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

Jörg Hoffmann
Max Planck Institute for CS
Saarbrücken, Germany

Carla Gomes
Cornell University
Ithaca, NY, USA

Bart Selman
Cornell University
Ithaca, NY, USA

Abstract

In AI Planning, as well as Verification, a successful method is to compile the application 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 Planning context, we identify a form of *problem structure* concerned with the symmetrical or asymmetrical nature of the cost of achieving the individual planning goals. We quantify this sort of structure with a simple numeric parameter called $AsymRatio$, ranging between 0 and 1. We show empirically that $AsymRatio$ correlates strongly with SAT solver performance in a broad range of Planning benchmarks, including the domains used in the 3rd International Planning Competition. We then examine carefully crafted *synthetic planning domains* that allow to control the amount of structure, and that are clean enough for a rigorous analysis of the combinatorial search space. The domains are parameterized by size n, and by a structure parameter k, so that $AsymRatio$ is asymptotic to k/n. The CNFs we examine are unsatisfiable, encoding one planning step less than the length of the optimal plan. We prove upper and lower bounds on the size of the best possible DPLL refutations, under different settings of k, as a function of n. We also identify the best possible sets of branching variables. With *minimum* $AsymRatio$, we prove exponential lower bounds, and identify minimal branching sets of size linear in the number of variables. With *maximum* $AsymRatio$, we identify *logarithmic* DPLL refutations (and branching sets), showing a doubly exponential gap between the two structural extreme cases. This provides a concrete insight into the practical efficiency of modern SAT solvers.

Introduction

There has been a long interest in a better understanding of what makes combinatorial problems from SAT and CSP hard or easy. The most successful work in this area involves random instance distributions with phase transition characterizations (e.g., (Cheeseman, Kanefsky, & Taylor 1991; Hogg, Huberman, & Williams 1996)). 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. On the other hand, many unsatisfiable formulas from Verification and AI Planning contain well over 100,000 variables and can be proved unsatisfiable within a few minutes (e.g., with Chaff (Moskewicz *et al.* 2001)). 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 Planning. We view this as an entry point to similar studies in other areas.

We focus on showing infeasibility. Precisely, we consider the difficulty of *proving optimality of plans*. SAT-based planning (Kautz & Selman 1999) 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. Our focus is on the last unsuccessful iteration. 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.

Our overall approach can be described as follows. (A) Formulate an intuition about what makes search perform well or bad in practical examples. (B) Design some numeric measure of that sort of problem structure. Show empirically that the measure correlates with search performance in the relevant examples. (C) Design synthetic domains that capture the problem structure in a clean form. Analyze the behavior of search, within the synthetic domains, in detail.

Step (C) is a *case study* aimed at obtaining a deeper understanding of the structural phenomenon. There are many pitfalls in this "3-step strategy". Numeric measures of structure that stably correlate with search performance are a rare find. Importantly, we didn't execute steps (A), (B), and (C) as a sequence. The steps were heavily intertwined, forming a process of increasingly accurate intuitions and results in a trial-and-error fashion over a long period of time.

Note that our approach is very different from identifying tractable classes. Step (C) may yield results on polynomial best-case or worst-case behavior, but these hold only for the synthetic domains looked at. Step (B) is satisfied with *empirical* correlations between structure and performance – with the advantage of not being bound to syntactically identifiable tractable classes, which are typically too restrictive for practical examples.

Our analysis method in step (C) is to prove upper and lower bounds on the size of the best-case DPLL (Davis, Logemann, & Loveland 1962) proof trees, i.e., on the number of search nodes. We also investigate the best possible sets of branching variables. Such variable sets were recently coined "backdoors" (Williams, Gomes, & Selman 2003). 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.*[1] That is, a smallest possible backdoor encapsulates the best possible branching variables for DPLL, a question of huge practical interest. Identifying backdoors is also a technical device: we obtain our upper bounds as a side effect of the proofs of backdoor properties. 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.

Goal Asymmetry

Observing that, in many benchmarks, the individual goal facts correspond quite naturally to individual "sub-problems" of the task, our intuitions are these. (1) Proving plan optimality is hard if the optimal plan length (the number of steps needed to solve the task) arises from complex interactions between many sub-problems. (2) Proving plan optimality is easy if the optimal plan length arises mostly from a *single* sub-problem. We formalize both intuitions using a view based on sub-problem "cost", offering the possibility to *interpolate* between (1) and (2). We distinguish a *symmetrical case* – where the individual sub-problems are all (symmetrically) "cheap" – and an *asymmetrical case* – where a single sub-problem (asymmetrically) "dominates the overall cost". The asymmetrical case obviously corresponds to intuition (2). The symmetrical case corresponds to intuition (1) because, if each single sub-problem is cheap, but their conjunction is costly, then that cost must be the result of some sort of "competition for a resource" – an interaction between the sub-problems. One can interpolate between the symmetrical and asymmetrical cases by measuring to what extent any single sub-problem "dominates the overall cost". To turn these intuitions into a formal definition, it remains to define what precisely "cost" is, and what "dominating the overall cost" means. Various definitions are thinkable. Our theoretical work (the synthetic domains and their analysis) is relevant for any definition. In our empirical work, we chose to instantiate "cost" with the (optimal) number of steps needed, and "dominating the overall cost" with a simple maximization and normalization operation: we obtain our parameter $AsymRatio$ by selecting the most costly goal fact, and dividing that cost by the cost of achieving the conjunction of *all* goals. $AsymRatio$ ranges between 0 and 1. With the above intuitions, $AsymRatio$ should be thought of as *a high-level measure of the degree of sub-problem interactions*. It is an important open question whether more low-level (syntactic, ideally) measures can be found.

We constructed two synthetic Planning domains: a simple transportation-kind of domain, and a stacking domain. Each is characterized by a size parameter, n, and by a structure parameter, k. In the "transportation" domain, for example, one moves along the edges of a graph of a certain shape, and the goal is to visit some graph nodes. *What* nodes must be visited depends on k. In both domains, $AsymRatio$ is asymptotic to $\frac{k}{n}$. For increasing n, $AsymRatio$ converges to 0 for the lowest setting of k, while it converges to 1 for the highest setting of k. We also used our intuition about problem structure to create a *non-Planning* example that shows similar behavior, namely a structured version of the Pigeon Hole problem, where k controls how many holes one particular "bad" pigeon needs.

Investigating the effect that the problem structure modelled in our synthetic domains has on the corresponding DPLL proofs, we found dramatic differences in DPLL proof and backdoor size. In the symmetrical case, we could prove exponential (in n) lower bounds on the size of DPLL trees. The backdoor sets in the symmetrical case are linear in the total number of variables. With increasing k, the backdoors become smaller. In the two planning domains, with maximum k – asymmetrical case – the backdoors are of *logarithmic size*, $O(logn)$. UP immediately yields a contradiction for one of the settings of each of the backdoor variables, so the DPLL trees degenerate to lines, and the number of search nodes is also $O(logn)$, showing a doubly exponential gap to the symmetrical case.

To confirm that our quantification of problem structure, $AsymRatio$, 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 (Long & Fox 2003) (*IPC-3*). This is a recent 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.[2] We also ran the experiments in Blocksworld and Logistics, two of the most classical Planning benchmarks. We plotted the performance of a state-of-the-art SAT solver, namely, ZChaff (Moskewicz *et al.* 2001), as a function of $AsymRatio$. We were surprised ourselves by how clearly the results came out. In most domains, a larger $AsymRatio$ consistently results in planning CNFs that are a lot easier to solve. $AsymRatio$ thus provides a useful indicator of typical problem hardness in Planning.[3] This is of course just a first example of such an indicator; presumably, others exist.

[1] 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.

[2] For the domains used in the 4th International Planning Competition there are no random generators.

[3] While $AsymRatio$ can not be computed efficiently, there exists a variety of techniques to approximate plan length (Blum & Furst 1997; Bonet & Geffner 2001; Edelkamp 2001; Helmert 2004). These can be used to approximate $AsymRatio$, and predict SAT solver performance. Exploring this is an open topic.

Related Work

The logarithmic backdoors in our synthetic examples nicely reflect the recent (empirical) finding that many Planning CNFs contain exorbitantly small backdoors in the order of 10 out of 10000 variables (Williams, Gomes, & Selman 2003). In difference to these results, we also explain what these backdoor variables *are* (what they correspond to in the original planning task), and how their interplay works.

Our analysis of synthetic examples is, in spirit, similar to the work in proof complexity (e.g., (Cook & Reckhow 1979; Haken 1985; Buss & Pitassi 1997)) where formula families, such as Pigeon Hole problems, have been the key to a better understanding of the length of resolution proofs. Generally speaking, the main difference is that, in proof complexity, one fixes an example and investigates the behavior of different proof calculi. By contrast, we consider the single proof calculus DPLL, and modify the *examples*. Major technical differences arise also due to the kinds of formulas considered – formulas from Planning vs. any kind of synthetic formula provoking a certain behavior – and, importantly, the central goal of the research. Proof complexity is mostly about lower bounds (separating the power of proof systems). But, to understand real-world structure, and explain the good performance of SAT solvers, interesting formula families with *small* DPLL trees are more revealing.

There is some work on problem structure in the ICAPS community. Hoffmann (2005) investigates topological properties of certain wide-spread heuristic functions. Howe and Dahlman (2002) analyze planner performance from a perspective of syntactic changes and computational environments. Streeter and Smith (2005) provide an analysis of search space surface in Job Shop scheduling. Obviously, all these works are quite different from ours.

There is a large body of work on structure in the constraint reasoning community, see for example (Frank, Cheeseman, & Stutz 1997; Slaney & Walsh 2001; Nudelman *et al.* 2004; Hulubei & Sullivan 2005). However, as far as we are aware, all these works differ considerably from ours. In particular, *all* works we are aware of define "structure" on the level of the CNF formula/the CSP problem instance, rather than, as we do, on the level of the modelled application. Empirical work on structure is mostly based on random problem distributions, and theoretical analysis is mostly done in the context of identifying tractable classes. Still, one structural concept must be discussed in more detail: *cutsets* (e.g. (Dechter 1990; 2003)). 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. The main *difference* between a backdoor and a cutset is 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. In that sense, a cutset is a backdoor that can be detected statically, and that can thus be directly exploited in a search algorithm. We will see that, in the particular formula families we consider here, there are no small 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" – which is precisely what tends to happen in practice, where there is usually no easy-to-see indication of how hard it will be to solve a (Planning, e.g.) formula.

In the next section, we provide some notation. Then a section presents our empirical work on $AsymRatio$ as a measure of problem hardness in Planning benchmarks. Thereafter, a section describes our synthetic domains and our analysis of DPLL proofs, and another section concludes.

Preliminaries

We use the STRIPS formalism, using the terminology and notation *initial state* I, *goal* G, and actions a with $pre(a)$, $add(a)$, $del(a)$, with the standard syntax and semantics. By *planning tasks* we mean instances of STRIPS.

CNF formulas are sets of clauses, where each clause is a set of literals. For a CNF formula ϕ 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) ϕ, 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. The *size* of a DPLL tree is the number of search nodes in it. The size of a resolution refutation is the number of clauses in it.

We use two different methods to encode planning tasks as CNF formulas. In our empirical work, we use the original *Graphplan-based* encoding from Blackbox (Kautz & Selman 1999), which we assume the reader is familiar with. In our formal analysis, to keep the formulas feasibly simple, we use a somewhat simplified version of that encoding. We use variables only for the actions, taking the form $a(t)$, $1 \leq t \leq b$, where b is the bound on the plan length. As in Blackbox, there is an artificial *NOOP* action for each fact p, whose only precondition is p, and whose only (add) effect is p. The NOOPs are treated just like normal actions in the encoding. 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 $p \in pre(a)$, at least one action a' is present at $t-1$ with $p \in add(a')$. For each action a present at a time t and for each $p \in pre(a)$, there is a *precondition* clause of the form $\{\neg a(t), a_1(t-1), \ldots, a_l(t-1)\}$, where $a_1, \ldots, a_l$ are all actions present at $t-1$ with $p \in add(a_i)$. For each goal fact $g \in G$, there is a *goal* clause $\{a_1(b), \ldots, a_l(b)\}$, where $a_1, \ldots, a_l$ are all actions present at b that have $g \in add(a_i)$. Finally, for each *incompatible* pair a and a' of actions present at a time t, there is a *mutex* clause $\{\neg a(t), \neg a'(t)\}$. 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 $p \in del(a')$ (or vice versa) – i.e., our synthetic CNFs encode sequential planning.[4]

[4] In our synthetic domains, there is no parallelism anyway, and

Goal Asymmetry in Planning Benchmarks

As said, we quantify goal asymmetry as follows.

Definition 1 *Let P be a planning task with goal G. For a conjunction C of facts, let $cost(C)$ be the length of a shortest plan achieving C. The* asymmetry ratio *of P is:*

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

Note that $cost(\bigwedge_{g \in G} g)$, 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 "sub"-goals. 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 subproblems of the task.[5]

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 $\phi(P, m-1)$ unsatisfiable is strongly correlated with $AsymRatio(P)$.*

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 $AsymRatio$ 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 $AsymRatio$ works surprisingly well in a broad range of domains.

To test our hypothesis, as said, we ran large experiments in all STRIPS domains used in the 3rd International Planning Competition (Long & Fox 2003) (IPC-3), plus Blocksworld and Logistics. The IPC-3 domains are Depots, Driverlog, Freecell, Rovers, Satellite, and Zenotravel. Depots is a mixture between Blocksworld and Logistics, where 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, 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. Importantly, *within each of all these domains, deciding bounded plan existence — the problem encoded by our CNFs — is* **NP***-hard* (Helmert 2003). So our experiments are on challenging, if not real-world realistic, problems.

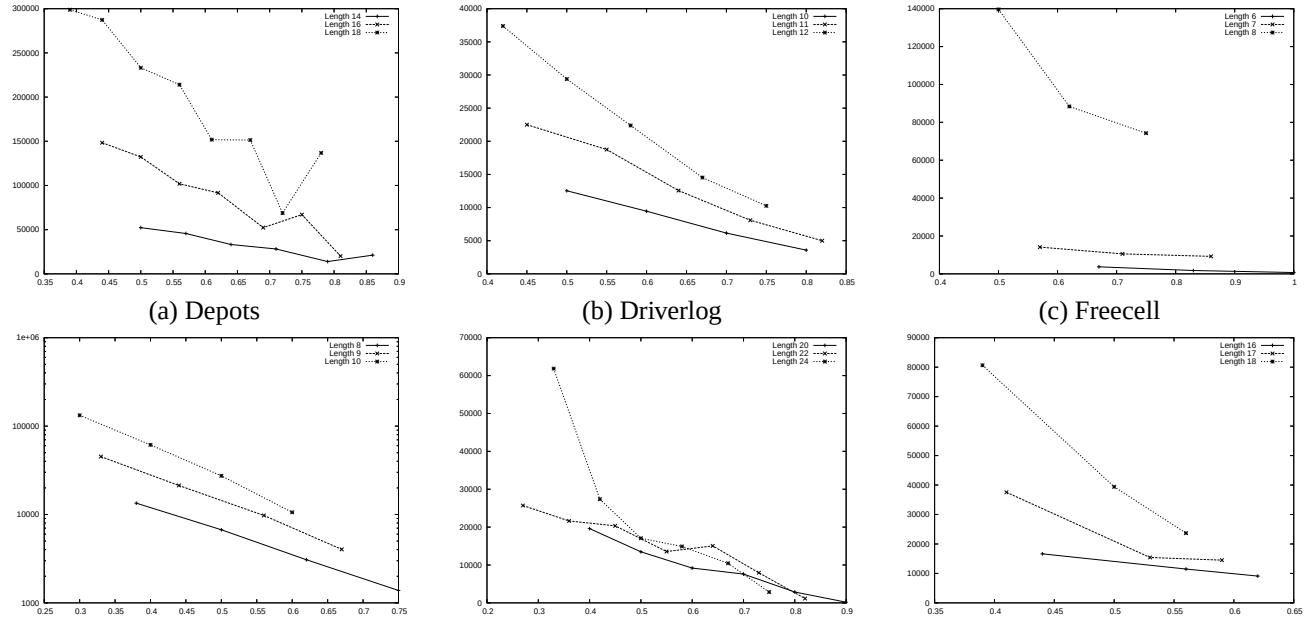
To obtain a reliable picture of how a complex DPLL-based SAT solver (ZChaff) typically behaves in CNF formulas generated from a domain, within each domain we generated and examined tens of thousands of instances – precisely, 50000. We chose the instance size parameters by testing the original IPC-3 instances, and selecting the largest one for which we could compute $AsymRatio$ reasonably fast.[6] E.g. in Driverlog we selected the instance indexed 10 out of 20, and, accordingly, generated random instances with 6 road junctions, 2 drivers, 6 packages, and 3 trucks. In Blocksworld, we generated random tasks with 9 blocks, in Logistics we generated random tasks with 1 airplane, 8 cities with 2 locations each, and 8 packages. According to the setup in Hypothesis 1 (we also use the notations), within each domain we separated the 50000 instances into sub-sets $\mathcal{P}_m$ with identical optimal plan length m. For each P in a set $\mathcal{P}_m$, we computed $AsymRatio(P)$, and ran ZChaff(Moskewicz *et al.* 2001) on the formula $\phi(P, m-1)$, measuring the search tree size (nr. of backtracks). We plotted the latter against $AsymRatio$ by dividing each $\mathcal{P}_m$ into 100 bins, with $AsymRatio(P) \in [0, 0.01), \ldots, [0.99, 1]$; we took the mean value out of each bin, avoiding noise by skipping bins with less than 100 elements. We were surprised ourselves by how clearly the results came out; see Figure 1. (Plots for medium values are almost identical.)

The plots are clearly supportive of Hypothesis 1. Consider for example Figure 1 (d), the Rovers domain. The three curves correspond to the classes of instances with optimal plan length 8, 9, and 10, respectively (the entire distribution of optimal plan length is $5 \ldots 20$, and 62% of the 50000 instances lie in the shown classes). In each of the classes, the search tree size decreases *exponentially* over a linear increase in $AsymRatio$ – note the logarithmic scale. From the relative positions of the different curves, one can also nicely see the influence of optimal plan length/formula size

the mutex clauses have an effect only on the power of UP. We remark that Graphplan detects the linear nature of the domains, so, there, both encodings have the same action mutexes.

[5] A more stable approach would be to identify a hierarchy of layers of "landmarks" (Hoffmann, Porteous, & Sebastia 2004): set $G_0 := G$; iteratively, set $G_{i+1} := \bigcup_{g \in G_i} \bigcap_{a: g \in add(a)} pre(a)$, until G_{i+1} is contained in the previous layers. One could then select the largest landmark layer G_i and define $AsymRatio$ based on that, for example as $max_{g \in G_i} cost(g) + i$ divided by $cost(\bigwedge_{g \in G} g)$. This approach could not be fooled by replacing G with a single goal. It seems an overkill since, as said, typically the facts in G already correspond naturally to different sub-problems. Exploring this issue in more depth is an open topic.

[6] That computation was done by calls to Blackbox.



(a) Depots (b) Driverlog (c) Freecell

(d) Rovers (e) Blocksworld (f) Logistics

Figure 1: Mean search tree size of ZChaff, plotted against $AsymRatio$ (log-plotted in (d)), in CNFs encoding instances from the IPC-3 domains (except Satellite and Zenotravel, see text), Blocksworld, and Logistics. Curves for different subsets $\mathcal{P}_m$ of 50000 random instances in each domain: the subsets corresponding to the 3 most frequently occurring optimal plan lengths m.

– the longer the optimal plan, the larger the search tree. In Blocksworld, the decrease of tree size over $AsymRatio$ is more noisy, and roughly linear except in the largest CNFs (encoding 24 steps), where it is stronger. In Depots, Driverlog, and Logistics, the decrease is roughly linear in all classes. In Freecell, the correlation is vague in the smaller CNFs, but very clear in the largest ones. Note that there are less data points (different $AsymRatio$ values) in Freecell and Logistics than in the other domains. This is a domain property: due to the particular semantics, more or less of the m possible $AsymRatio$ values for a set $\mathcal{P}_m$ actually appear in practice. Indeed, Satellite and Zenotravel are not shown in Figure 1 because there is hardly *any* difference in $AsymRatio$: the number of steps needed to achieve the individual goal facts is nearly constant. In fact, in Satellite, within each class $\mathcal{P}_m$ all instances have *the same* $AsymRatio$ value. In Zenotravel, there are up to three different values of $AsymRatio$ within each $\mathcal{P}_m$, but one of these values accounts for almost all of the instances (99% and more). We conclude that the degree of $AsymRatio$ variance, and its correlation with performance, depends on the domain – just as one would expect of a measure of problem structure. In six of our eight domains, a – sometimes drastic – correlation can be observed.

Beside the search tree size of ZChaff, we also measured the (maximum) search tree depth, the size of the identified backdoors (the sets of variables branched upon), and the *ratio* between size and depth of the search tree.[7] The latter gives an indication of how "broad" or "thin" the shape of the search tree is – denoting depth with d, if the tree is full binary then the ratio is $(2^{d+1}-1)/d$; if the tree is degenerated to a line then the ratio is $(2d+1)/d$. For each of our domains (except Satellite and Zenotravel), and for each of all these parameters, we found an at least linear decrease over increasing $AsymRatio$. Figure 2 shows the size/depth ratio data for the Rovers domain, as an example. We find the results regarding size/depth ratio particularly interesting since they nicely reflect the wide-spread intuition that, as problem structure increases, UP can prune many branches early on and so makes the search tree grow thinner.

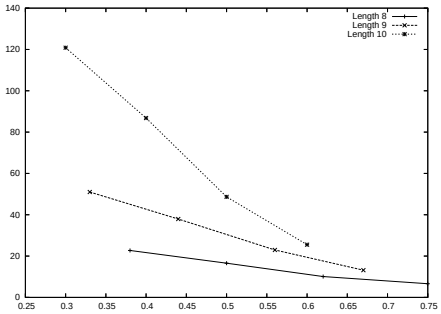


Figure 2: Size/depth ratio of ZChaff's search trees, plotted against $AsymRatio$, in Rovers.

We finally measured and compared the relative number of instances, over $AsymRatio$, in our eight domains as well as in a domain of purely random instances generated using Rintanen's (2004) "Model A" with 40 state variables. As already mentioned, in Satellite and Zenotravel the distribution of instances over $AsymRatio$ has all (almost all) its weight in a single point. In our other domains, most of the time the distributions are roughly Gaussian, with a peak of around 50%

[7] Note that backdoors play a different role in ZChaff than in standard DPLL, due to the effects of clause learning, which may, e.g., cause a choice variable to be switched by UP.

of the instances at an $AsymRatio$ of around 0.6, the other instances distributed with maximum distance of around 0.2 on both sides of the peak. More precisely, in the respectively most inhabitated $\mathcal{P}_m$ class, Depots has 32% of the instances at $AsymRatio$ 0.56, Driverlog has 40% at 0.55, Freecell has 51% at 0.83, Rovers has 67% at 0.5, Blocksworld has 31% at 0.6, and Logistics has 62% at 0.62. Now, by contrast, the distribution generated with Rintanen's method has a peak of 30% at $AsymRatio$ 0.3 – the (roughly Gaussian) distribution has its weight in *significantly lower* $AsymRatio$ *values.* We take this to confirm the intuition that a high $AsymRatio$ is not likely to occur in a purely random world – it is indeed something that is typical for *structured* instances only.[8]

Analyzing Goal Asymmetry in Synthetic Domains

We analyzed three classes of synthetic domains/CNF formulas, called MAP, SBW, and SPH. MAP is a simple transportation kind of domain, SBW is a block stacking domain. SPH is a structured version of the Pigeon Hole problem. Each of the domains/CNF classes is parameterized by size n and structure k. In the planning domains, we use the simplified Graphplan-based encoding described earlier, and consider CNFs that are one step short of a solution. We denote the CNFs with MAP_n^k, SBW_n^k, and SPH_n^k, respectively.

Due to space restrictions, we consider only MAP in detail, and we omit all proofs. Full details and proofs are available in a TR (Hoffmann, Gomes, & Selman 2006). As said, the aim of our analysis was to obtain a deeper understanding of the observed empirical correlation. We chose the MAP and SBW domains because they are related to Logistics and Blocksworld, two of the most classical Planning benchmarks. We chose SPH for its close relation to the formulas considered in proof complexity. The reader will notice that the synthetic domains are *very* simple. The reasons for this are threefold. First, we wanted to capture $AsymRatio$ in as clean a form as possible, without "noise". Second, even though the Planning tasks are quite simple, *the resulting CNF formulas are complicated* – e.g., much more complicated than the Pigeon Hole formulas often considered in proof complexity. Third, we identify *provably minimal* backdoors. To do so, one has to take account of every tiny detail of the effects of unit propagation. The respective proofs are quite involved for our simple domains already – for MAP, e.g., they occupy 9 pages in the TR, featuring myriads of interleaved case distinctions. To analyze more complicated domains, one probably has to sacrifice precision.

We emphasize once more that the design of our domains is, to a large extent, independent of the precise quantification we used to turn our intuitions about "the degree of subproblem interactions" into a number – $AsymRatio$, namely. The domains are relevant for any formal definition of the same sort of intuitive problem structure. The same is true for the analysis of the corresponding search spaces.

[8]The correlation of $AsymRatio$ with search tree size, depth, size/depth ratio and backdoor size is also present in the random instances, with a little more variance.

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 $\{L_1^1, \ldots, L_n^1\}$. For each increase of k by 2, the goal on the L_1-branch goes up by two steps, and the highest-indexed of the other goals is skipped. For $k = 2n-3$, we get the goal $\{L_1^{2n-3}, L_2^1\}$.[9] We refer to $k = 1$ as the *symmetrical case*, and to $k = 2n-3$ as the *asymmetrical case*, see Figure 3 (a) and (b), respectively.

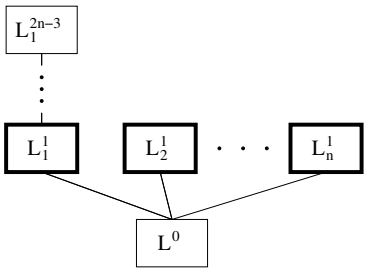
The length of a shortest plan is $2n-1$ independently of k; our CNFs encode $2n-2$ steps; $AsymRatio$ is $\frac{k}{2n-1}$. Figure 3 (c) and (d) illustrate that the setting of k has a quite drastic effect on backdoor size. We will detail this below. First, observe that the setting of k has only very little impact on the size and shape of the constraint graph (the undirected graph where nodes are variables and edges indicate common membership in at least one clause), illustrated in Figure 3 (e) and (f): 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. In general, between formulas MAP_n^k and $MAP_n^{k'}$, $k' > k$, there is *no difference* except that $k' - k$ goal clauses are skipped, and that the content of the goal clause for the L_1-branch changes. Precisely, the number of clauses in MAP_n^k is $3n^3 + 27n^2 - 73n + 39 - (k+1)/2$. The number of variables is $16n^2 - 33n + 14$, irrespectively of k. It even holds that, also irrespectively of k, for any constant b, the b-cutset size in MAP_n^k is a square function in n. This can be seen as follows. 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 to or from L^0, which are fully connected due to the mutex clauses. From a clique of size l, one has to remove $l - 1 - b$ nodes in order to get to an induced width of $1 \leq b \leq l - 1$. Since the mentioned cliques are disjoint, this shows the claim.

The hidden structure in our formulas does not affect the size of b-cutsets. It *does* affect the size of DPLL refutations, and backdoors. First, we proved that, in the symmetrical case, the DPLL trees are large.

Theorem 1 (MAP symmetrical case, DPLL LB) *Every DPLL refutation of MAP_n^1 must have size exponential in n.*

The proof of Theorem 1 proceeds by a "reduction" of MAP_n^1 to a variant of the Pigeon Hole problem. A reduction here is a function that *transforms a resolution refutation of MAP_n^1 into a resolution refutation of the Pigeon Hole.* Obviously, given a reduction function from formula class A

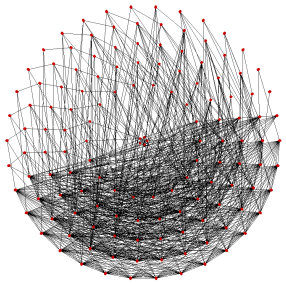
[9]For $k = 2n-1$, MAP_n^k contains an empty clause: no supporting action for the goal is present at the last time step.



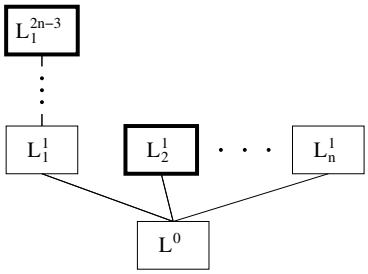
(a) goals symmetrical case

	1	2	3	4
1	NA–0			
2				
3		MV–2	MV–3 NV–3	MV–4 NV–4
4				
5		MV–2	MV–3 NV–3	MV–4 NV–4
6				

(c) backdoor symmetrical case



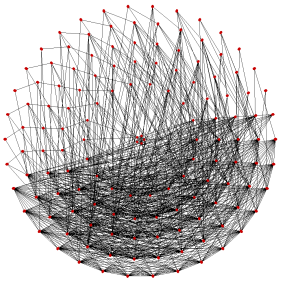
(e) constraint graph symmetrical case



(b) goals asymmetrical case

	1	2	3	4
1	MV–1			
2				
3	MV–23			
4				
5				
6				

(d) backdoor asymmetrical case



(f) constraint graph asymmetrical case

Figure 3: Goals, backdoors, and constraint graphs in MAP. In (a) and (b), goal locations are indicated in bold face, for the symmetrical case (a), and the asymmetrical case (b). In (c), (d), (e), and (f), $n = 4$. In (c) and (d), the horizontal axis indicates branches in the map, and the vertical axis indicates time steps; abbreviations: "NA-0" for $NOOP\text{-}at\text{-}L^0(1)$, "MV-i" for $move\text{-}L^0\text{-}L^1_i$, "NV-i" for $NOOP\text{-}visited\text{-}L^1_i$, and "MV-23" for $move\text{-}L^2_1\text{-}L^3_1$. 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.

into formula class B, a lower bound on the size of resolution refutations of B is also valid for A, modulo the maximal size increase induced by the reduction. We define a reduction function from MAP^1_n into the *onto functional Pigeon Hole problem*, $ofPHP_n$. This is the standard Pigeon Hole – where $n+1$ pigeons must be assigned to n holes – plus "onto" clauses saying that at least one pigeon is assigned to each hole, and "functional" clauses saying that every pigeon is assigned to at most one hole. Every resolution refutation of $ofPHP_n$ must have size $exp(\Omega(\frac{n}{(log(n+1))^2}))$ (Razborov 2004). Our reduction proceeds by first setting many variables in MAP^1_n to 0 or 1, and identifying other variables (renaming x and y to a new variable z).[10] We prove that such operations do not increase the size of a resolution refutation. The reduced formula is a "temporal" version of the onto Pigeon Hole problem; we call it $oTPHP_n$. It is similar to the standard (onto) Pigeon Hole problem except that now the "holes" are time steps, in analogy to Planning encodings. We prove that, from a resolution refutation of $oTPHP_n$, one can construct a resolution refutation of $ofPHP_n$ by replacing each resolution step with at most n^2+n new resolution steps. This proves the claim for general resolution, which suffices since DPLL corresponds to a restricted form of resolution (e.g., see (Beame, Kautz, & Sabharwal 2004)). The same is true for DPLL with clause learning, as done in the ZChaff solver we use in our experiments.

The proof of Theorem 1 by reduction to the Pigeon Hole problem makes intuitive sense: clearly, trying to visit the n goal locations in not enough time is a Pigeon Hole style situation. However, the proof does not tell us very much about what is actually going on inside a DPLL procedure run on MAP^1_n. To shed more light on this, we investigated the best choices of branching variables for such a procedure. We identified the following backdoor:

$$\begin{aligned} MAP^1_nB := &\{move\text{-}L^0\text{-}L^1_i(t) \mid t \in T, 2 \le i \le n\} \cup \\ &\{NOOP\text{-}visited\text{-}L^1_i(t) \mid t \in T, 3 \le i \le n\} \cup \\ &\{NOOP\text{-}at\text{-}L^0(1)\} \cup \\ &\{move\text{-}L^0\text{-}L^1_1(t) \mid t \in T \setminus \{2n-5, 2n-3\}\} \end{aligned}$$

Here, $T = \{3, 5, \ldots, 2n-3\}$. Compare Figure 3 (c).

Theorem 2 (MAP symmetrical case, BD) *MAP^1_nB is a backdoor for MAP^1_n.*

Obviously, the size of MAP^1_nB is $\Theta(n^2)$.[11] For the proof to Theorem 2, first note that, in the 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 – the time step is "occupied" (this is relevant also in the asymmetrical case below). Now, to see the basic proof argument, assume for the moment that MAP^1_nB contains *all* $move\text{-}L^0\text{-}L^1_i(t)$ and

[10] For example, we set all $NOOP\text{-}at$ variables to 0. Such a variable will never be set to 1 in an optimal plan; similar intuitions are behind all the made operations.

[11] Remember that the total number of variables is also $\Theta(n^2)$.

$NOOP$-$visited$-$L_i^1(t)$ variables, for $1 \leq i \leq n$ and $t \in T$. Assigning values to all these variables results, by UP, in a sort of goal regression. In the last time step of the encoding, $2n-2$, the goal clauses form n constraints requiring to either visit a location L_i^1, or to have visited it earlier already (i.e., to achieve it via a NOOP). Examining the interactions between $move$s and $NOOP$s at $t = 2n-3$, one sees that, if all these are set, then at least $n-1$ goal constraints will, by UP, be transported down to $t = 2n-4$. Iterating the argument over the $n-2$ time steps $t \in T$, one gets 2 goal constraints at $t = 2$: two nodes L_i^1 must be visited within the first two time steps. It is easy to see, then, that branching over $NOOP$-at-$L^0(1)$ yields an empty clause in either case. What makes identifying a *non-redundant* (minimal) backdoor difficult is that UP is, in a variety of subtleties, slightly more powerful than just performing the outlined "regression". MAP_n^1B contains hardly any variables for branch $i = 1$. So at $t = 2$ one gets only a single goal constraint, achieving which isn't a problem. We perform an intricate case distinction about the precise pattern of time steps that are occupied after the regression, taking account of, e.g., such subtleties as the possibility to achieve $visited$-L_1^1 by moving in from L_1^2 *above*. In the end, one can show that UP enforces commitments to accommodate also the 2 $move$-L^0-L_i^1 actions that weren't accommodated in the regression. For this, there is not enough room left.

We conjecture that the backdoor identified in Theorem 2 is also a minimum size (i.e., an optimal) backdoor; for $n \leq 4$ we verified this empirically, by enumerating all smaller variable sets.[12] As said, the backdoor is minimal.

Theorem 3 (MAP symmetrical case, BD minimality) *Let B' be a subset of MAP_n^1B obtained by removing one variable. The number of UP-consistent assignments to B' is always greater than 0, and at least $(n-3)!$ for $n \geq 3$.*

To prove this theorem, one figures out how wrong things can go when a variable is missing in the proof of Theorem 2.

Intuitively, the backdoor in the symmetrical case has square size because n branches are involved at $2n-2$ steps. One would expect that, in the asymmetrical case, a DPLL refutation involving only branch 1 could yield a backdoor of linear size in n. It turns out one can do much better. Using the convention that L_1^0 stands for L^0, the backdoors we identify have the form (compare Figure 3 (d)):

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

Theorem 4 (MAP asymmetrical case, BD) *$MAP_n^{2n-3}B$ is a backdoor for MAP_n^{2n-3}.*

The size of $MAP_n^{2n-3}B$ is $\lceil log_2 n \rceil$. We conjecture that this is optimal, which we verified empirically for $n \leq 8$.

Theorem 5 (MAP asymmetrical case, BD minimality) *Let B' be a subset of $MAP_n^{2n-3}B$ obtained by removing one variable. There is exactly one UP-consistent assignment to B'.*

[12] Enumerating variable sets in small enough examples was also our method to find the backdoors in the first place.

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 DPLL procedures. 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 requires, at least, a non-trivial branching heuristic – the *worst*-case DPLL refutations of MAP_n^{2n-3} are still exponential in n.

Let us have a closer look at how the logarithmic backdoors arise. The proof of Theorem 4 uses the following two properties of UP, in MAP_n^{2n-3}:

(1) If one sets a variable $move$-L_1^{i-1}-$L_1^i(i)$ 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_1^{i-1}-$L_1^i(i)$ 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. 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 $\{move\text{-}L^0\text{-}L_1^1(1), move\text{-}L_1^2\text{-}L_1^3(3), move\text{-}L_1^6\text{-}L_1^7(7)\}$. Figure 4 contains an illustration.

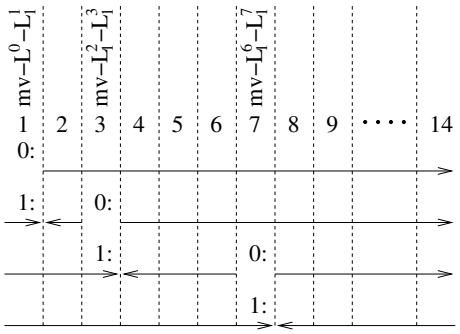


Figure 4: The workings of the optimal backdoor for MAP_8^{13}. Arrows indicate moves on the L_1-branch forced to 1 by UP. Direction $\rightarrow$ means towards L_1^{13}, $\leftarrow$ means towards L^0. When only a single open step is left, $move$-L^0-L_2^1 is forced to 1 at that step by UP, yielding an empty clause.

Consider the first (lowest) variable in the backdoor, $move$-L^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_1^{13}; UP recognizes this, and forces moves towards L_1^{13} 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_2^1 goal clause sets $move$-L^0-$L_2^1(1)$ to 1, yielding a contradiction to the precondition clause of the move set to 1 at time 2. So $move$-L^0-$L_1^1(1)$ must be set to 1.

Consider the second variable in the backdoor, $move$-L_1^2-$L_1^3(3)$. 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_2^1 must be achieved by an action at step 3. But we have committed to $move$-L^0-L_1^1 at step 1. This forces us to move back to L^0 at step 2 and to move to L_2^1 at step 3. But then the move forced in earlier at 4 becomes impossible. It follows that we must assign $move$-L_1^2-$L_1^3(3)$ 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-$L_1^2(2)$ is also set to 1.)

Consider the third variable in the backdoor, $move$-L_1^6-$L_1^7(7)$. 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_2^1 goal at step 7, we have to take **three steps to move back from L_1^3 to L^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_1^6-$L_1^7(7)$ 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_1^7 to L^0**, and an eighth step to get to L_2^1. But we have only the 7 steps $8, \ldots, 14$ available, so the goal for L_2^1 is unachievable.

The key to the logarithmic backdoor size is that, to achieve the L_2^1 goal, we have to move back from L_1^t locations we committed to earlier (as indicated in bold face above for $t = 3$ and $t = 7$). We committed to move to L_1^t, and the UP propagations force us to move back, thereby occupying $2{*}t$ steps in the encoding. This yields the possibility to double the value of t between variables.

Proving Theorem 5 is a matter of figuring out what can go wrong in the proof to Theorem 4, after removing one variable. Note that the DPLL tree for MAP_n^{2n-3} actually *degenerates to a line*: if one processes the $MAP_n^{2n-3}B$ variables from $t = 1$ upwards, then, for every variable, assigning 0 immediately yields an empty clause in UP.

Corollary 1 (MAP asymmetrical case, DPLL UB) *For MAP_n^{2n-3}, there is a DPLL refutation of size $2{*}\lceil log_2 n \rceil{+}1$.*

Besides small backdoors, (nearly) degenerated DPLL trees are also typical in structured examples, as the empirical results summarized in the previous section (specifically, Figure 2) show. Note that we have now shown a *doubly* exponential gap between the sizes of the best-case DPLL refutations in the symmetrical case and the asymmetrical case.

It would be interesting to determine what the optimal backdoors are in general, i.e. in MAP_n^k, 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.

To conclude our analysis, Figure 5 shows the behavior of ZChaff in MAP. As expected, we get exponential scaling for the symmetrical case $k = 1$, and polynomial scaling for the asymmetrical case $k = 2n - 3$. If we fix a value of n in Figure 5 (consider the intersections of the curves with a vertical line) we observe a strong correlation with $AsymRatio$, just like in the Planning benchmarks from Figure 1. This directly connects our formal analysis to our empirical results.

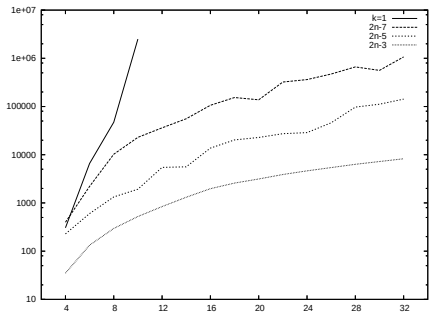


Figure 5: Search tree size of ZChaff in MAP, log-plotted against n, for different settings of k.

SBW

This is a block-stacking domain, with *stacking restrictions* on what blocks can be stacked onto what other blocks. The blocks are initially all located 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 restrictions. There are k "bad" blocks $b_1, \ldots, b_k$ and $n - k$ "good" blocks $g_1, \ldots, g_{n-k}$. Each b_i, $i > 1$, 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 any g_j, and onto t_2.

Independently of k, the optimal plan length is n: move actions stack one block onto another block or a table. $AsymRatio$ is $\frac{1}{n}$ if $k = 0$, and $\frac{k}{n}$ otherwise. In the symmetrical case, $k = 0$, we identify backdoors of size $\Theta(n^3)$ – linear in the total number of variables. In the asymmetrical case, $k = n - 2$, there are $O(logn)$ DPLL refutations and (minimal) backdoors.

SPH

Finally, we constructed a *non-Planning* example that also exhibits similar asymmetric structure and DPLL behavior. We modified the Pigeon Hole problem. In our SPH_n^k formulas, like in the standard Pigeon Hole problem, the task is to assign $n + 1$ pigeons to n holes. The difference is 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 combinations of k and n, for $k = n - 1$ we identify an $O(n)$ DPLL refutation. With results by Buss and Pitassi (1997), this implies an exponential complexity gap to $k = 1$.

Conclusion

Modern DPLL-based SAT solvers are very efficient in "structured" CNFs encoding applications from Planning and Verification. We defined a concrete notion of what "structure" is, in Planning, and we revealed empirically that this

structure indeed often governs performance, in practical examples. Our analytical results provide a detailed case study of how this phenomenon arises. In particular, we show that the phenomenon can make an (even doubly) exponential difference.

From a purely practical point of view, our research may inspire the development of novel search heuristics. The very different forms of the backdoors in the symmetrical and asymmetrical cases suggest to approximate AsymRatio, and choose a specialized branching heuristic depending on the outcome. Similarly, the use of symmetry detection and exploitation techniques (e.g. (Rintanen 2003; Sabharwal 2005)) seems particularly relevant with low AsymRatio, and could be done dependent on this. Most importantly maybe, with some more work, approximated AsymRatio could, at least within some fixed domain of interest, probably be made a successful runtime predictor, which is useful in various situations (like, making a priori decisions).

From a more principled point of view, our results promote the formal understanding of what is relevant for search performance in practical examples. Such an understanding is, we believe, of great importance in itself, and should be given more attention in the field. We do not claim that the presented results "solve" this issue in an exhaustive way. Quite differently, we hope and believe that our approach will inspire similar investigations of other forms of practical problem structure, and that this will make our understanding of what's going on inside search more mature.

Acknowledgements. We thank Ashish Sabharwal for advice on proving resolution lower bounds, Rina Dechter for discussions on the relation between backdoors and cutsets, Toby Walsh for discussions on the intuition behind $AsymRatio$, Jussi Rintanen for providing the code of his random instance generator, and Martin Gütlein for implementing the enumeration of CNF variable sets under exploitation of symmetries.

References

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *JAIR* 22:319–351.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *AI* 90(1-2):279–298.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AI* 129(1–2):5–33.

Buss, S., and Pitassi, T. 1997. Resolution and the weak pigeon-hole principle. In *Proc. CSL'97*, 149–156.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the *really* hard problems are. In *Proc. IJCAI'91*, 331–337.

Cook, S., and Reckhow, R. 1979. The relative efficiency of propositional proof systems. *JSL* 44:26–50.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Communications of the ACM* 5:394–397.

Dechter, R. 1990. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *AI* 41(3):273–312.

Dechter, R. 2003. *Constraint Processing*. Morgan-Kauffmann.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP'01*, 13–24.

Frank, J.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *JAIR* 7:249–281.

Haken, A. 1985. The intractability of resolution. *TCS* 39:297–308.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *AI* 143:219–262.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS'04*, 161–170.

Hoffmann, J.; Gomes, C.; and Selman, B. 2006. Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. Technical Report. Available at http://www.mpi-sb.mpg.de/∼hoffmann/tr-icaps06b.ps.gz.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Hogg, T.; Huberman, B.; and Williams, C. 1996. Phase Transitions and Complexity. *AI* 81.

Howe, A., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *JAIR* 17:1–33.

Hulubei, T., and Sullivan, B. 2005. Optimal refutations for constraint satisfaction problems. In *Proc. IJCAI05*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. IJCAI'99*, 318–325.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *JAIR* 20:1–59.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proc. DAC'01*, 530–535.

Nudelman, E.; Leyton-Brown, K.; Hoos, H.; Devkar, A.; and Shoham, Y. 2004. Understanding random SAT: beyond the clauses-to-variable ratio. In *Proc. CP-04*, 438–452.

Razborov, A. 2004. Resolution lower bounds for perfect matching principles. *J. Computer and Systems Sciences* 69(1):3–27.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In *Proc. ICAPS'03*, 32–41.

Rintanen, J. 2004. Phase transitions in classical planning: An experimental study. In *Proc. ICAPS'04*, 101–110.

Sabharwal, A. 2005. Symchaff: A structure-aware satisfiability solver. In *Proc. AAAI'05*, 467–474.

Slaney, J., and Walsh, T. 2001. Backbones in optimization and approximation. In *Proc. IJCAI01*.

Streeter, M., and Smith, S. 2005. Characterizing the distribution of low-makespan schedules in the job shop scheduling problem. In *Proc. ICAPS'05*, 61–70.

Williams, R.; Gomes, C.; and Selman, B. 2003. Backdoors to typical case complexity. In *Proc. IJCAI'03*.