) ranges between 0 and 1. A value close to 1 represents a large structural asymmetry. In MAP and SBW, \( \text{AsymRatio} \) is asymptotic to the ratio between \( k \) and \( n \). In particular, for the lowest value of \( k \) (symmetrical case), \( \text{AsymRatio} \) converges to 0 for increasing \( n \); for the highest \( k \) value (asymmetrical case), the ratio converges to 1. Note that such a high amount of asymmetry appears unlikely to occur in purely randomly generated problem instances. Note further that \( \text{AsymRatio} \) characterizes a kind of hidden structure. It can not be computed efficiently even based on the original planning task representation. Much less is it evident to a satisfiability tester attacking the CNF representation of the task, without even knowing that the formula originates from a planning task. More concretely, the constraint graphs of our formulas (more on this below) generally don’t change much over \( k \).

In some initial experiments, we observed that a high value of \( \text{AsymRatio} \) enables the (unsatisfiable) formulas to be effectively solved by current SAT solvers. Investigating this in our synthetic domains, we found dramatic differences in backdoor size. At the bottom ends of the \( k \) scales, with symmetrical subgoals, the backdoor sets are of polynomial size (in \( n \)) in all cases. With increasing value of \( k \), the backdoors become smaller; in the two synthetic planning domains, at the top end of the \( k \) scales, the backdoors are of logarithmic size.\(^4\)

To confirm that \( \text{AsymRatio} \) correlates with SAT solver performance in practice (i.e., in more complex benchmarks than our synthetic domains), we ran large-scale experiments in the six benchmark domains used in the 3rd International Planning Competition [7]. This is a recent (published in 2002) and widely used set of benchmarks, and is provided, by the IPC-3 organizers, with instance generators; the latter are essential for our experiments, where we generated and examined tens of thousands of instances in each domain.\(^5\) We plotted the performance of a state-of-the-art SAT solver, namely, ZChaff [3], as a function of \( \text{AsymRatio} \). Our experiments show that a larger \( \text{AsymRatio} \) results in planning CNFs that are significantly easier to solve. \( \text{AsymRatio} \) thus provides a useful indicator of typical problem hardness for Planning domains.\(^6\) This is of course

\(^4\) It is important to note that we obtain logarithmic size backdoors. This suggests that our underlying planning problems do not become “trivial” — in particular, they still require some subtle branching choices of the DPLL procedure, and are not just solved by unit propagation.

\(^5\) For the domains used in the 4th International Planning Competition [8], run in 2004, there are no random generators.

\(^6\) While \( \text{AsymRatio} \) can not be computed efficiently, there exists a variety of techniques to approximate the number of steps needed to achieve a goal (e.g., [9–11]).
just a first example of a hardness measure for structured problems; presumably, other useful measures exist.

The investigation of structure in constraint reasoning problems is not new, see for example [12–20]. However, to the best of our knowledge, our particular approach – to empirically identify a relevant structural parameter and then analyze that in synthetic domains – has not been pursued before. Still, one structural concept is particularly closely related to the concept of a backdoor, and should be discussed in more detail: cutsets (e.g. [12–14]). A cutset is a set of variables so that, once these variables are removed from the constraint graph – the undirected graph where nodes are variables and edges indicate common membership in at least one clause – that graph has a property that enables efficient reasoning: an induced width of at most a constant bound \( b \) (if \( b = 1 \) then the graph is cycle-free, i.e., can be viewed as a tree). Backdoors are a generalization of cutsets in the sense that any cutset is a backdoor relative to an appropriate subsolver (that exploits properties of the constraint graph).\(^7\) The main difference between a backdoor and a cutset is, from a general point of view, that, given a set of variables, one can determine in polynomial time whether or nor that set is a cutset. The same test is, in general, not possible for a backdoor. In particular, it is not possible for the unit propagation procedure we consider here, that depends heavily on what values are assigned to the backdoor variables. In that sense, a cutset is a backdoor that can be detected statically, and that can thus be directly exploited in a search algorithm. Backdoors in general only provide a parameter measuring properties of search spaces. We will see that, in the particular formula families we consider here, there are no small statically detectable cutsets. Indeed, as we detail below, the constraint graphs do generally not change much with \( k \) and are thus not suitable to capture what happens on the structural scale. In that sense, the structure in our formulas is “hidden”.

In Section 2, we provide background on AI planning and the SAT encodings we use. In Section 3, we present our empirical findings showing the relevance of AsymRatio as a measure of problem hardness in structured domains. In Section 4, we describe our synthetic domains and our analysis of backdoors. Section 5 provides a summary of results and directions for future research.

## 2 Background

We consider the “STRIPS” formalism. States are described as sets of (the currently true) propositional facts. A planning task is a tuple of initial state (a set of facts), goal (also a set of facts), and a set of actions. Actions \( a \) are fact set triples: the precondition \( \text{pre}(a) \), the add effect \( \text{add}(a) \), and the delete effect \( \text{del}(a) \). The semantics are that an action is applicable to a state (only) if \( \text{pre}(a) \) is contained in the state. When executing the action, the facts in \( \text{add}(a) \) are included into the state, and the facts in \( \text{del}(a) \) are removed from it (the intersection between

\(^7\) We thank Rina Dechter for insightful discussions on this issue.
add(a) and \( \text{del}(a) \) is assumed empty; executing a non-applicable action results in an undefined state). A plan for the task is a sequence of actions that, when executed iteratively, maps the initial state into a state that contains the goal.

Planning can be mapped into a sequence of SAT problems, by incrementally increasing a plan length bound \( b \): start with \( b = 0 \); generate a CNF \( \phi(b) \) that is satisfiable iff there is a plan with \( b \) steps; if \( \phi(b) \) is satisfiable, stop; else, increment \( b \) and iterate. This process was first implemented in the Blackbox system [4].

There are, of different, different ways of generating the formulas \( \phi(b) \), i.e., there are different encoding methods. In our empirical experiments, we use the original Graphplan-based encoding used in Blackbox. In our theoretical investigations, we use a somewhat simplified version of that encoding.

The Graphplan-based encoding is a straightforward translation of a \( b \)-step planning graph [9] into a CNF. The encoding has \( b \) time steps \( 1 \leq t \leq b \). It features variables for facts at time steps, and for actions at time steps. There are artificial NOOP actions, i.e. for each fact \( p \) there is an action NOOP-\( p \) whose only precondition is \( p \), and whose only (add) effect is \( p \). The NOOPs are treated just like normal actions in the encoding. Amongst others, there are clauses to ensure that all action preconditions are satisfied, that the goals are true in the last time step, and that no “mutex” actions are executed in the same time step.\(^8\) The set of fact and action variables at each time step, as well as pairs of “mutex” facts and actions, are read off the planning graph (which is the result of a non-trivial propagation of constraints).

We do not describe the Graphplan-based encoding in detail since that is not necessary to understand our experiments. For the simplified encoding used in our theoretical investigations, some more details are in order. The encoding uses variables only for the actions, i.e., \( a(t) \) is 1 iff action \( a \) is to be executed at time \( t \), \( 1 \leq t \leq b \). A variable \( a(t) \) is included in the CNF iff \( a \) is present at \( t \). An action \( a \) is present at \( t = 1 \) iff \( a \)'s precondition is true in the initial state; \( a \) is present at \( t > 1 \) iff, for every \( p \in \text{pre}(a) \), at least one action \( a' \) is present at \( t - 1 \) with \( p \in \text{add}(a') \). For each action \( a \) present at a time \( t \) and for each \( p \in \text{pre}(a) \), there is a precondition clause of the form \( \neg a(t) \lor a_1(t - 1) \lor \ldots \lor a_l(t - 1) \), where \( a_1, \ldots, a_l \) are all actions present at \( t - 1 \) with \( p \in \text{add}(a_i) \). For each goal fact \( g \in G \), there is a goal clause \( a_1(b) \lor \ldots \lor a_l(b) \), where \( a_1, \ldots, a_l \) are all actions present at \( b \) that have \( g \in \text{add}(a) \). Finally, for each incompatible pair \( a \) and \( a' \) of actions present at a time \( t \), there is a mutex clause \( \neg a(t) \lor \neg a'(t) \). Here, a pair \( a, a' \) of actions is called incompatible iff either both are not NOOPs, or \( a \) is a NOOP for fact \( p \) and \( p \in \text{del}(a') \) (or vice versa).

We interpret CNF formulas as sets of clauses, where each clause is a set of literals. For a CNF formula \( \phi \) with variable set \( V \), a variable subset \( B \subseteq V \), and a value assignment \( a \) to \( B \), we say that \( a \) is UP-consistent if applying \( a \) to (the literals in) \( \phi \), and performing unit propagation on the resulting formula, does not yield an empty clause. \( B \) is a backdoor if it has no UP-consistent assignment.

\(^8\) Actions can be executed in the same time step if their effects and preconditions are not contradictory.
3 Asymmetric Structure in Planning

As discussed above, we quantify subgoal asymmetry as follows.

**Definition 1.** Let $P$ be a planning task with goal $G$. For each fact $g \in G$, let $\text{cost}(g)$ denote the length of a shortest plan achieving just $g$; let $\text{cost}(G)$ denote the length of a shortest plan achieving all facts in $G$. The asymmetry ratio of $P$ is:

$$\text{AsymRatio}(P) := \frac{\max_{g \in G} \text{cost}(g)}{\text{cost}(G)}$$

Note that $\text{cost}(G)$, in this definition, is the optimal plan length: to simplify notation, we will henceforth denote this with $m$. Note also that, of course, a definition as simple as Definition 1 can not be fail-safe. Imagine replacing $G$ with a single goal $g$ and an additional action with precondition $G$ and add effect \{g\}: the (new) goal is then no longer a set of “subgoals”. However, in the benchmark domains that are actually used by researchers to evaluate their algorithms, $G$ is almost always composed of several goal facts, and the single goal facts correspond quite naturally to different sub-problems of the task.\(^9\) Our hypothesis in the experiments is:

**Hypothesis 1** Let $\mathcal{P}_m$ be a set of planning tasks from the same domain with the same size parameter values, and with the same optimal plan length $m$. For $P \in \mathcal{P}_m$, let $\phi(P, m-1)$ denote the Graphplan-based CNF encoding of $m-1$ action steps. Then, over $\mathcal{P}_m$, the hardness of proving $\phi(P, m-1)$ unsolvable is strongly correlated with $\text{AsymRatio}(P)$.

First, note that, certainly, whether this hypothesis holds or not depends on the domain; in that sense it is a different hypothesis for every domain. Second, note that the instance size parameter values (nr. of vehicles for transportation, e.g.), together with the number of action steps encoded – the optimal plan length minus 1 – determine the size of the formula. Of course, formula size is typically correlated with SAT solver performance. Our hypothesis concerns performance in formulas of similar size. Please note that we do not wish to imply that $\text{AsymRatio}$ is “the” parameter predicting SAT solver performance in Planning CNFs. There are, presumably, many important factors and interplay between them. Our (only) observation, below, is that $\text{AsymRatio}$ works well in an important range of domains.

To test our hypothesis, as said, we ran large experiments in all STRIPS domains used in the 3rd International Planning Competition [7] (IPC-3), as was carried out in 2002. The domains are called Depots, Driverlog, Freecell, Rovers, Satellite, and Zenotravel. Depots is a mixture between the classical Blocksworld and Logistics domains; Blocksworld requires arrangement of blocks in stacks on a

\(^9\) A more stable approach would be to identify a hierarchy of layers of “landmarks” [21], and define $\text{AsymRatio}$ based on that. In the benchmarks, because of what we just said, this does not seem to add much value. Exploring the issue in more depth is a topic for future work.
Fig. 1. Log-scaled mean number of backtracks needed by ZChaff, plotted over AsymRatio, in CNF formulas encoding planning instances from the IPC-3 benchmark Rovers. Curves for different subsets of more than 40000 randomly generated instances: all instances with optimal plan length 7, all instances with optimal plan length 8, and all instances with optimal plan length 9. Entire distribution of optimal plan length is 4 . . . 19; 7, 8, and 9 are the most frequent, and together contain 60% of all instances.

Table, using a robot arm; Logistics requires transportation of packages via trucks and airplanes; in Depots, blocks must be transported and arranged in stacks. Driverlog is a version of Logistics with drivers, where drivers and trucks move on different (arbitrary) road maps. Freecell encodes the well-known solitaire card game where the task is to re-order a random arrangement of cards, following certain stacking rules, using a number of “free cells” for intermediate storage. Rovers and Satellite are simplistic encodings of NASA space-applications. In Rovers, rovers move along individual road maps, and have to gather data about rock or soil samples, take images, and transfer the data to a lander. In Satellite, satellites must take images of objects in space, which involves calibrating cameras, turning the right direction, etc. Zenotravel is a version of Logistics where moving a vehicle consumes fuel that can be re-plenished using a “refuel” operator. It is important to note that, within each of the IPC-3 domains, deciding bounded plan existence — the problem encoded by our CNFs — is NP-hard [22]. So our experiments are on challenging, if not real-world realistic, problems.

To obtain a reliable picture of how a complex DPLL-based SAT solver (ZChaff) typically behaves in CNF formulas generated from a domain, within each domain we generated and examined tens of thousands of instances. We chose the instance size parameters by testing the original IPC-3 instances, and selecting the largest one for which we could compute AsymRatio reasonably fast.\textsuperscript{10} E.g. in Driverlog we selected the instance indexed 9 out of 20 (instance size here scales with growing index), and, accordingly, generated random instances with 5 road junctions, 2 drivers, 6 packages, and 3 trucks. According to the setup in

\textsuperscript{10} That computation was done by a combination of calls to Blackbox [4].
Hypothesis 1 (we also use the notations), within each domain we assigned the instances to sub-sets $P_m$ with identical optimal plan length $m$. For each $P$ in a set $P_m$, we computed $\text{AsymRatio}(P)$, and ran ZChaff[3] on the formula $\phi(P, m-1)$, measuring the number of backtracks. We plotted the latter over $\text{AsymRatio}$ by dividing each $P_m$ into 100 bins, with $\text{AsymRatio}(P) \in [0, 0.01), \ldots, [0.99, 1]$; we took the mean value out of each bin, avoiding noise by skipping bins with less than 100 elements. The results are in Figures 1 and 2. (Plots for medium values are almost identical.)

For the Rovers domain, Figure 1 clearly shows the hypothesized correlation within each of the displayed subsets $P_m$, $m \in \{7, 8, 9\}$. Note that, from the relative positions of the different curves, one can see the influence of optimal plan length/formula size — the higher $m$, the more backtracks are needed. These observations are also typical for the other IPC-3 domains. Figure 2 shows the plots, which clearly support Hypothesis 1.\(^{11}\)

\(^{11}\) There is no plot for Satellite because, there, due to a lack of variance in subgoal hardness, all instances within the sets $P_m$ have the same $\text{AsymRatio}$. 

Fig. 2. Mean number of backtracks of ZChaff, plotted against $\text{AsymRatio}$, in CNF formulas encoding planning instances from the IPC-3 benchmark domains except Satellite (see text), and Rovers (which is displayed in Figure 1). Curves for different subsets $P_m$ of around 50000 random instances in each domain: the subsets corresponding to the 3 most frequently occurring optimal plan lengths $m$. For all domains except Zenotravel, the curves are shown separately for each $m$. For Zenotravel, in each $P_m$ there are at most two bins with over 100 instances; so the curve is for the union of $P_6$, $P_7$, and $P_8$. 

For the Rovers domain, Figure 1 clearly shows the hypothesized correlation within each of the displayed subsets $P_m$, $m \in \{7, 8, 9\}$. Note that, from the relative positions of the different curves, one can see the influence of optimal plan length/formula size — the higher $m$, the more backtracks are needed. These observations are also typical for the other IPC-3 domains. Figure 2 shows the plots, which clearly support Hypothesis 1.\(^{11}\)
4 Asymmetric Structure in Synthetic Domains

As said, in order to obtain a deeper understanding of the observed correlation, we studied three classes of synthetic formulas, called MAP, SBW, and SPH. MAP and SBW come from planning domains, SPH is a structured version of the pigeon hole. Each of the formula classes/domains is parameterized by size \( n \) and structure \( k \). In the planning domains, we use the simplified Graphplan-based encoding (see Section 2), and consider CNF formulas that are one step short of a solution. We denote the formulas with \( MAP^k_n \), \( SBW^k_n \), and \( SPH^k_n \), respectively.

Due to space restrictions, we consider only MAP in detail, and we omit all proofs. The missing informations are available in a technical report [23]. We remark that the proofs are rather involved (the MAP proofs alone occupy 9 pages in this format), due to the many details one needs to take account of when determining the effects of UP in complicated formulas.

**MAP.** In the MAP domain, one moves on the road map graph, parameterized by \( n \), shown in Figure 3 (a) and (b). The available actions take the form \( move\-x\-y \), where \( x \) is connected to \( y \) with an edge in the graph. The precondition is \{\( at\-x \)\}, the add effect is \{\( at\-y, visited\-y \)\}, and the delete effect is \{\( at\-x \)\}. Initially one is located at \( L^0 \). The goal is to visit a number of locations. **What** locations must be visited depends on the value of \( k \in \{1,3,\ldots,2n-3\} \). If \( k = 1 \) then the goal is to visit each of \{\( L^1_1,\ldots,L^1_n \)\}. For each increase of \( k \) by 2, the goal on the \( L^1_1\)-branch goes up by two steps, and one of the other goals is skipped. For \( k = 2n-3 \), the goal is \{\( L^{2n-3}_1, L^1_1 \)\}. We refer to \( k = 1 \) as the **bottom case**, and to \( k = 2n-3 \) as the **top case**, see Figure 3 (a) and Figure 3 (b), respectively.

The length of a shortest plan is \( 2n-1 \) independently of \( k \); our CNFs encode \( 2n-2 \) steps; \( AsymRatio \) is \( \frac{k}{2n-1} \). Figure 3 (c) and (d) illustrate that the setting of \( k \) has a quite drastic effect on backdoor size. We will detail this below. First, observe that the setting of \( k \) has only very little impact on the size and shape of the constraint graph, illustrated in Figure 3 (e) and (f). Between formulas \( MAP^k_n \) and \( MAP^{k'}_n \), \( k' > k \), there is no difference except that \( k' - k \) goal clauses are skipped, and that the content of the goal clause for the \( L^1_1\)-branch changes. Precisely, the number of clauses in \( MAP^k_n \) is \( 3n^3 + 27n^2 - 73n + 39 - (k+1)/2 \). The number of variables is \( 16n^2 - 33n + 14 \), irrespectively of \( k \). Also irrespectively of \( k \), the constraint graph contains, at each time step \( 2 \leq t \leq 2n-2 \), large cliques of variables, for example the \( 2n \) variables corresponding to moves from or to \( L^0 \), which are fully connected due to the mutex clauses. From a clique of size \( m \), one has to remove \( m-1-b \) nodes in order to get to an induced width of \( 1 \leq b \leq m-1 \). Since the mentioned cliques are disjoint, this shows that, for any constant \( b \), the \( b \)-cutset size in \( MAP^k_n \) is a square function in \( n \), irrespectively of \( k \). Details, also on other kinds of cutsets, are in the TR [23].

The hidden structure in our formulas can not be characterized in terms of \( b \)-cutsets. It can be characterized in terms of the effects of unit propagation. For the bottom case, we identify a backdoor called \( MAP^1_n B \), defined as follows:

\[ \text{For } k = 2n-1 \text{, } MAP^k_n \text{ contains an empty clause: no supporting action for the goal is present at the last time step.} \]
Fig. 3. Goals, backdoors, and constraint graphs in MAP. In (a) and (b), goal locations are indicated in bold face, for the bottom end (a) and the top end (b) of the \( k \) scale. In (c) and (d), the horizontal axis indicates branches in the map, and the vertical axis indicates time steps; abbreviations: “NA-0” for \( \text{NOOP} - L^0(1) \), “MV-i” for \( \text{move-}L^0-L^1_i \), “NV-i” for \( \text{NOOP-visited-}L^1_i \), and “MV-23” for \( \text{move-}L^2-L^3_1 \). In (e) and (f), the variables at growing time steps lie on circles with growing radius. Edges indicate common membership in at least one clause. Stepping from (e) to (f), three edges within the outmost circle disappear (one of these is visible on the left side of the pictures, just below the middle) and one new edge within the outmost circle is added.

\[
MAP^*_B := \{ \text{move-}L^0-L^1_i(t) \mid t \in T, 2 \leq i \leq n \} \cup \\
\{ \text{NOOP-visited-}L^1_i(t) \mid t \in T, 3 \leq i \leq n \} \cup \\
\{ \text{NOOP-at-}L^0(1) \} \cup \\
\{ \text{move-}L^0-L^1_i(t) \mid t \in T \setminus \{2n-5, 2n-3\} \}
\]

Here, \( T = \{3, 5, \ldots, 2n-3\} \). Compare Figure 3(c). The size of \( MAP^*_B \) is \( \Theta(n^2) \).
**Theorem 1 (MAP bottom case, backdoors).** Let \( n > 1 \). \( MAP_n^1 B \) is a backdoor for \( MAP_n^1 \).

To prove Theorem 1, one has to examine the effects of UP in the formulas \( MAP_n^1 \) quite closely [23]. The proof goes as follows. First, note that, in our encoding, any pair of move actions is incompatible. So if one move action is set to 1 at a time step, then all other move actions at that step are forced out by UP over the mutex clauses. Now, think about the backdoor variables in a backward fashion, assigning values to them starting at the last time step. In that step, the goal clauses form \( n \) constraints requiring to either visit a location \( L_1^1 \), or to have visited it earlier already (i.e., to achieve it via a NOOP). When assigning values to all \( MAP_n^1 B \) variables at that time, at least \( n - 1 \) goal constraints will be transported to the time step below. Iterating the argument, one gets at least 1 goal constraint at time 2. Taking account of several case distinctions, e.g. about the value assigned to \( NOOP-L_0^0(1) \), one can show that, after UP, \( n - 2 \) of the \( move-L_0^0-L_1^1(t) \) variables, \( i \neq 1 \), are set to 1 in non-adjacent time steps \( t \). With case distinctions about at exactly what non-adjacent \( t \) the move variables are set to 1, one can show that UP also enforces commitments to accommodate the remaining 2 \( move-L_0^0-L_1^1 \) actions – for which there is not enough room left.

We conjecture that the backdoor identified in Theorem 1 is also a minimum size (i.e., an optimal) backdoor; for \( n \leq 4 \) we verified this empirically. Note that the total number of variables in the CNF is also a square function in \( n \), so the backdoor is a linear-size variable subset. We proved that the backdoor is minimal, i.e., does not contain redundant variables.

**Theorem 2 (MAP bottom case, backdoors minimality).** Let \( n > 1 \). Let \( B' \) be a subset of \( MAP_n^1 B \) obtained by removing one variable. Then the number of UP-consistent assignments to the variables in \( B' \) is always greater than 0, and at least \( (n-3)! \) for \( n \geq 3 \).

The proof of this theorem is a matter of figuring out how wrong things can go when a variable is missing in the proof of Theorem 1.

Using the convention that \( L_0^0 \) stands for \( L^0 \), the backdoors we identify for the top case, called \( MAP_n^{2n-3} \), have the form:

\[
MAP_n^{2n-3}B := \{move-L_1^1(2^{i-2}-L_1^i-1) \mid 1 \leq i \leq \lceil \log_2 n \rceil \}
\]

Compare Figure 3 (d). Obviously, the size of \( MAP_n^1 B \) is \( \lceil \log_2 n \rceil \).

**Theorem 3 (MAP top case, backdoors).** Let \( n > 1 \). \( MAP_n^{2n-3} B \) is a backdoor for \( MAP_n^{2n-3} \).

We again conjecture that this is also a minimum size, optimal, backdoor. For \( n \leq 8 \) we verified this empirically. We can show that the backdoor is minimal.

**Theorem 4 (MAP top case, backdoors minimality).** Let \( n > 1 \). Let \( B' \) be a subset of \( MAP_n^{2n-3} B \) obtained by removing one variable. Then there is exactly one UP-consistent assignment to the variables in \( B' \).
The $\Theta(\log_2 n)$ backdoor size proved here for $\text{AsymRatio} = \frac{2n-3}{2n-1}$, compared to the $\Theta(n^2)$ backdoor from Theorem 1 for $\text{AsymRatio} = \frac{1}{2n-1}$, nicely reflects our empirical findings. We consider it particularly interesting that the $MAP_n^{2n-3}$ formulas have logarithmic backdoors. This shows, on the one hand, that these formulas are (potentially) easy for Davis Putnam procedures, having polynomial-size proofs. On the other hand, the formulas are non-trivial, in two important respects. First, they do have non-constant backdoors and are not just solved by unit propagation. Second, finding the logarithmic backdoors involves, at least, a non-trivial branching heuristic: the worst-case DPLL proofs for $MAP_n^{2n-3}$ are still exponential in $n$.

The $MAP_n^{2n-3}$ formulas being interesting in that way, it is instructive to have a closer look at how the logarithmic backdoors arise. The proof of Theorem 3 uses the following two properties of UP, in $MAP_n^{2n-3}$:

1. If one sets a variable $move-L^{i-1}_1-L^i_1(i)$ to 1, then at all time steps $j < i$ a move variable is set to 1 by UP.
2. If one sets a variable $move-L^{i-1}_1-L^i_1(i)$ to 0, then at all time steps $j > i$ a move variable is set to 1 by UP.

Both properties are caused by the “tightness” of branch 1, i.e., by UP over the precondition clauses of the actions moving along that branch, in combination with the goal to visit the outmost location. Other than what one may think at first sight, the two properties by themselves are not enough to determine the log-sized backdoor. The properties just form the foundation of a subtle interplay between the different settings of the backdoor variables, exploiting exponentially growing UP implication chains on branch 1. The interplay is best explained with an example. For $n = 8$, the backdoor is $\{move-L^0_0-L^1_1(1), move-L^2_1-L^3_1(3), move-L^6_1-L^7_1(7)\}$. Figure 4 contains an illustration.

Consider the first (lowest) variable in the backdoor, $move-L^0_0-L^1_1(1)$. If one sets this to 0, then property (2) applies: only 13 of the 14 available steps are
left to move towards the goal location $L_{13}^1$; UP recognizes this, and forces moves towards $L_{13}^1$ at all steps $2 \leq t \leq 14$. Since $t = 1$ is the only remaining time step not occupied by a move action, UP over the $L_3$ goal clause sets move-$L_0^0$-$L_1^1(1)$ to 1, yielding a contradiction to the precondition clause of the move set to 1 at time 2. So move-$L_0^0$-$L_1^1(1)$ must be set to 1.

Consider the second variable in the backdoor, move-$L_2^1$-$L_3^1(3)$. Say one sets this to 0. By property (2) this forces moves at all steps $4 \leq t \leq 14$. So the goal for $L_3^1$ must be achieved by an action at step 3. But then the move forced earlier at 4 becomes impossible. It follows that we must assign move-$L_2^1$-$L_3^1(3)$ to 1. With property (1), this implies that, by UP, all time steps below 3 get occupied with move actions. (Precisely, in our case here, move-$L_1^1$-$L_2^1(2)$ is also set to 1.)

Consider the third variable in the backdoor, move-$L_6^1$-$L_7^1(7)$. If we set this to 0, then by property (2) moves are forced in by UP at the time steps $8 \leq t \leq 14$. So, to achieve the $L_7^1$ goal at step 7, we have to take three steps to move back from $L_7^1$ to $L_0^0$: steps 4, 5, and 6. A move to $L_2^1$ is forced in at step 7, in contradiction to the move at 8 forced in earlier. Finally, if we assign move-$L_6^1$-$L_7^1(7)$ to 1, then by property (1) moves are forced in by UP at all steps below 7. We need seven steps to move back from $L_7^1$ to $L_0^0$, and an eighth step to get to $L_1^1$. But we have only the 7 steps 8, . . . , 14 available.

The key to the logarithmic backdoor size is that, to achieve the $L_1^1$ goal, we have to move back from $L_1^1$ locations we committed to earlier (as indicated in bold face above for $t = 3$ and $t = 7$). We committed to move to $L_1^1$, and the UP propagations force us to move back, thereby occupying $2 \times t$ steps in the encoding. This yields the possibility to double the value of $t$ between variables.

Proving Theorem 4 is a matter of figuring out what can go wrong in the proof to Theorem 3, after removing one variable [23]. Note that, with the above, the DPLL proof for $MAP_{2n-3}^2$ actually degenerates to a line, and has only $\lceil \log_2 n \rceil$ (non-failed) nodes. Besides small backdoors, such degenerated proof trees are probably also typical in structured examples.\[13\]

It would be interesting to determine what the optimal backdoors are in general, i.e. in $MAP_k^n$, particularly at what point the backdoors become logarithmic. Such an investigation turns out to be extremely difficult. For interesting combinations of $n$ and $k$ it is practically impossible to find the optimal backdoors empirically, and so get a start into the theoretical investigation. We developed an enumeration program that exploits symmetries in the planning task to cut down on the number of variable sets to be enumerated. Even with that, the enumeration didn’t scale up far enough. We leave this topic for future work.

**SBW.** This is a block-stacking domain ($n$ blocks), with restrictions on what blocks can be stacked onto what other blocks. These are initially all located

\[13\] In fact, we proved in the meantime that the size of DPLL search trees for $MAP_n^k$ is exponentially lower-bounded in $n$. (The proof goes by a reduction to the Pigeon Hole problem, and is not yet available in the TR.) This shows a doubly exponential complexity gap between DPLL proofs in the bottom and top cases.
side-by-side on a table $t_1$. The goal is to bring all blocks onto another table $t_2$, that has only space for a single block; so the $n$ blocks must be arranged in a single stack on top of $t_2$. The parameter $k$, $0 \leq k \leq n$, defines the amount of stacking restrictions. There are $k$ “bad” blocks $b_1, \ldots, b_k$ and $n - k$ “good” blocks $g_1, \ldots, g_{n-k}$. For $1 < i \leq k$, $b_i$ can only be stacked onto $b_{i-1}$; $b_1$ can be stacked onto $t_2$ and any $g_i$. The $g_i$ can be stacked onto each other, and onto $t_2$.

Independently of $k$, the optimal plan length is $n$: a single move action stacks one block onto another block or a table. $AsymRatio$ is $\frac{1}{n}$ if $k = 0$, and $\frac{k}{n}$ otherwise. Our CNF formulas encode $n - 1$ action steps. In the bottom case, $k = 0$, we prove the existence of backdoors of size $\Theta(n^3)$. In the top case, $k = n - 2$, there are $O(\log_2 n)$ backdoors.

**SPH.** Finally, we constructed a non-Planning example that also exhibits similar asymmetric structure and backdoor size behavior. We modified the well-known Pigeon Hole problem. In our $SPH_k^n$ formulas, like in the classical Pigeon Hole problem, the task is to assign $n + 1$ pigeons to $n$ holes. The difference lies in that there is now one “bad” pigeon that requires $k$ holes, and $k - 1$ “good” pigeons that can share a hole with the bad pigeon. The remaining $n - k + 1$ pigeons are normal, i.e., need exactly one hole each. The range of $k$ is between 1 and $n - 1$. Independently of $k$, $n + 1$ holes are needed overall. Apart from identifying minimal backdoors for all $k$ and $n$, for $k = n - 1$ we identify an $O(n)$ DPLL proof, which implies an exponential complexity gap to $k = 1$ [24].

## 5 Conclusions

Current DPLL-based SAT solvers are very efficient in “structured” formulas encoding real-world applications from Planning and Verification. We considered the effect of structure on the hardness of Planning formulas. Our findings reveal a mechanism – strong asymmetries in subgoal structure, a semantic notion – which can give rise to very small DPLL proofs. Most interestingly, with high subgoal asymmetry, we identified classes of planning formulas with logarithmic size backdoors and DPLL proofs.

Our results promote the understanding of what is relevant for solver performance in practice. Such an understanding is, we think, important in itself. From a more practical point of view, it may inspire the development of new search heuristics. As a simple example, our analysis of minimal backdoors suggests to use different branching heuristics depending on the value of $AsymRatio$ (which can be approximated using various techniques from the literature [9–11]): with a high $AsymRatio$, concentrate on the actions supporting the most difficult subgoal; with a low $AsymRatio$, do not concentrate on any particular sub-goal and distribute the branching more uniformly.

Our results are, so far, mainly for planning domains. We are currently extending our work to consider other constraint reasoning applications. In general, we hope that our approach will lead to the investigation of other forms of problem structure that can be identified empirically, and be captured in synthetic domains and analyzed rigorously.
References