Planning CNFs with Small DPLL Proofs

Jörg Hoffmann   Carla Gomes   Bart Selman

MPI–I–2005–2–001   February 2005

Authors' Addresses

Jörg Hoffmann
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
Saarbrücken, Germany
`hoffmann@mpi-sb.mpg.de`

Carla Gomes
Department of Computer Science
Cornell University
Ithaca, NY, USA
`gomes@cs.cornell.edu`

Bart Selman
Department of Computer Science
Cornell University
Ithaca, NY, USA
`selman@cs.cornell.edu`

Abstract

In order to better understand SAT-based planning and, more broadly, the good performance of current DPLL-based SAT solvers on real-world instances, we design three CNF formula families, two of which are derived from synthetic planning domains. The formulas are parameterized both in their size and in the amount of inherent structure. We investigate DPLL search tree size via the intermediate notion of backdoor size: a backdoor is a set of variables so that, once they are instantiated, the remaining problem is solved by polytime propagation. We show that, as the structure increases, the backdoors become smaller; in the two planning domains, at the extreme ends the backdoors are logarithmic, and thus the search trees are polynomial. The structure of the domains (with highly asymmetric goal sets) provides good intuitive insights into what may lead to small search trees in practice. We present empirical results supporting these intuitions.

Keywords

Propositional Satisfiability, Planning, Verification

1 Introduction

There has been a long interest in a better understanding of what makes combinatorial problems arising in AI hard or easy. The most successful work in this area involves random instance distributions with phase transition characterizations (e.g., [3]). Progress on more structured problems has been more difficult. Nevertheless, current satisfiability (SAT) solvers, such as CHAFF [14] are surprisingly effective on a wide range of structured problems arising in real-world domains, such as hardware and software verification and AI planning. The underlying SAT problems (CNF formulas) can contain hundreds of thousands of variables and millions of constraints. In this paper, we will focus on CNF formulas arising in planning applications. Our goal is to obtain a better understanding of why certain planning problems can be solved so effectively with DPLL based solvers.

An investigation of this nature is rather difficult because the structured instances from e.g. the Planning or SAT competitions are generally too complex to allow for a direct rigorous analysis. We introduce an alternative approach to the study of problem structure. We explicitly design three problem instance families, parameterized both in size and in structure (more details below). We then investigate the size of the best-case DPLL proofs as a function of the parameters. We view this approach as analogous to the work in proof complexity (e.g., [8, 2, 9, 13]) where instance families, such as Pigeon Hole problems [4], have been the key to a better understanding of the length of resolution, including DPLL-style, proofs. Indeed, one of our formula families is a structured version of the Pigeon Hole problem; the other two families come from synthetic planning domains. Note that, in proof complexity, formulas that lead to large proofs are most interesting, since they help us prove better lower-bounds. However, *to understand real-world problem structure, and explain the good performance of SAT solvers, interesting formula families with small DPLL search trees are most revealing.*

We investigate DPLL proof size via the intermediate vehicle of "backdoor" size [17]. A backdoor is a set of variables so that, once they are instantiated, the remaining problem is solved by polytime propagation; in particular, unit-propagation (UP) in DPLL-style proofs. Backdoors encapsulate the practically critical question of what variables to branch on.[1] The size of the backdoor set can be used to bound the size of the DPLL search tree. More specifically, in the worst case, the DPLL search tree is exponential in the backdoor size. As imme-

[1]Backdoors are closely related to concepts introduced earlier, in particular connectivity measures of the underlying constraint graph, such as treewidth and cutset. See for example [5, 15, 6]. For our purposes, the notion of backdoor is particular suitable in the study of DPLL-based solvers. For example, Horn formulas have backdoors of size zero but may still have large treewidth. On the other hand, a formula may be easy for e.g. directed resolution, but hard for DPLL. We are currently investigating the precise relationships between these related structural measures.

diate consequence, it follows that if we identify a logarithmic size backdoor, the corresponding DPLL tree will be of at most polynomial size.

We focus on showing infeasibility, i.e., all our CNFs are unsatisfiable. Our main motivation for this is that infeasibility generally provides a more robust characterization of the size of the combinatorial space. We consider the difficulty of proving the infeasibility of the structured pigeon hole problem, or that a plan with one step less than the shortest possible (*optimal*) plan does not exist.[2] In all cases, we rigorously derived an upper bound on the size of the smallest possible (*optimal*) backdoor, by determining a backdoor subset of variables. For small enough formulas, we verified empirically that there is no smaller backdoor. We conjecture that our upper bounds are tight in general. Note that establishing lower bounds on backdoor size is closely related to establishing lower bounds on proof size, which is a challenging task even in rather simple formulas such as the standard pigeon hole (c.f. above).

Our first synthetic domain (called *SPH*) is, as said, a structured version of the Pigeon Hole problem. Our other classes come from synthetic planning tasks: a logistics planning domain (*MAP*) and a stacking domain (*SBW*). The formulas are characterized (exactly) by a size parameter n, and by a structure parameter k. The structure parameter controls the amount of an intuitive "bottleneck" behavior or "asymmetry" in the underlying task. As the value of k increases, one part of the underlying combinatorial task becomes more and more difficult to achieve, while the other parts become relatively easier. More concretely, in our planning problems, as the value of k increases, one of the subgoals requires an ever longer action sequence to be achieved while other subgoals remain achievable in a few steps. This represents a large structural asymmetry in the underlying task, leading to provably small backdoors, and therefore enabling these (unsatisfiable) instances to be effectively solved by current SAT solvers. So, with increasing subgoal asymmetry, the backdoors become smaller; in the two planning domains, at the top end of the k scale, the backdoors set is of *logarithmic size*.[3] With more symmetrical subgoals (each requiring a similar number of actions), the backdoor sets are of polynomial size (in n) in all cases.

For example, in the MAP domain, an agent must visit a number of goal nodes in a graph-like world. We can vary the size of the backdoor set, thus the com-

[2]Note that, in planning with a plan length bound (the number of plan steps encoded by the CNF), the available plan steps are a type of resource. Empirically, the best predictor of performance on CNFs encodings of planning tasks is the difference between the length bound and the length of an optimal plan. The hardest cases arises, typically, when the length bound is just one step too short.

[3]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.

plexity of the task, by changing the definition of *what* goal nodes must be visited. For the smallest value of k (bottom case), the goal nodes are all distributed symmetrically, and the backdoor size grows proportionally to the optimal plan length multiplied with n (which equals the number of goal nodes).[4] At the top end of the k scale, the goal nodes are very asymmetric, one of them taking a lot of steps to be reached. We identify a backdoor set which grows logarithmic in n, the problem size. As an example, below we discuss in some detail one example CNF formula (called MAP_8^{13}) with 774 variables and 13,860 clauses, encoding 14 plan steps on a graph with 21 nodes, featuring a backdoor of only 3 variables. Branching on only these 3 variables suffices to prove unsatisfiability. The interplay between them exploits long UP implication chains after commitments on only three actions at specific time steps.

Our observations in our synthetic stacking domain (SBW) are very similar to the above. As an interesting aside, note that such asymmetry between subgoals is unlikely to occur in purely randomly generated problem instances, only in truly structured domains can one expect to observe such phenomena.

We also present empirical results suggesting that, at least in the planning context, our formulas do indeed capture behavior that is relevant in practice. As said, the intuition behind our structure parameters is that one part of the problem becomes more difficult than the rest. In the Planning context, one can define the "parts" of the problem as the single sub-goals to be achieved (e.g. in MAP, the goal nodes to be visited). The "difficulty" of a sub-goal can be taken to be the length of an optimal plan achieving just that sub-goal, the difficulty of the overall task is the length of an optimal plan achieving all sub-goals. Then the structure of the task can be quantified in terms of the ratio between maximum sub-goal difficulty and overall difficulty. We call this the *bottleneck ratio*.[5] In the synthetic planning domains, for the lowest value of k, the bottleneck ratio converges to 0 for increasing n. For the highest k value, the ratio converges to 1. This leads us to the hypothesis that DPLL-based solver performance in CNF encodings of planning tasks is correlated with bottleneck ratio, i.e., given tasks with the same optimal plan length, CNFs for tasks with higher bottleneck ratio are typically easier to reason about.

We ran large experiments on three commonly used Planning benchmark domains, plotting the performance of two state-of-the-art SAT solvers, namely, ZCHAFF [14] and SATZ [12] as a function of the bottleneck ratio. Our experiments show that in these domains, increasing the bottleneck parameter does indeed significantly decrease the runtime of the solvers. These results provide initial

[4]We will see that even the best-case DPLL search tree induced by the backdoor grows exponentially in n.

[5]we experiments. Our definition is specific to Planning. How to extend this to other domains is a topic for further research.

evidence that the parameterized structure of our synthetic problem instance families can provide insights into the structure of real-world problem domains and their practical computational difficulty.

The next section forms the core of the paper, formally describing our synthetic domains and the corresponding backdoor sets. After that, a section describes the experiments outlined above. The last section closes the paper with a few concluding remarks.

2 Synthetic CNFs and Backdoor Sets

As said, we consider three classes of synthetic, unsatisfiable, CNFs. SPH is a structured version of the Pigeon Hole problem, where there is a "bad" pigeon that needs several holes. MAP is a class of formulas encoding tasks in a simplified version of the Logistics planning domain, using an underlying road map graph of a certain shape. SBW is a class of formulas encoding tasks in a structured version of the Blocksworld planning domain, with restrictions on what blocks can be stacked onto what other blocks.

In each of the classes, formulas are characterized by exactly two parameters: a size parameter n and a structure parameter k. The latter parameter ranges between some fixed min value, and a max value depending on n. The higher k becomes, the more "bottleneck" structure is introduced into the CNF. The overall constrainedness, however, remains the same: in SPH, the number of needed holes does not change with k, in the planning domains the number of needed action steps does not change with k. We denote the formulas with SPH^k_n, MAP^k_n, and SBW^k_n, respectively.

2.1 SPH

In SPH^k_n, 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 one hole each. The range of k is between 1 and n. Independently of k, $n + 1$ holes are needed overall. In particular the CNF is unsatisfiable.

Precisely, the CNF is the following. The variables are x_y where x is a pigeon and y is a hole: x_y set to 1 means that x is assigned to y. The bad pigeon is $x = 0$, the good pigeons are $x = 1, \dots, x = k - 1$, the normal pigeons are $x = k, \dots, x = n$. The holes are $y = 1, \dots, n$. The clauses are:

- $\{x\_1, \dots, x\_n\}$ for all $x \neq 0$: all pigeons except the bad one need one hole.

- $\{\neg x\_y, \neg x'\_y\}$ for all $x, x' \neq 0$, $x < x'$, and all y: no two normal or good pigeons can share a hole.
- $\{\neg 0\_y, \neg x\_y\}$ for all $x \geq k$ and all y: none of the normal pigeons can share a hole with the bad pigeon.
- $GEQ(\{0\_1, \ldots, 0\_n\}, k)$: a set of clauses that is satisfiable iff at least k variables out of the set $\{0\_1, \ldots, 0\_n\}$ are set to 1. We consider two options for the definition of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$, see below.

The most straightforward, *naive*, definition of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ is the set of clauses $\{0\_y \mid y \in Y\}$ for all $Y \subseteq \{1, \ldots, n\}, |Y| = n - k + 1$. To see that this is a correct encoding, observe that the size of the sets Y_0 is chosen so that, when removing the holes Y_0 from Y, only $k - 1$ holes are left. So if p_0 occupies k holes then out of each set Y_0, p_0 occupies at least one hole. The other way around, if p_0 occupies only (at most) $k - 1$ holes, then (any $n - k + 1$ subset of) the complement of these holes forms a set Y_0 where the respective clause evaluates to 0.

The drawback of the naive definition of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ is that it can require an exponential number of clauses, e.g. if one sets k to $n/2$ and lets n range. If one fixes k, or the difference between n and k, to some value, however (as e.g. at the bottom/top ends of the k scale where k is fixed to $1/n - k$ is fixed to 0), then the number of clauses is polynomial (in n). One can obtain a definition of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ that is polynomial in both n and k by introducing additional auxiliary variables, and connect them via appropriate clauses to implement a counter of the holes the bad pigeon is assigned to. We discuss such an encoding further below. For the moment, i.e. for the following formal discussion, we assume the naive definition of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ in the SPH formulas. Beside being more clear and easier to understand, these formulas have the desirable property that they are a generalization of the standard pigeon hole formula. If we set $k = 1$ then we obtain exactly the standard formula using the single clause $GEQ(\{0\_1, \ldots, 0\_n\}, 1) = \{\{0\_1, \ldots, 0\_n\}\}$ to ensure that the bad pigeon is assigned to at least one hole.

For $k = n$, we obtain a formula that is inconsistent under UP: $|Y| = n - k + 1 = 1$ so we get the n clauses $\{0\_1\} \ldots \{0\_n\}$. In words, the bad pigeon occupies all holes. By UP, the clause $\{n\_1, \ldots, n\_n\}$ becomes empty, i.e. there is no space for the last remaining normal pigeon. For $k = n - 1$, the CNF formula is consistent under UP.

The intuition behind our construction is that, as k increases, the bad pigeon becomes a bottleneck that yields smaller backdoors. It turns out that this intuition is exactly right, in the form of a linear decrease.

Theorem 1 (SPH) *Let $n > 1$, $1 \leq k \leq n - 1$. To SPH_n^k, there is a backdoor of size* $(n - k) * (n - 1)$.

The backdoors of size $(n-k)*(n-1)$ are described below. We conjecture that the proved upper bound on backdoor size is also a lower bound. For all formulas with $n \leq 5$, we verified this empirically, by enumerating all smaller variable sets.[6]

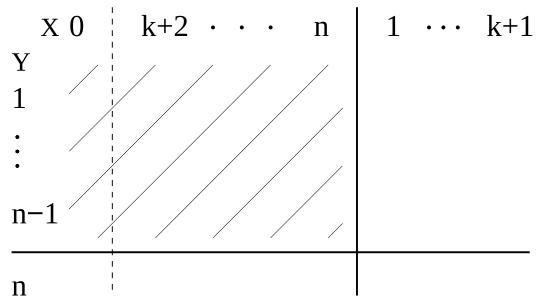


Figure 1: A $(n - k) * (n - 1)$ backdoor in SPH_n^k.

The $(n - k) * (n - 1)$ backdoors in SPH_n^k are illustrated in Figure 1. One just selects all but two of the normal pigeons (i.e., $n-k-1$ of them) as set X, and $n-1$ of the locations as set Y. The variable set $\{x\_y \mid x \in X \cup \{0\}, y \in Y\}$ is then a backdoor. (Figure 1 shows the case where one selected all normal pigeons except $x = k$ and $x = k + 1$, and all holes except $y = n$.) It involves a number of case distinctions, but is not overly difficult to see, that after setting all variables in the backdoor at least $n - 1$ holes are occupied by UP. Then both remaining normal pigeons (those not in X) are forced into the single remaining hole, yielding an empty clause. The main reason why less variables are needed with increasing k is that the naive encoding of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ has the following *efficiency* property: if a partial assignment assigns 0_y to 0 for $n - k$ holes y, then UP sets all other $0\_y'$ variables to 1; if a partial assignment assigns 0_y to 0 for more than $n - k$ holes y, then UP produces an empty clause. I.e., UP suffices to find out if there is just enough/not enough space left for the bad pigeon (the same holds true for the $k = 1$ needed holes of any other pigeon).

As k increases, the backdoor size goes down from a linear function in the total number of variables (namely, $(n-1)*(n-1)$ of $n*(n+1)$ variables) to a square root function ($n - 1$) in that number.

As opposed to the naive encoding of $GEQ(\{0\_1, \ldots, 0\_n\}, k)$ treated above, Bailleux and Boufkhad [1] propose a polynomial-size encoding called

[6]Enumerating variable sets was also our method for finding the optimal backdoors in the first place.

$\Psi(\{0\_1, \ldots, 0\_n\}, k, n)$ (in $\Psi(V, l, u)$, u is an upper bound on the variables set to 1). The encoding introduces $\Theta(nlogn)$ new variables, and $\Theta(n^2)$ clauses, to implement a tree-shaped counting procedure that creates a vector of n output variables where all holes assigned to the bad pigeon are moved to one side.[7] One can then simply check, by looking at that side of the output variables, if the bad pigeon occupies enough holes. Theorem 1 remains valid because, as proved by Bailleux and Boufkhad, $\Psi(\{0\_1, \ldots, 0\_n\}, k, n)$ has the same efficiency property as mentioned above for the naive encoding. The question is whether the backdoors can become smaller. We empirically found that this can indeed be the case.

For the case $n = 4/k = 1$ our enumeration procedure found backdoors of size 8, rather than the 9 variables necessary in the naive encoding. The reason for the smaller backdoor here is that, in the Bailleux and Boufkhad encoding, setting only one (particular one) of the variables in the counter tree suffices to select, by UP, *two* holes (adjacent leaf nodes of the counter tree) in one of which the bad pigeon must lie. If, of every pair of adjacent leaf nodes, only one is left open, then the bad pigeon gets forced into a hole this way. When $n - 2$ holes were already occupied, this leaves only a single hole open and UP determines a contradiction.

In more detail, for arbitrary values of n (while keeping $k = 1$), we get the following. From the backdoor according to Theorem 1 for $k = 1$, to obtain the new backdoor one can skip $\lfloor n/2 \rfloor$ variables regarding normal pigeons. Namely, a particular one of the $n - 1$ variables regarding each of $\lfloor n/2 \rfloor$ selected normal pigeons can be skipped, without losing the property that: after value assignment, at least $n-2$ holes are occupied, and of each pair of holes at least one is occupied. The number of normal pigeons of which we can *not* skip a variable (without losing the latter property) is $\lceil n/2 \rceil - 1$. In place of the skipped variables, we need to select some variables from the counter tree, enough to ensure that UP can determine in what pairs of holes the bad pigeon can lie. When n is a power of 2, $n = 2^l > 2$, the number of tree variables that must be selected is $2^{l-1} - 1$. So then the backdoor we get this way has size $(n-1) * (\lceil n/2 \rceil - 1) + (n-2) * \lfloor n/2 \rfloor + 2^{l-1} - 1$. With $n = 2^l$ this simplifies to $(n-1) * (2^{l-1} - 1) + (n-2) * 2^{l-1} + 2^{l-1} - 1$ which is equal to $(n-1) * (2^l - 1) - 1 = (n-1) * (n-1) - 1$, i.e. one variable less than the $(n-1) * (n-1)$ variables needed according to Theorem 1.

For $n = 4$, the new backdoor is optimal; we did not check (were not able to check empirically) if this is also the case for larger n where n is a power of 2, but we conjecture that it is so. When n is not a power of 2, then it suffices to select $2^{\lceil log_2 n \rceil - 1} - 1$ variables from the counter tree; probably, fewer tree variables suffice in certain cases, we did not check that in detail. Neither did we investigate what the effects of UP are, and what optimal backdoors result, for arbitrary values

[7]I.e. in any satisfying assignment to $\Psi(\{0\_1, \ldots, 0\_n\}, k, n)$ the new variables behave in this way.

of k. Probably, the size of the optimal backdoors becomes somewhat smaller in general, but only for rather high values of n and k, beyond the reach of complete enumeration. For $n \leq 3$, for $n = 4/k = 2$, for $n = 4/k = 3$, for $n = 5/k = 4$, for $n = 5/k = 3$, for $n = 6/k = 5$, and for $n = 6/k = 7$ the optimal backdoor size is the same as under the naive encoding.

2.2 MAP

The two planning domains are formalized in STRIPS [7]. This is the following simple problem-solving formalism. World states are described as sets of (the currently true) propositional facts. The task is to find a path of *actions* leading from a given *initial state* I to a state that contains a given set G of *goal* facts. Actions a have a *precondition* $pre(a)$ requiring a set of facts to be true, and an *effect* making one set $add(a)$ of facts (the *add effects*) true and another set $del(a)$ of facts (the *delete effects*) false.

SAT encodings of STRIPS planning were first proposed in the context of the Blackbox system [11, 10]. Here, we use a somewhat simplified version of the so-called *Graphplan-based* encoding, which is the most efficient (known) encoding for solving practical domains. Our simplification of the encoding makes it easier to determine the exact behavior of UP, while largely preserving the original encoding's efficiency.

The Blackbox approach to planning works as follows. Set bound b on the plan length to 0. Generate a CNF ϕ formula that is satisfiable iff there exists a plan of length b. Use a SAT solver to decide satisfiability. If ϕ is satisfiable, then generate a plan based on the satisfying assignment. Otherwise, increment b and iterate. Obviously, this way, if there is a plan, this approach will find an optimal plan when b has reached the respective length.

In our two synthetic planning domains, we consider CNF encodings that are just one step too short, i.e., if m action steps are needed to solve a task then our CNF encodes $b = m - 1$ steps only. The variables used in our encoding take the form $a(t)$ where a is an action and t is a *time step*, i.e. $1 \leq t \leq m - 1$. The meaning is that $a(t)$ is set to 1 iff action a will be executed at time t. We use artificial *NOOP* actions to concisely encode commitments to keeping some fact true in a time step. A NOOP for fact p has precondition $pre(a) = \{p\}$, add effects $add(a) = \{p\}$, and no delete effects. For each fact p, we have such a NOOP, called $NOOP$-p. Setting $NOOP$-$p(t)$ to 1 means, intuitively, that the search commits to not deleting p at time t.

An action a is *present* in a time step t only if all preconditions of the action *can be true* at time t. The latter is a simple over-approximation of reachability: a fact can be true at time 1 iff it is contained in the initial state. A fact p can be true at a time $t > 1$ iff there is at least one action a present at $t - 1$, with $p \in add(a)$.

The action variables $a(t)$ in the encoding correspond to all a, t pairs where action a is present at t. The clauses in the encoding are the following:

- *Action precondition (AC) clauses*. These clauses ensure that action preconditions are satisfied. For every action a at every time step $t > 1$, for every precondition fact p of a, we have the clause $\{\neg a(t), a_1(t-1), \dots, a_l(t-1)\}$, where $a_1, \dots, a_l$ are all actions present at $t - 1$ that add p. Note that action preconditions at time 1 don't need to be supported (they are true in the initial state).

- *Goal (GC) clauses*. These clauses ensure that the goals are satisfied after the last time step. For every goal fact g, we have the clause $a_1(m - 1), \dots, a_l(m - 1)\}$, where $a_1, \dots, a_l$ are all actions present at $m - 1$ that add g.

- *Exclusion (EC) clauses*. These clauses ensure that *incompatible* pairs of actions can not be executed at the same time step. A pair a, a' of actions is called incompatible if either both are not NOOPs, or a (a') is a NOOP for fact p and a' (a) deletes p. The inserted clause is $\{\neg a(t), \neg a'(t)\}$ for all such pairs of incompatible actions, for all time steps t where both are present.[8]

In the MAP domain, the task is to visit a number of locations on the road map graph, parameterized by n, shown in Figure 2. Initially one is located at L_1^0. The only available actions have the form $move\text{-}x\text{-}y$, with precondition $\{at\text{-}x\}$, add effects $\{at\text{-}y, visited\text{-}y\}$, and delete effects $\{at\text{-}y\}$. Such an action is available for every pair x, y of locations that is connected by an edge in the road map graph. Remember that any pair of non-NOOP 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.

What locations must be visited depends on the value of k. That value ranges in $k \in \{1, 3, \dots, 2n - 3\}$; we require $n > 1$. If $k = 1$ then the goal is to visit each of $L_1^1, \dots, L_n^1$. For each increase of k by 2, the goal on the L_1-branch wanders out by two steps, and one of the other goals is skipped. That is, the goal is to visit L_1^k, $L_2^1, \dots, L_{n-(k-1)/2}$. For $k = 2n - 3$ this is L_1^{2n-3}, L_2^1. We refer to $k = 1$ as the *bottom case*, and to $k = 2n - 3$ as the *top case*. These cases are indicated by the “b” and “t” inscriptions in Figure 2. Note that the length of a shortest plan is $2n - 1$ independently of k. Our CNFs encode $2n - 2$ steps.

If one was to set $k = 2n - 1$, then the CNF MAP$_n^k$ could not even be built. The last time step would not contain a supporting action for the goal (no such

[8]In the Graphplan-based encoding, additional pairs of incompatible actions are identified by a more accurate approximation of reachability, yielding better runtime performance in many domains. In the MAP domain, our results below remain valid when including such additional clauses. We believe that the same holds true in the SBW domain, but we haven't checked that in detail yet.

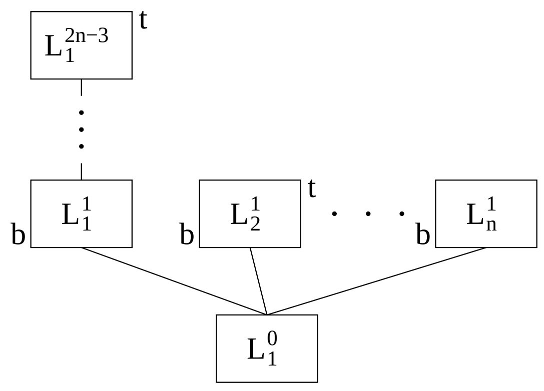


Figure 2: The MAP road map; “b” and “t” indicate the goal locations in the bottom case and the top case, respectively.

action would be present). With $k \in \{1, 3, \dots, 2n-3\}$, the CNF is consistent under UP. The intuition behind the definition of MAP$_n^k$ is that, as k increases, the L_1^k goal more and more becomes a bottleneck that can be exploited for efficient reasoning. We will now see how this intuition translates into the form of the optimal backdoors in the bottom and top cases.

In the bottom case, to prove unsatisfiability one has to do case distinctions for a number of variables square in n.

Theorem 2 (MAP bottom case) *Let $n > 1$. To MAP$_n^1$, there is a backdoor of size* $2n^2 - 7n + 7$.

We conjecture this is also a lower bound on backdoor size; for all $n \leq 4$ 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.

The $\Theta(n^2)$ number of backdoor variables comes from a linear number $(n-1)$ of map branches at a linear number $(n-2)$ of time steps. In more detail, at all time steps $t \in \{3, \dots, 2n-3\}$, for all but the L_1-branch on the map, the backdoor contains variables. For $n-2$ of the branches $i \neq 1$ the variables are $move\text{-}L_1^0\text{-}L_i^1(t)$ and $NOOP\text{-}visited\text{-}L_i^1(t)$, for one branch $i \neq 1$ there is only $move\text{-}L_1^0\text{-}L_i^1(t)$.[9] Finally, the backdoor contains the variable $NOOP\text{-}at\text{-}L_1^0(1)$, yielding the overall size $(n-2) * [(n-2) * 2 + 1] + 1 = 2n^2 - 7n + 7$.

We remark that the DPLL search space induced by the above backdoor set is exponential in n independently of the order in that one considers the variables. In any subset of the backdoor obtained by leaving out at least one of the time steps

[9]A variable $NOOP\text{-}visited\text{-}L_i^1(t)$ encodes the commitment to have or have not visited location L_i^1 by time t.

$t \in \{3, \dots, 2n-3\}$, all assignments are consistent under UP that assign exactly one move variable to 1 in each step. That is, to see that there is not enough time, UP has to have commitments to *all* time steps.

Intuitively, the backdoor in the bottom case has square size in n because n branches are involved at $2n-2$ time steps. As a first guess, one would expect that, in the top case, a DPLL proof involving only a single branch (branch 1, namely) could yield a backdoor of linear size in n. It turns out one can do much better than that.

Theorem 3 (MAP top case) *Let $n > 1$. To MAP$_n^{2n-3}$, there is a backdoor of size $\lceil log_2 n \rceil$.*

We conjecture that this is also a lower bound; for all $n \leq 8$ we verified this empirically. Note that n itself is asymptotic to the *square root* of the total number of variables in the CNF.

It is instructive to have a closer look at how the log-size backdoors in the top case arise. They have the form

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

That is, starting with $move\text{-}L_1^0\text{-}L_1^1(1)$, one has $move\text{-}L_1^{t-2}\text{-}L_1^{t-1}(t-1)$ variables where the value of t is doubled between each two variables. For $n = 8$, the backdoor is $\{move\text{-}L_1^0\text{-}L_1^1(1), move\text{-}L_1^2\text{-}L_1^3(3), move\text{-}L_1^6\text{-}L_1^7(7)\}$. Let us sketch why branching over this set of variables suffices to prove unsatisfiability. Figure 3 contains an illustration.

If one assigns $move\text{-}L_1^0\text{-}L_1^1(1)$ the value 0, then 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 $move\text{-}L_1^0\text{-}L_2^1(1)$ to 1, yielding a contradiction to the precondition clause of $move\text{-}L_1^0\text{-}L_1^1(2)$, that was already set to 1. So we must assign $move\text{-}L_1^0\text{-}L_1^1(1)$ the value 1 instead of 0. Now, say one assigns $move\text{-}L_1^2\text{-}L_1^3(3)$ the value 0. By UP 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 $move\text{-}L_1^0\text{-}L_1^1$ at step 1. This forces us to move back to L_1^0 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 $move\text{-}L_1^2\text{-}L_1^3(3)$ the value 1. Next step, if we assign $move\text{-}L_1^6\text{-}L_1^7(7)$ the value 0, then moves are forced at $8 \leq t \leq 14$, and to achieve the L_2^1 goal at step 7 we have to **move back from L_1^3 to L_1^0** in the steps 4, 5, and 6. A move to L_2^1 is forced at step 7, in contradiction to the move at 8 forced in earlier. Finally, if we assign $move\text{-}L_1^6\text{-}L_1^7(7)$ the value 1 then we need 7 steps to get back to L_1^0, and an eighth

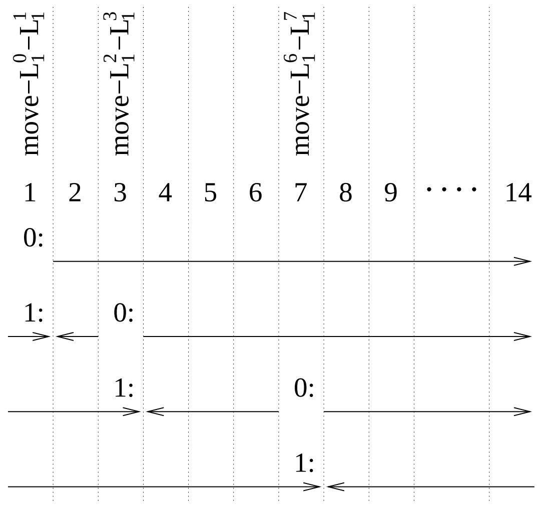


Figure 3: The workings of the optimal backdoor for MAP$_8^{13}$. Arrows indicate moves on the L_1-branch forced in by UP. Direction $\rightarrow$ means towards L_1^{13}, $\leftarrow$ means towards L_1^0. When only a single open step is left, $move$-L_1^0-L_2^1 is forced in at that step by UP, yielding a contradiction.

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 $t = 3$). 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, without losing the backdoor property.

We remark that the above shows that there is a DPLL search tree for MAP$_8^{13}$ that contains only 3 inner (non-failed) nodes: setting $move$-L_1^0-$L_1^1(1)$ to 0 yields a contradiction by UP, after setting it to 1 setting $move$-L_1^2-$L_1^3(3)$ to 0 yields a contradiction by UP, after setting that to 1 setting $move$-L_1^6-$L_1^7(7)$ to either 0 or 1 yields a contradiction by UP. The only non-failed nodes of the tree are those corresponding to the assignments $\emptyset$, $\{move$-L_1^0-$L_1^1(1) \mapsto 1\}$, and $\{move$-L_1^0-$L_1^1(1) \mapsto 1, move$-L_1^2-$L_1^3(3) \mapsto 1\}$. In general (for arbitrary n), the DPLL search tree we get by ordering the backdoor variables this way has $\lceil log_2 n \rceil$ non-failed nodes.

We remark that, while the above variables are a very good choice for branching points in DPLL, there also are very bad choices. For example, consider the variable set $\{NOOP$-$visited$-$L_i^1(3) \mid 2 \leq i \leq n\}$. For none of the 2^{n-1} possible value assignments to these variables does UP cause any propagation. The same holds for any set $\{NOOP\text{-}visited\text{-}L_i^1(t) \mid 2 \leq i \leq n\}$ where $4 \leq t \leq 2n-3$. Indeed, even for the entire set $\{NOOP\text{-}visited\text{-}L_i^1(t) \mid 2 \leq i \leq n, 3 \leq t \leq 2n-3\}$ there are exponentially many value assignments that do not cause any unit propagation, namely all those where there is no pair i and t so that $NOOP\text{-}visited\text{-}L_i^1(t)$ is assigned to 0 and $NOOP\text{-}visited\text{-}L_i^1(t+1)$ is assigned to 1 (the latter enforces a move to L_i^1 at t, yielding a contradiction by UP). Interestingly, even for the $move\text{-}L_1^{t-2}\text{-}L_1^{t-1}$ actions used in the optimal backdoors, sub-optimal choices can be made, namely for the time index. One can set all variables $move\text{-}L_1^{t-2}\text{-}L_1^{t-1}(t)$, instead of the $move\text{-}L_1^{t-2}\text{-}L_1^{t-1}(t-1)$ variables used in the optimal backdoors, to 0 without causing any UP. In particular, $\{move\text{-}L_1^0\text{-}L_1^1(2)$, $move\text{-}L_1^2\text{-}L_1^3(4)$, $move\text{-}L_1^6\text{-}L_1^7(8)\}$ is not a backdoor for MAP_8^{13}.[10]

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 completely infeasible to find the optimal backdoors empirically, and so get a start into the theoretical investigation. We developed and implemented an enumeration program that utilizes symmetries in the planning task to cut down on the number of variable sets to be enumerated, but even with that the enumeration didn't scale up far enough. We leave this topic open for future work.

2.3 SBW

As said, we also constructed and examined a structured version of the Blocksworld planning domain, with restrictions on what blocks can be stacked onto what other blocks.

The parameter n is the number of blocks. They 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. That is, the n blocks must be arranged in a single large stack on top of t_2. The parameter k defines the amount of *stacking restrictions*. There are k "bad" blocks $b_1, \dots, b_k$ and $n-k$ "good" blocks $g_1, \dots, g_{n-k}$. 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. See an illustration in Figure 4.

The operators are the usual Blocksworld operators of the domain version that does not make use of an explicit robot arm, i.e. the predicates dealt with are "on", "on-table", and "clear" (as well as some supplementary stuff to implement the stacking restrictions and the goal). More precisely, there are two kinds of moving actions: those that move a block x from the table to a block y that is above t_2 (a la Move-x-from-t-to-t2, short "movetot2-x-y"), and that move a block x from a block

[10]The resulting DPLL search trees when considering the $move\text{-}L_1^{t-2}\text{-}L_1^{t-1}(t)$ variables aren't very large, since in most cases setting such a variable to 1 makes the formula collapse during UP.

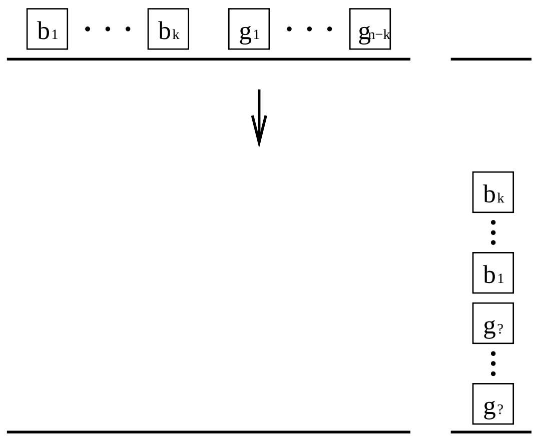


Figure 4: An (incomplete) illustration of the SBW domain.

y above t_2, or from t_2 itself, to t_1 (a la Move-x-from-t2-to-t, short "movefromt2-x-y"). The former action adds a fact stating that now x is "above" t_2, the latter action deletes that fact. Initially only t_2 is "above" itself. The goal condition is the conjunction of "above" for all blocks. As in MAP, the most important feature in our proofs is that every pair of move actions is incompatible, and so setting a move at time t to 1 excludes, by UP, all other move actions at t.

The optimal plan length in SBW$_{n,k}$ is n, independently of k. As before we consider the unsatisfiable CNFs that are just a single step short of a solution, i.e. we consider the CNF encodings with the $n-1$ time steps $1, \dots, n-1$.

With $n = 1$, optimal plan length is 1 so our CNF encoding would have 0 steps and be empty. It is easy to see that, with $n = 2$, even with $k = 0$ SBW$_{n,k}$ is inconsistent under UP – there is just a single time step, into which two moving actions must be fit. It is also easy to see that the CNF can not be constructed – no goal achievers are present – for SBW$_{n,n}$, since only $n-1$ steps are available to stack the n blocks b_i in sequence. Also, SBW$_{n,n-1}$ is inconsistent under UP because there is only just enough time to stack the b_i sequence; each time step is assigned such a stacking action, and there is no time to stack the single g block. Altogether, in what follows we thus assume that $n \geq 3$, and $0 \leq k \leq n-2$.

It turns out that, similar as in the MAP family of formulas, at the bottom end of the k scale the optimal (conjectured) backdoors are linear in the total number of variables.

Theorem 4 (SBW bottom case) *Let $n > 1$. To SBW$_n^0$, there is a backdoor of size $n^3 - 5n^2 + 9n - 6$.*

The total number of variables in the bottom case is also in $\Theta(n^3)$. We conjecture that the proved upper bound is also a lower bound; empirically we could verify this only for $n = 3$.

The $\Theta(n^3)$ number of backdoor variables comes from a linear number of blocks, to be stacked onto a linear number of other blocks, at a linear number of time steps. Precisely, the backdoors we identified have the form

$$\{movetot2\text{-}g_i\text{-}g_j(t) \mid 2 \leq t \leq n-1, 1 \leq i \leq n-2, 0 \leq j \leq n, j \neq i\} \setminus$$

$$\{movetot2\text{-}g_i\text{-}g_j(i+1) \mid 2 \leq i \leq n-2, 0 \leq j \leq n-2\}.$$

That is, at all time steps except the first one, one considers all possibilities of stacking the first $n-2$ blocks onto any other block above t_2; g_0 here denotes t_2, i.e. t_2 is the "lowermost good block" to simplify the notation.[11] In certain cases, precisely for all blocks $2 \leq i \leq n-2$, at time $i+1$, one can, without losing the backdoor property, leave out the moves onto $g_0, \dots, g_{n-2}$. So the total number of variables in the backdoor set is $(n-2)(n-2)n-(n-3)(n-2) = n^3-5n^2+9n-6$.

At the top end of the k scale, as before a logarithmic number of variables suffices.

Theorem 5 (SBW top case) *Let $n > 1$. To SBW$_n^{n-2}$, there is a backdoor of size $2 + \lceil log_2(n/3) \rceil$.*

We verified this as a lower bound for up to $n = 8$, and we conjecture that it is a lower bound in general. The total number of variables in the top case is only a square function in n, so the backdoor size here is, as in MAP, logarithmic in the square root of that number. Table 1 illustrates the backdoors we identified, for up to $n = 48$.

Precisely, the backdoors have the form

$$\{movetot2\text{-}g_1\text{-}t_2(1), movetot2\text{-}g_2\text{-}t_2(1)\} \ \cup$$

$$\{movetot2\text{-}b_{3*2^{i-1}-1}\text{-}b_{3*2^{i-1}-2}(3*2^{i-1}-1) \mid 1 \leq i \leq \lceil log_2(n/3) \rceil\}.$$

The reason for the existence of these logarithmic size backdoors is, similarly to before, that UP can exploit long implication chains between variables at time steps whose distance doubles between each pair of variables. The variables (except those regarding the two good blocks) encode commitments regarding moves of bad blocks at appropriate time steps. The interplay of the variables, and our proof of their backdoor property, is quite reminiscent of the MAP top case. By induction over i, each variable $movetot2\text{-}b_{3*2^{i-1}-1}\text{-}b_{3*2^{i-1}-2}(3*2^{i-1}-1)$ must be set to 1

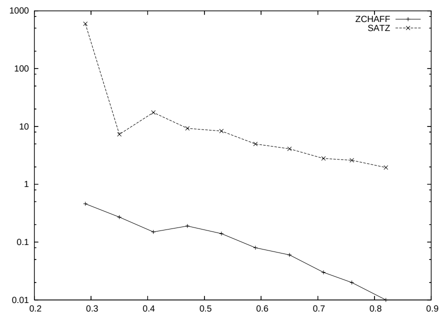
[11]Actually, see Appendix A, one can arbitrarily choose an $n-2$ subset of the blocks, i.e. the selection of $g_1, \dots, g_{n-2}$ is just to simplify the notation.

n	i	(additional) backdoor variable(s)
$3 = 3 * 2^i$	0	$movetot2$-g_1-$t_2(1)$ $movetot2$-g_2-$t_2(1)$
4 5 $6 = 3 * 2^i$	1	$movetot2$-b_2-$b_1(2)$
7 . . $12 = 3 * 2^i$	2	$movetot2$-$b_5, b_4(5)$
13 . . $24 = 3 * 2^i$	3	$movetot2$-b_{11}-$b_{10}(11)$
25 . . $48 = 3 * 2^i$	4	$movetot2$-b_{23}-$b_{22}(23)$

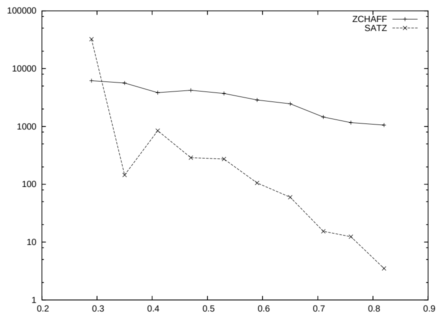
Table 1: An illustration of the optimal backdoors in SBW_n^{n-2}, for $n \leq 48$. Index i indicates the number of variables added on top of $movetot2$-g_1-$t_2(1)$ and $movetot2$-g_1-$t_2(1)$. This is basically the index of the exponentially growing "equivalence classes" of subsequent formulas with the same backdoor set.

or else UP enforces moves "towards the goal" at $1, \dots, 3 * 2^{i-2} - 1$, moves "away from the goal" at $3 * 2^{i-2}, \dots, 3 * 2^{i-1} - 2$, and moves "towards the goal" at $3 * 2^i, \dots, n-1$. Setting the topmost (at the highest time step) variable in the backdoor to 1 yields a contradiction because UP enforces moves "away from the goal" at all higher time steps. In difference to the MAP top case, the time-step distance between the backdoor variables contains a factor 3. This is due to the 2 plus 1 steps needed to move/remove b_2 *and* b_1 to/from t_2, plus scheduling the moves for g_1 and g_2 (giving a contradiction under UP).

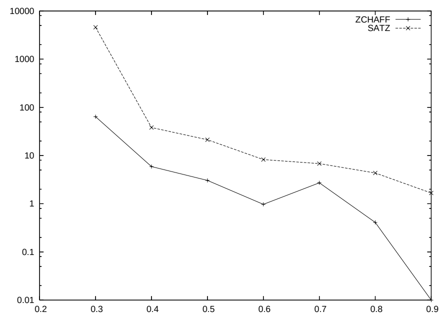
As in MAP, we consider it interesting, but found it extremely difficult, to determine what the optimal backdoors are in general, in particular at what point of the k scale they become logarithmic. We leave this topic open for future work.



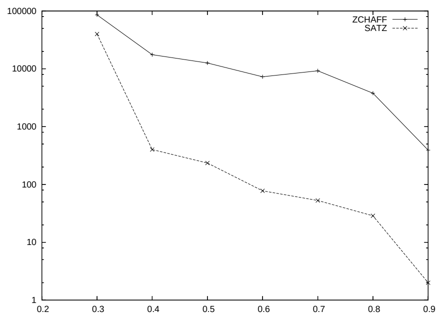
Mean runtime, Driverlog



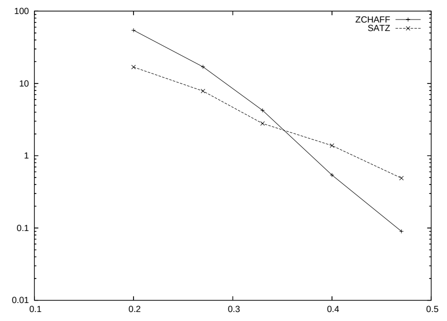
Mean search nodes, Driverlog



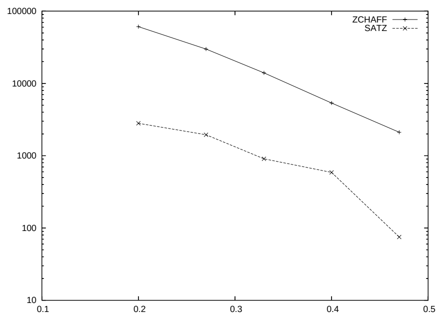
Mean runtime, Blocksworld



Mean search nodes, Blocksworld



Mean runtime, Rovers



Mean search nodes, Rovers

Figure 5: SAT solver performance, on a logarithmic scale, plotted against $BNratio$ in CNFs formulas encoding planning instances from our three test domains.

3 Bottleneck Structure in Real-World CNFs

Our intuition in the construction of the synthetic problems was to capture, in a clean form, structure that appears also in more realistic problems, if in a more blurred form. We ran some empirical experiments to check that our intuitions were correct. Our results suggest that this is indeed the case.

We ran experiments in three commonly used benchmark planning domains: Driverlog, Blocksworld, and Rovers. The first and last were used as benchmarks in the 3rd international planning competition. Driverlog is a complex version of the Logistics domain, involving trucks, drivers, and transportable objects. Drivers and trucks move on different road maps, the task is to bring objects, drivers, and trucks to specified goal locations. Rovers is a simplistic encoding of a space-application where several 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. Blocksworld is the standard domain encoding using a (single) robot arm to rearrange (stack, unstack, pickup, putdown) blocks on a table.

As explained in the introduction, to quantify the amount of "bottleneck" structure in a planning instance we defined the somewhat simplistic notion of the bottleneck ratio. For each goal fact g, we define $cost(g)$ to be the length of a shortest plan achieving (just) g. By $cost(G)$ we denote the length of a shortest plan achieving the entire set G of goals. The bottleneck ratio then is $BNratio := max_{g \in G} cost(g)/cost(G)$.

As in the synthetic planning domains, we considered CNF formulas just one step short of a solution, i.e., with length $cost(G) - 1$, and we compared CNF formulas of the same length (i.e., planning instances with equal solution length). The CNF encoding used was the original Graphplan-based encoding from Blackbox [10]. For each of the three domains, we generated many instances randomly. Instances with equal $cost(G)$ were compared based on SAT solver performance, and $BNratio$. The hypothesis was that solver performance improves with increasing $BNratio$, giving an indication that the "bottleneck" nature occurs in the domains, and yields easier solvable CNF formulas.

In the two synthetic planning domains, with growing n, in the bottom case $BNratio$ converges to 0 while in the top case it converges to 1. Precisely, $BNratio$ is $\frac{1}{2n-2}$ respectively $\frac{2n-3}{2n-2}$ in MAP, and $\frac{1}{n}$ respectively $\frac{n-2}{n}$ in SBW.

For Driverlog and Rovers, we used the random problem generator provided with the competition examples. For Blocksworld, we used the random generator by Slaney and Thiebaux [16]. We ran a few tests to determine instance size parameters that enabled us to compute $BNratio$ reasonably quickly, then we randomly generated thousands of instances of that size. We selected the largest resulting class of instances with equal $cost(G)$ (since the distributions of $cost(G)$ were rather broad, we needed many instances to obtain sufficiently large data sets). We ran ZCHAFF [14] and SATZ [12] on the respective CNF formulas, and measured the number of search nodes and the runtime taken. We plotted the results as a function of $BNratio$, taking the mean of the sets of cases with identical $BNratio$. Figure 5 shows the result plots. Both SAT solvers clearly show the hypothesized performance correlation, across all three domains. (On the side, we observe that, typically, ZCHAFF explores more search nodes than SATZ, but in less time.)

Due to the nature of our experiments – thousands of instances needed to be run in order to obtain sufficient data, and for every one of them we needed to compute $BNratio$ and run ZCHAFF as well as SATZ – the size parameters we could use were relatively small (e.g., 9 blocks in the Blocksworld). So we don't want to make excessive claims on the role of $BNratio$ as a measure of empirical problem hardness. Moreover, of course there are planning domains where there is no "bottleneck" structure whatsoever, e.g. the domain of deadlock detection in Dining-Philosophers. In such domains, $BNratio$ does not, or hardly, change over instances, and our structural observations do not apply. Still, our empirical results suggest that our synthetic CNFs capture a kind of structure that is relevant to at least some more realistic domains.

4 Conclusions

We have presented detailed results on the connections between problem structure and backdoor size in several synthetic domains that we crafted to capture our intuitions about bottleneck behavior in CNF encodings of application domains. The most striking observation is that, depending on the amount of bottleneck behavior, the backdoor sizes can vary on an exponential scale.

Our proved results do not establish lower bounds on the backdoor sizes, but we were able to empirically verify our upper bounds as lower bounds for reasonably large formulas, in most cases. Establishing exact lower bounds analytically is, as remarked earlier, likely to be extremely difficult. So, to obtain meaningful results in reasonably complicated and realistic example domains, our mixed analytical/empirical approach is likely to work best.

We hope to inspire other research groups to perform similar studies. Questions to be investigated in the future include: Are similar kinds of structure relevant in applications from the verification area? What other kinds of structure are relevant? Last but not least, can one come up with novel search algorithms/search heuristics that exploit bottleneck structure?

Acknowledgements

Martin Gütlein has worked, as a student programmer, on the development and implementation of the enhancement of our backdoor enumeration tool to take into account symmetries in planning task descriptions. Jörg Hoffmann is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). Carla Gomes and Bart Selman are partially supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076).

References

[1] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*, pages 108–122, 2003.

[2] S. Buss and G. Turan. Resolution proofs of generalized pigeon hole principles. *Theoretical Computer Science*, 62:311–317, 1988.

[3] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the *really* hard problems are. In *Proc. IJCAI'91*, pages 331–337, 1991.

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

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

[6] R. Dechter. *Constraint Processing*. Morgan-Kauffmann, 2003.

[7] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[8] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[9] K. Iwama and S. Miyazaki. Tree-like resolution is super-polynomially slower than dag-like resolution for the pigeonhole principle. In *Proceedings of Algorithms and Computation*, pages 133–142, 1999.

[10] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 318–325, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[11] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI-96)*, pages 1194–1201. MIT Press, July 1996.

[12] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, Nagoya, Japan, August 1997. Morgan Kaufmann.

[13] David Mitchell. Resolution and constraint satisfaction. In *Proc. CP'03*, 2003.

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference, Las Vegas, June 2001*, pages 530–535, 2001.

[15] I. Rish and R. Dechter. Resolution versus search: Two strategies for SAT. *JAR*, 24(1/2):225–275, 2000.

[16] John Slaney and Sylvie Thiebaux. Blocks world revisited. *Artificial Intelligence*, 125:119–153, 2001.

[17] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In G. Gottlob, editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, August 2003. Kaufmann.

A Proofs

We prove the theorems regarding existence of backdoors of a certain size, for SPH, MAP, and SBW in turn.

A.1 SPH

Theorem 1 *Let $n > 1$, $1 \leq k \leq n-1$. To SPH_n^k, there is a backdoor of size $(n-k) * (n-1)$.*

Proof: Denote by BD any var set such that there exist $X \subseteq \{k, \dots, n\}$, and $Y \subseteq \{1, \dots, n\}$, $|X| = n-k-1$, $|Y| = n-1$, with $BD = \{x\_y \mid x \in X \cup \{0\}, y \in Y\}$. In words, BD talks about $n-1$ (i.e. all but one of the) holes, for the bad pigeon, and for $n - k - 1$ (i.e. all but two of the) normal pigeons. Obviously, $|BD| = (n - k) * (n - 1)$.

Before we prove that BD is a backdoor, observe that the naive encoding of $GEQ(\{0\_1, \dots, 0\_n\}, k)$ has the following efficiency property. For any subset Y', $|Y'| = n - k$, of holes, if $0\_y'$ is set to 0 for all $y' \in Y'$, then 0_y is set to 1 by UP for all $y \notin Y'$. The reason is that, in the respective clause $\{0\_y' \mid y' \in Y' \cup \{y\}\}$ requiring holes for the bad pigeon, only the single open literal 0_y is left. If $|Y'| > n - k$, then one of these clauses is contained in Y' and becomes empty.

Now, let a be any value assignment to the variables in BD. Consider the $n - k - 1$ pigeons x in B_x. For all these x, BD talks about all but one hole, so there is at least one hole y so that $x\_y = 1$ after UP. Since no two x can occupy the same hole, it follows that, when all X variables are set, at least $n - k - 1$ $0\_y'$ variables are set to 0 after UP. If more than $n - k$ $0\_y'$ variables are set to 0, we get an inconsistency with the efficiency property observed above. If exactly $n - k$ $0\_y'$ variables are set to 0, then with the efficiency property the remaining k 0_y variables are set to 1. The holes are then all occupied and the two normal pigeons not in X yield a contradiction. So we can assume that the $n - k - 1$ pigeons in X occupy only one hole each. The respective $n - k - 1$ $0\_y'$ variables are set to 0 after UP. Consider the remaining unset 0_y variables in BD. If a single one of them was set to 0, with what was said above all others would be set to 1. This would yield $n - 1$ occupied holes, and in effect by UP we would get a contradiction with the two normal pigeons not in X. So a must set all remaining unset 0_y variables in BD to 1. But there are at least k of these unset 0_y variables in BD: if all holes occupied by the X pigeons are in Y, then exactly k $(= n - 1 - (n - k - 1))$ variables $0\_y \in BD$ are unset; if one of the holes occupied by the X pigeons is the single hole not in Y, then $k + 1$ variables $0\_y \in BD$ are unset. So, when a sets all unset 0_y variables in BD to 1, at least $n - k - 1 + k = n - 1$ holes are occupied, and again we get an inconsistency with the two normal pigeons not in X. This concludes the argument. ■

A.2 MAP

We first consider the bottom case, then the top case.

A.2.1 Bottom Case

The proof to Theorem 2 is currently in a step-by-step form, i.e. a sequence of lemmas and theorems. We use the following notations. For a CNF ϕ, ϕ_a denotes the CNF that results from inserting the a values, and removing satisfied clauses as well as unsatisfied literals. For a planning CNF, the set of clauses with maximum time t in them are denoted $\phi(t)$. By $UP(\phi)$ we denote the result of applying Unit

Propagation to the CNF ϕ, where again satisfied clauses/unsatisfied literals are removed. Remember that we call the clauses encoding action precondition support *action precondition (AC) clauses*, the clauses encoding goal support *goal (GC) clauses*, and the clauses encoding action incompatibility *exclusion (EC) clauses*.

We consider the formulas MAP^1_n. We denote $T := \{3, 5, \dots, 2n - 3\}$. We denote by $G(\phi)(t)$ the subset of $\{1 \dots n\}$ s.t. for $i \in G(\phi)(t)$: $\{NOOP\text{-}visited\text{-}L^1_i(t), move\text{-}L^0_1\text{-}L^1_i(t)\} \in \phi(t)$. We denote by BD_{full} the set $\{NOOP\text{-}visited\text{-}L^1_i(t), move\text{-}L^0_1\text{-}L^1_i(t) \mid 1 \leq i \leq n, t \in T\}$.

Lemma 1 *Denote with ϕ the CNF formula MAP$_{n,1}$. Let $t \in T$, and let a' be an assignment to the vars in BD_{full} at times $\geq t + 2$. Then, for any assignment a that extends a' to the vars in BD_{full} at times $\geq t$, either $UP(a(\phi))$ is inconsistent or we have $|G(UP(\phi_a))(t-1)| \geq |G(UP(\phi_{a'}))(t+1)| - 1$.*

Proof:

We distinguish three cases. Case 1: $a(move\text{-}L^0_1\text{-}L^1_x(t)) = 1$ for one $x \in \{1 \dots n\}$. Then, by UP, the following assignments follow: $move\text{-}L^0_1\text{-}L^1_{x'}(t) = 0$ for $x' \neq x$ (in particular we get a contradiction if one of these vars is already set to 1); $move\text{-}L^0_1\text{-}L^1_i(t+1) = 0$ for all i; $NOOP\text{-}visited\text{-}L^1_g(t+1) = 1$ for $g \in G(UP(\phi_{a'}))(t+1)$; $NOOP\text{-}visited\text{-}L^1_g(t) = 1$ for $g \in G(UP(\phi_{a'}))(t+1) \setminus \{x\}$ [because the respective drive actions are out already; if these NOOPs are set to 0 we get a contradiction]. The last step proves the claim: for each $g \in G(UP(\phi_{a'}))(t+1) \setminus \{x\}$ we get, from the AC clause to $NOOP\text{-}visited\text{-}L^1_g(t)$, a clause $\{NOOP\text{-}visited\text{-}L^1_i(t-1), move\text{-}L^0_1\text{-}L^1_i(t-1)\}$.

Case 2: $a(move\text{-}L^0_1\text{-}L^1_i(t)) = 0$ for all i, and $a(NOOP\text{-}visited\text{-}L^1_x(t)) = 0$ for one $x \in G(UP(\phi_{a'}))(t+1)$. Then, by UP, the following assignments follow: $NOOP\text{-}visited\text{-}L^1_x(t+1) = 0$ [because both precond achievers are out]; $move\text{-}L^0_1\text{-}L^1_x(t+1) = 1$ [due to G constraint]; $move\text{-}L^0_1\text{-}Lx'1(t+1) = 0$ for $x' \neq x$; $NOOP\text{-}visited\text{-}Lg1(t+1) = 1$ for $g \in G(UP(\phi_{a'}))(t+1) \setminus \{x\}$ [due to G constraints]; $NOOP\text{-}visited\text{-}L^1_g(t) = 1$ for $g \in G(UP(\phi_{a'}))(t+1) \setminus \{x\}$ [because the respective drive actions are assigned out; if these NOOPs are set to 0 we get a contradiction]. The claim now follows with the same argument as in Case 1.

Case 3: none of the above cases holds. Then, in particular, $a(NOOP\text{-}visited\text{-}L^1_g)(t) = 1$ for all $g \in G(UP(\phi_{a'}))(t+1)$. But this directly proves our claim as above.

■

See Figure 6 for an illustration of BD_{full} (and its restrictions as discussed below) with $n = 4$.

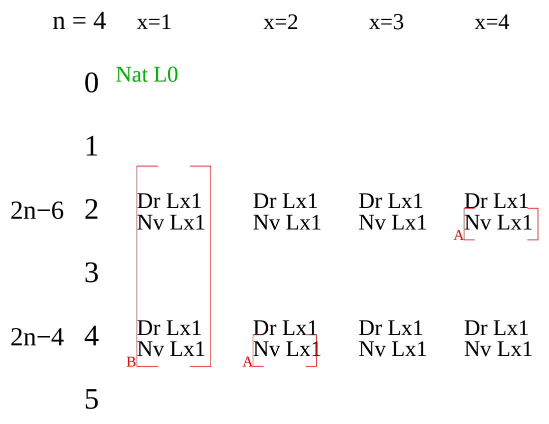


Figure 6: The MAP BD_{full} variables for $n = 4$. The areas marked []A/B illustrate the restrictions A and B as discussed below.

Lemma 2 *Denote with ϕ be the CNF formula MAP$_{n,1}$. Let a be an assignment to some subset of the variables including $NOOP$-at-$L_1^0(1)$. If $|G(UP(\phi_a))(2)| \geq 2$ then $UP(\phi_a)$ is inconsistent.*

Proof:

Assume $a(NOOP\text{-}at\text{-}L_1^0(1)) = 0$. By UP, we get the assignments: $move\text{-}L_1^0\text{-}L_x^1(2) = 0$ and $move\text{-}L_1^0\text{-}L_{x'}^1(2) = 0$ [$NOOP\text{-}at\text{-}L_1^0$ is the only precond achiever]; $NOOP\text{-}visited\text{-}L_x^1(2) = 1$ and $NOOP\text{-}visited\text{-}L_{x'}^1(2) = 1$ [G constraints, see above]; $move\text{-}L_1^0\text{-}L_x^1(1) = 1$ and $move\text{-}L_1^0\text{-}L_{x'}^1(1) = 1$ in contradiction [only precond achievers as NOOP not yet in]. Assuming $a(NOOP\text{-}at\text{-}L_1^0(1)) = 1$, UP gives us: $move\text{-}L_1^0\text{-}L_x^1(1) = 0$ and $move\text{-}L_1^0\text{-}L_{x'}^1(1) = 0$ [incompatible with the NOOP]; $NOOP\text{-}visited\text{-}L_x^1(2) = 0$ and $NOOP\text{-}visited\text{-}L_{x'}^1(2) = 0$ [precondition achievers out]; $move\text{-}L_1^0\text{-}L_x^1(2) = 1$ and $move\text{-}L_1^0\text{-}L_{x'}^1(2) = 1$ in contradiction. This finishes the argument.

■

Denote by $BD_{[B]}$ the var set $BD_{[B]} = \{NOOP\text{-}visited\text{-}L_i^1(t), move\text{-}L_1^0\text{-}L_i^1(t) \mid t \in T, 2 \leq i \leq n\}$. In words, $BD_{[B]}$ results from BD_{full} by skipping, at each step t in T, the vars for the L_1 branch. (The L_1 branch plays a special role since the goal for L_1^1 is the only one that can *not* only be achieved by moving there from L_1^0 – one can also move there from L_1^2.)

Theorem 6 *Denote with ϕ the CNF formula MAP$_{n,1}$. Let a be an arbitrary assignment to the vars in $BD_{[B]} \cup \{NOOP\text{-}at\text{-}L_1^0(1)\}$. Then $UP(\phi_a)$ is inconsistent.*

Proof:

First, observe that a can be constructed by starting with the variables at time $2n-3$ and proceeding downwards. By $a(t)$ we now denote a's restriction to the vars in $BD_{[B]}$ at times $\geq t$. At $t = 2n-3$, $t+1$ is the final encoding layer, and the goal clauses give us $G(UP(\phi_{a(t+2)}))(t+1) = \{2 \dots n\}$. Applying Lemma 1 gives us a contradiction, or $|G(UP(\phi_{a(t)}))(t-1)| \geq n-2$. Iteratively applying the lemma downwards over the additional $n-3$ time points t in T gives us (a contradiction or) $|G(UP(\phi_{a(3)}))(2)| \geq 1$. That is, under unit propagation, no matter what values a assigns to the vars in $BD_{[B]}$ at times ≥ 3, we have (a contradiction or) a y such that $\{NOOP\text{-}visited\text{-}L^1_x(1), move\text{-}L^0_1\text{-}L^1_x(1)\} \in UP(\phi_{a(3)})(2)$.

Next, observe that there are certain regularities regarding the "lemma cases" that can be "used" without inconsistencies. First, if at some t Case 3 is used (all NOOPs are set to 1) then $|G(UP(\phi_{a(2)}))(2)| \geq 2$ follows and we get an inconsistency with Lemma 2. So we can assume that only Cases 1 and 2 are used. Similarly, if at some point t Case 1 or 2 is used on a value that has no G constraint at $t+1$, then all the other G constraints are transported to $t-1$ and we get $|G(UP(\phi_{a(2)}))(1)| \geq 2$, yielding a contradiction. Thus, whenever Case 2 is used, the NOOP at t must be set to 0 for a value i that has a G constraint at $t+1$ (implying that by UP $move\text{-}L^0_1\text{-}L^1_i(t+1)$ is set to 1). Now, say at some consecutive levels t and $t+2$ Case 2 is used at t and Case 1 is used at $t+2$. Then for some i $move\text{-}L^0_1\text{-}L^1_i(t+1)$ is set to 1, and for some other i' $move\text{-}L^0_1\text{-}L^1_{i'}(t+2)$ is set to 1, which is a contradiction [as $move\text{-}L^0_1\text{-}L^1_i(t+1)$ forces out all precond achievers of $move\text{-}L^0_1\text{-}L^1_{i'}(t+2)$]. Thus we can assume that there is some point $t_0 \in T \cup \{1\}$ such that at all $t \leq t_0$ Case 1 is used, and at all $t > t_0$ Case 2 is used. (If $t_0 = 1$ then only Case 2 is used, if $t_0 = 2n-3$ then only Case 1 is used.)

Next, observe that the G constraint for 1 gets transported down to t_0. When using Case 2 at $2n-3$, some $move\text{-}L^0_1\text{-}L^1_i(2n-2)$ is set to 1, and UP sets $move\text{-}L^0_1\text{-}L^1_1(2n-2)$ to 0, $move\text{-}L^2_1\text{-}L^1_1(2n-2)$ to 0, and $NOOP\text{-}visited\text{-}L^1_1(2n-2)$ to 1, producing the G constraint for 1 at $2n-3$. When using Case 2 at t s.t. the G constraint for 1 is present at $t+2$, some $move\text{-}L^0_1\text{-}L^1_i(t+1)$ is set to 1, and UP sets $move\text{-}L^0_1\text{-}L^1_1(t+1)$ to 0, $move\text{-}L^2_1\text{-}L^1_1(t+1)$, $move\text{-}L^0_1\text{-}L^1_1(t+2)$ to 0, $move\text{-}L^2_1\text{-}L^1_1(t+2)$ to 0, $NOOP\text{-}visited\text{-}L^1_1(t+2)$ to 1 (G constraint), and $NOOP\text{-}visited\text{-}L^1_1(t+1)$ to 1, producing the G constraint for 1 at t.

We are now ready to prove the theorem by a case distinction over the value of t_0. Case A: $t_0 = 2n-3$, only Case 1 is used. If $a(NOOP\text{-}at\text{-}L^0_1(1)) = 1$, UP assigns $move\text{-}L^0_1\text{-}L^1_y(2)$ to 1 [G constraint] in contradiction to the $move\text{-}L^0_1\text{-}L^1_i(1)$ that is already set to 1 [by Case 1]. If $a(NOOP\text{-}at\text{-}L^0_1(1)) = 0$, then UP assigns $move\text{-}L^0_1\text{-}L^1_y(2)$ to 0, $NOOP\text{-}visited\text{-}L^1_y(2)$ to 1 [G constraint], and $move\text{-}L^0_1\text{-}L^1_y(1)$ to 1. Then, every 2 layers, $1, 3, \dots, 2n-3$, a $move\text{-}L^0_1\text{-}L^1_i$ var is set to 1. This implies that $move\text{-}L^0_1\text{-}L^1_1(t)$ and (where available) $move\text{-}L^2_1\text{-}L^1_1(t)$ are set to 0 by UP for all $1 \leq t \leq 2n-2$, which implies that the G constraint for

1 at $2n-2$ becomes unachievable.

Case B: $1 < t_0 < 2n-3$, up to t_0 Case 1 is used and above t_0 Case 2 is used. If $a(NOOP\text{-}at\text{-}L_1^0)(1) = 1$ we get a contradiction exactly as in Case A. If $a(NOOP\text{-}at\text{-}L_1^0)(1) = 0$, then similar to Case A we get that $move\text{-}L_1^0\text{-}L_1^1(t)$ and (where available) $move\text{-}L_1^2\text{-}L_1^1(t)$ are set to 0 for all $1 \leq t \leq t_0+1$; $move\text{-}L_1^2\text{-}L_1^1(t_0+2)$ is also set to 0 because the preconditions can not be achieved. UP sets $NOOP\text{-}visited\text{-}L_1^1(t_0+2)$ to 0, and with the above [transportation of the G constraint for 1] UP sets $move\text{-}L_1^0\text{-}L_1^1(t_0+2)$ to 1 in contradiction to the $move\text{-}L_1^0\text{-}L_i^1(t_0+3)$ set to 1 by the Case 2 used at time t_0+2.

Case C: $t_0 = 0$, only Case 2 is used. If $a(NOOP\text{-}at\text{-}L_1^0(1)) = 1$, UP assigns $move\text{-}L_1^0\text{-}L_1^1(1)$ to 0, $move\text{-}L_1^0\text{-}L_y^1(1)$ to 0, $NOOP\text{-}visited\text{-}L_y^1(2)$ to 0, $move\text{-}L_1^0\text{-}L_y^1(2)$ to 1 [G constraint], $move\text{-}L_1^0\text{-}L_1^1(2)$ to 0, $move\text{-}L_1^0\text{-}L_1^1(3)$ to 0, and $move\text{-}L_1^2\text{-}L_1^1(3)$ to 0. As all the moves to L_1^1 are out, the G constraint for 1 at time 3 becomes unachievable. If $a(NOOP\text{-}at\text{-}L_1^0(0)) = 0$, UP sets $move\text{-}L_1^0\text{-}L_y^1(2)$ to 0, $NOOP\text{-}visited\text{-}L_y^1(2)$ to 1 [G constraint], $move\text{-}L_1^0\text{-}L_y^1(1)$ to 1, $move\text{-}L_1^0\text{-}L_1^1(1)$ to 0, $move\text{-}L_1^0\text{-}L_1^1(2)$ to 0, $move\text{-}L_1^2\text{-}L_1^1(3)$ to 0, $NOOP\text{-}visited\text{-}L_1^1(3)$ to 0, and $move\text{-}L_1^0\text{-}L_1^1(3)$ to 1 [transportation of the G constraint for 1] in contradiction to the $move\text{-}L_1^0\text{-}L_i^1(4)$ set to 1 already by the Case 2 used at time 3.

■

Denote by $BD_{[AB]}$ any var set s.t. exists a vector of $x_t \in \{2,\dots,n\}$, with $BD_{[AB]} = \{NOOP\text{-}visited\text{-}L_i^1(t) \mid t \in T, 2 \leq i \leq n, i \neq x_t\} \cup \{move\text{-}L_1^0\text{-}L_i^1(t) \mid t \in T, 2 \leq i \leq n\}$. In words, $BD_{[AB]}$ results from BD_B by skipping one NOOP var at each step t in T.

Theorem 7 *Denote with ϕ the CNF formula MAP$_{n,1}$. Let a be an arbitrary assignment to the vars in a set $BD_{[AB]} \cup \{NOOP\text{-}at\text{-}L_1^0(1)\}$. Then $UP(\phi_a)$ is inconsistent.*

Proof:

Observe that the only way to avoid $|G(UP(\phi_{a(2)}))(1)| \geq 2$ is to get rid of one G constraint at every layer $t \in T$. This means that, at each t, either: Case 1 is used and $move\text{-}L_1^0\text{-}L_{y_t}^1(t)$ is set to 1 for a y_t with G constraint at $t+1$; or Case 2 is used and $move\text{-}L_1^0\text{-}L_{y_t}^1(t+1)$ is set to 1 for a y_t with G constraint at $t+1$; or Case 3 is used and $x_t =: y_t$ has a G constraint at $t+1$ (otherwise all G constraints at $t+1$ are transported to $t-1$ by the $NOOP\text{-}visited\text{-}L_i^1(t)$ vars set to 1). The G constraints must be different for each t, i.e., the y_t in our notation here are pairwise different. If a y_t G constraint has been removed at some t, then below t the $NOOP\text{-}visited\text{-}L_{y_t}^1$ var, if it is in the var set, must be set to 0 to avoid re-introduction of the G constraint. So, when y_1 is the value for which a G constraint is left at point 2, then y_1 is different from all y_t.

Now, say the lowest layer where Case 3 is used is t. No matter what value a assigns to $NOOP$-at-$L_1^0(0)$, $move$-L_1^0-Ly_01 gets set to 1 at either point 0 or point 1, and all other $move$-L_1^0-L_i^1 vars get set to 0 at both points 0 and 1. As Case 1 or 2 is used at all layers below t, $move$-L_1^0-$L_{y_t}^1$ is set to 0 at all these layers (Case 1 is clear, Case 2 at a point $t' < t$ excludes $move$-L_1^0-$L_{y_t}^1(t'+1)$ by UP, and sets $move$-L_1^0-$L_{y_t}^1(t')$ to 0 by the assignment itself). Note that as argued above the $move$-L_1^0-$L_{y_{t'}}1)$ vars set to 1 below t are all such that $y_{t'}$ is different from y_t. It follows that $NOOP$-$visited$-$L_{y_t}^1(t)$ is set to 0 (the precondition achievers are out). But then, with the G constraint for y_t at $t+1$, the assignment we get at t and $t+1$ is exactly Case 2. Applying the argument iteratively upwards to the other layers where Case 3 is used, we end up with an assignment that does not use Case 3 at all. The theorem follows from here with exactly the same arguments as given in the proof to Theorem 6. (The assignments can differ in that some of the $NOOP$-$visited$-$Lx_t{}^1$ vars can remain unset where they are set to 0 in $BD_{[B]}$. But that does not affect the arguments regarding the behavior of Case 1 and Case 2 with respect to each other, the transportation of the G constraint for 1 over the Case 2 layers, and the t_0 case distinction.)

■

The size of $BD_{[AB]} \cup \{NOOP\text{-}at\text{-}L_1^0(1)\}$ is $(n-2)*[(n-2)*2+1]+1 = 2n^2-7n+7$. So Theorem 2 follows from Theorem 7.

A.2.2 Top Case

Denote by BD_{top} the set $\{move\text{-}L_1^{2^i-2}\text{-}L_1^{2^i-1}(2^i-1) \mid 1 \leq i \leq \lceil log_2 n \rceil\}$. See Figure 7 for an illustration of BD_{top} with $n=4$. We have 1 var for $n=2$, 2 vars for $n=3,4$, 3 vars for $n=5,6,7,8$, 4 vars for $n=9,10,11,12,13,14,15,16$, and so on. The vars are $move$-L_1^0-$L_1^1(1)$, $move$-L_1^2-$L_1^3(3)$, $move$-L_1^6-$L_1^7(7)$, $move$-L_1^{14}-$L_1^{15}(15), \dots$

Theorem 8 *Let ϕ be the CNF encoding of MAP-L_n^{2n-3}. Let a be an arbitrary assignment to the vars in BD_{top}. Then $UP(\phi_a)$ is inconsistent.*

Proof:

We start with two observations:

- **(O1)** $move$-L_1^i-$L_1^{i+1}(i+1)$ **set to 1 implies, by UP,** $move$-L_1^j-$L_1^{j+1}(j+1)$ **set to 1 for all** $1 \leq j \leq i$**.** This is because, from i downwards, the only precond achiever in the level below is the (next lower) $move$.

- **(O2)** $move$-L_1^i-$L_1^{i+1}(i+1)$ **set to 0 implies, by UP,** $move$-L_1^j-$L_1^{j+1}(j+2)$ **set to 1 for all** $i \leq j \leq 2n-4$**.** This is because, with the

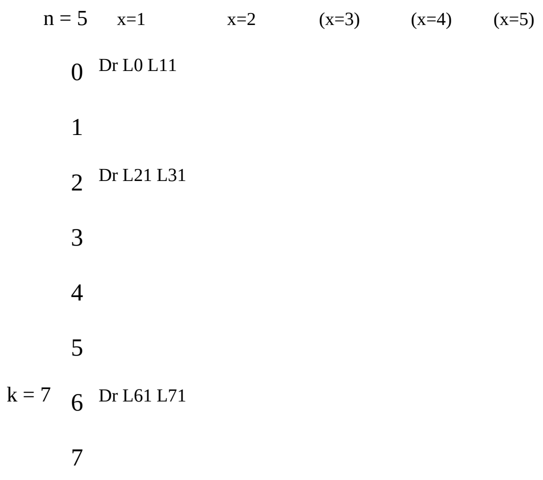


Figure 7: The MAP BD_{top} variables for $n = 5(6, 7, 8)$.

> loss of the add support of $move$-L_1^i-$L_1^{i+1}(i+1)$, $move$-L_1^{i+1}-$L_1^{i+2}(i+2)$ (as well as $NOOP$-at-$L_1^{i+1}(i+2)$) is forced out and so on until $move$-L_1^{2n-4}-$L_1^{2n-3}(2n-3)$ (and $NOOP$-at-$L_1^{2n-4}(2n-3)$). But then $NOOP$-at-$L_1^{2n-3}(2n-2)$ is forced out, and $move$-L_1^{2n-4}-$L_1^{2n-3}(2n-2)$ is forced in [G constraint], This propagates downwards until $move$-$L_1^i, L_1^{i+1}(i+2)$ because all the NOOPs in the respective levels are already forced out.

We prove that, for all $1 \leq i \leq \lceil log_2 n \rceil$, **(**)** $move$-$L_1^{2^i-2}$-$L_1^{2^i-1}(2^i-1)$ **must be set to** 1 **or else UP yields a contradiction.** This suffices because setting $move$-$L_1^{2^{\lceil log_2 n \rceil}-2}$-$L_1^{2^{\lceil log_2 n \rceil}-1}(2^{\lceil log_2 n \rceil}-1)$ to 1 yields a contradiction. Let us first show the latter. With **(O1)**, below $2^{\lceil log_2 n \rceil}$ every layer has a $move$ set to 1 so all other $move$s are out, in particular $move$-L_1^0-L_2^1. To re-insert $move$-L_1^0-L_2^1, one needs, by sequencing AC clauses i.e. going back from $L_1^{2^{\lceil log_2 n \rceil}-1}$ to L_1^0, $2^{\lceil log_2 n \rceil}-1$ steps. In all, $2^{\lceil log_2 n \rceil}$ steps to reach the branch-2 goal. Note that $2^{\lceil log_2 n \rceil} \geq n$. From layer $2^{\lceil log_2 n \rceil}$, there are only $2n - 2^{\lceil log_2 n \rceil} - 1$ layers left. With $2^{\lceil log_2 n \rceil} \geq n$, we have at most $n-1$ layers left, which is less than the number of needed layers, $\geq n$. So the branch-2 G constraint becomes unachievable.

We now prove **(**)** by induction over i. Base case, $i = 1$, $move$-L_1^0-$L_1^1(1)$. If we set that to 0, then by **(O2)** all $move$-L_1^j-$L_1^{j+1}(j+1)$ are forced in for $1 \leq j \leq 2n-3$. This means, in particular, that $move$-L_1^0-L_2^1 is forced out at all layers > 1. So the branch-2 G constraint gets transported by NOOPs down to layer 1, and $move$-L_1^0-$L_2^1(1)$ gets forced in, in contradiction to $move$-L_1^0-$L_1^1(2)$ which is already in. Inductive case, $i \rightarrow i+1$. We assume that $move$-$L_1^{2^i-2}$-$L_1^{2^i-1}(2^i-1)$ is set to 1, and by **(O1)** we get $move$-L_1^j-$L_1^{j+1}(j+1)$ set to 1 for all $1 \leq$

$j < 2^i - 1$. So $move$-L_1^0-L_2^1 is out for all layers $\leq 2^i - 1$. Say we set $move$-$L_1^{2^{i+1}-2}$-$L_1^{2^{i+1}-1}(2^{i+1} - 1)$ to 0. Then by **(O2)** we get $move$s at all layers $> 2^{i+1} - 1$, so $move$-L_1^0-L_2^1 is also out for all layers $> 2^{i+1} - 1$. We get the branch-2 G constraint at layer $2^{i+1} - 1$. Between layer $2^i - 1$ and layer $2^{i+1} - 1$, it are 2^i layers. But the number of steps we need to make the $move$-L_1^0-L_2^1 precondition achievable, i.e. to get back to L_1^0 from $L_1^{2^i-1}$, is $2^i - 1$. So $move$-L_1^0-L_2^1 first re-appears (is not set to 0 by AC clauses) in layer $2^{i+1} - 1$. Due to the G constraint it is set to 1, and iteratively downwards the same happens with the $move$s back from $L_1^{2^i-1}$ to L_1^0. But then $move$-$L_1^{2^{i+1}-2}$-$L_1^{2^{i+1}-1}(2^{i+1})$, which is already set to 1, becomes unachievable.

■

Theorem 8 obviously entails Theorem 3.

A.3 SBW

We first consider the bottom case, then the top case.

A.3.1 Bottom Case

In the case $k = 0$, there are only "good" blocks, with no restrictions whatsoever on the possible stacking order. With $n = 2$ this is inconsistent under UP, so we assume $n \geq 3$ and $k = 0$. We use the convention that g_0 denotes t_2, i.e. t_2 is the "lowermost good block" – this is useful to simplify the notation in the proofs.

Denote by BD_{full} any variable set such that exists

$$X = \{x_1, \dots, x_{n-2}\} \subseteq \{g_1, \dots, g_n\}$$

with

$$BD_{full} = \{movetot2\text{-}x_i\text{-}g_j(t) \mid 2 \leq t \leq n-1, 1 \leq i \leq n-2, 0 \leq j \leq n, j \neq x_i\}.$$

In words, we select an $n - 2$ subset of the blocks, and then include into BD_{full}, at all time steps except the first one (number 1), for all selected blocks x, all possibilities of getting x above s. For the rest of this section, for simplicity of the presentation we will assume that $X = \{g_1, \dots, g_{n-2}\}$. Obviously, this does not make us lose generality since the blocks are all symmetric. See Figure 8 for an illustration of BD_{full} in the case $n = 5$.

We re-use (an obvious generalization of) the shorthand notation from Figure 8 in the proof that BD_{full} is a strong backdoor.

Theorem 9 *For $n \geq 3$, BD_{full} is a backdoor in SBW-$L_{n,0}$.*

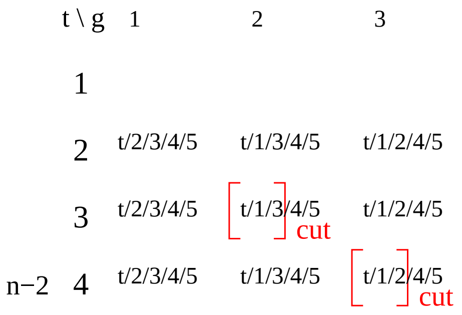


Figure 8: An illustration of BD_{full} for $n = 5$. The notations $t/2/3/4/5$ are shorthand for $movetot2$-g_1-t_2, $movetot2$-g_1-g_2, $movetot2$-g_1-g_3, $movetot2$-g_1-g_4, $movetot2$-g_1-g_5, likewise for $t/1/3/4/5$ and $t/1/2/4/5$. The areas marked "cut" illustrate the BD_{cut} variable set, see below.

Proof: Consider only the BD_{full} variables for one block x in separation, i.e., e.g., with $x = 1$ the variables $t/2/\dots/n$ at times $1, \dots, n-2$. Observe that, when all these variables are assigned a value, then after UP there is at least one time step were a $movetot2$-g_x-g' variable is set to 1. If all the vars are assigned 0 then by UP on the goal constraint for x $NOOP$-$abovet_2$-g_x is set to 1 at time $n-1$, and consecutively at all times $n-3, \dots, 1$; when $NOOP$-$abovet_2$-$g_x(2)$ is set to 1, $movetot2$-g_x-$t_2(1)$ is forced in. With the linearization constraints, it follows that, after assigning values to all BD_{full} variables, after UP there is either an inconsistency, or an $n-2$ subset T of the time steps $1, \dots, n-1$ such that at all $t \in T$ one $movetot2$-g_x-g' variable, $g_x \in X$, is set to 1. But when $n-2$ time steps are occupied by a $movetot2$-g_x-g', only a singe time step t_0 remains open. The goal constraints for the two blocks y and z not in X are transported, by $NOOP$-$abovet_2$-g_y and $NOOP$-$abovet_2$-g_z, down to t_0. All moves $movetot2$-g_y-g' and $movetot2$-g_z-g' are forced out at all times below t_0. So moves $movetot2$-g_y-g' and $movetot2$-g_z-g' are forced in at t_0, yielding an inconsistency. ■

In words, assigning values to BD_{full} forces us to fix moving time steps for the $n-2$ blocks X, and UP then sees that there is no way to move the remaining two blocks within the single remaining time step.

Note that the proof to Theorem 9 makes no use whatsoever of the fact that the actual target blocks g' in the selected moves $movetot2$-g_x-$g'(t)$ can not be arranged arbitrarily, due to the interactions between the moves (e.g. if g_1 was moved onto t_2 in step 1, then we can not move g_2 onto t_2, or g_3, or any other block than g_1, in step 1). We will now see that interactions of this kind lead to smaller backdoors.

Denote by BD_{cut} the variable set $BD_{cut} = BD_{full} \setminus \{movetot2\text{-}g_i\text{-}g_j(i+1) \mid 2 \leq i \leq n-2, 0 \leq j \leq n-2\}$ where as above we assume that the selected variables X for BD_{full} are $X = \{g_1, \dots, g_{n-2}\}$. In words, BD_{cut} results from BD_{full} by skipping, for all blocks $2 \leq i \leq n-2$, at time $i+1$, all possible moves except those to the two blocks not in X. See the marked areas in Figure 8 for an illustration.

Theorem 10 *For $n \geq 3$, BD_{cut} is a backdoor in SBW-$L_{n,0}$.*

Proof: Let a be an assignment to BD_{cut}. We show how an inconsistency can be derived by UP. For simplicity, we identify the blocks g_i with their indices i in what follows. For the *cut-affected* blocks $x \in \{2, \dots, n-2\}$, we refer to the move variables for x at time $x+1$ as the *cut-field* of x. Denote by $X \subseteq \{2, \dots, n-2\}$ the indices of those cut-affected blocks x whose cut-field is not occupied by a, i.e. where a does not assign any BD_{cut} variable at time x to 1. Denote by $\overline{X}$ the other cut-affected blocks, i.e. $\{2, \dots, n-2\} \setminus X$.

Observe that block 1, and all blocks in $\overline{X}$, occupy at least one time step, i.e. for all these blocks y there is a time t at which a move variable for y is set to 1 at t. This is simply due to the goal constraint for y, the fact that all move variables for y at times $1 \leq t \leq n-2$ are contained in BD_{cut} (except possibly the cut-field, but that is occupied), and the fact that UP forces a move at time 1 if no move is in for y at any of the other times. So altogether, the blocks $\{1\} \cup \overline{X}$ occupy at least $1 + |\overline{X}|$ time steps. Clearly, with the definition of X these time steps must be taken out of the set of times $\{1, 2\} \cup \{y+1 \mid y \in \overline{X}\}$ (using the block indices also as indices into time steps; in what follows we abbreviate $\{y+1 \mid y \in \overline{X}\}$ with $\overline{X+1}$). Now, if a block is moved at time 2 then UP immediately enforces an adequate move at time 1. It follows that, by the assignments to $\{1\} \cup \overline{X}$ blocks and UP, either all time steps $\{1, 2\} \cup \overline{X+1}$ are occupied, or the blocks $\{1\} \cup \overline{X}$ occupy all time steps $\{1, 2\} \cup \overline{X+1}$ except a single time step $t_0 \in \{2\} \cup \overline{X+1}$.

Next, observe the following. Say all time steps up to a time l are occupied by the blocks Y, and block $y \in Y$ occupies time l. Then at time $l+1$ the only possible (not forced out by UP) $movetot2$ actions are of the form $movetot2\text{-}x\text{-}y$. This is due to UP chaining over precondition (AC) clauses: no block other than y can be clear and above s at $l+1$. Now, say we additionally have that $NOOP\text{-}abovet_2\text{-}g_y$ is set to 1 at all times $t > l$. Then at all times $t > l+1$ the only possible $movetot2$ actions are of the form $movetot2\text{-}x\text{-}g$ where either $g = y$ or $g \notin Y$. This is also due to UP chaining over precondition clauses: since $NOOP\text{-}abovet_2\text{-}g_y$ is set to 1 at all times, y can not be moved away from where it is, and chaining precondition clauses detects that no block in $Y \setminus \{y\}$ can become clear again.

We are now ready for the final step of the proof. Let x be the smallest (index of a) block in X such that x does not occupy a time step by a. We show that a $movetot2$ action for x is forced to 1 at $x+1$ by UP. We distinguish two cases.

Case 1, $x \neq t_0$. Then all time steps below time $x+1$ are occupied. Say time x is occupied by block y. With the above, $movetot2$-x-y is the only $movetot2$ action for x not forced to 0 at time $x+1$. Now, observe that $NOOP$-$abovet_2$-x is set to 0 at all times $t \leq x+1$, simply because all the time steps below $x+1$ are occupied. Also, observe that $NOOP$-$abovet_2$-x is set to 1 at all times $t > x+1$, due to the goal constraint for x and the fact that all $movetot2$ actions for x at times greater than x belong to BD_{cut} and are set to 0 (as x does not occupy a time step). It follows that $movetot2$-x-y is set to 1 at time $x+1$ by UP – all other possibilities of satisfying the goal constraint at time $x+1$ (transported down to time $x+1$) are out.

Case 2, $x = t_0$, i.e. the blocks $\{1\} \cup \overline{X}$ occupy all time steps in $\{1,2\} \cup \overline{X+1}$, except t_0. Note that, then, each block in $\{1\} \cup \overline{X}$ occupies *exactly* one time step. All time steps below time t_0 are occupied, say by the blocks Y, where $y \in Y$ is the "top" block occupying time $t_0 - 1$. Now, we also have that $NOOP$-$abovet_2$-g_y is set to 1 at all times $t > t_0 - 1$. This is because y occupies only step $t_0 - 1$, so all its BD_{cut} variables at other steps are set to 0 by a; if y belongs to $\overline{X}$, and $y \neq t_0 - 2$, then y's variables at time $y+1$ are set to 0 by construction because time $y+1$ is occupied. Altogether, with what was said above we now have that at all times $t > t_0$, in particular at time t_0+1, the only possible $movetot2$ actions are of the form $movetot2$-x-g where either $g = y$ or $g \notin Y$. As for the $g \notin Y, g \in \{1, 2, \ldots, n-2\}$, the $movetot2$-x-$g(t_0+1)$ variables are forced to 0 by UP since the respective $movetot2$-g-y actions at time t_0 are assigned 0 by a (or otherwise t_0 would be occupied). As for the $g \notin Y, g \in \{n-1, n\}$, both respective variables $movetot2$-x-$g(t_0+1)$ are contained in BD_{cut} and are assigned 0 by a (or otherwise x would occupy a time step by a). We finally observe that $NOOP$-$abovet_2$-$x(t_0+1)$ is out because all $movetot2$ actions for x are out at the (occupied) time steps $t < t_0$, and $movetot2$-x-$y(t_0)$ is contained in BD_{cut} and assigned 0 by a. The goal constraint for x gets transported down to t_0+1 just as argued above in case 1, and it follows that $movetot2$-x-$y(t_0+1)$ is the only remaining option to satisfy that constraint at time $x = t_0+1$, and gets set to 1 by UP.

We now have that x, the smallest block in X such that x does not occupy a time step by a, occupies time step $x+1$ after UP. We can now remove x from X and apply the exact same argument to the new smallest x in X. Iterating this, it follows that all blocks in $\{1, 2, \ldots, n-2\}$ occupy at least one time step after UP. But then, at most only a single time step (namely the time step t_0 above) is left unoccupied. If no time step is left unoccupied, then the goal constraints for the blocks $n-1$ and n become unachievable. Otherwise, these goal constraints are transported into t_0, and the two variables $movetot2$-g_{n-1}-$y(t_0)$ and $movetot2$-g_n-$y(t_0)$ are set to 1 (where y is the block occupying step $t_0 - 1$), yielding an inconsistency. This concludes the argument. ■

The size of BD_{cut} is $(n-2)(n-2)n-(n-3)(n-2) = n^3-5n^2+9n-6$, so Theorem 4 follows.

A.3.2 Top Case

In the top case, $k = n-2$ (as said, for larger values of k the CNF can't be built or is inconsistent under UP). Denote by BD_{top} the set consisting of $movetot2$-g_1-$t_2(1)$ and $movetot2$-g_2-$t_2(1)$, as well as $movetot2$-$b_{3*2^{i-1}-1}$-$b_{3*2^{i-1}-2}(3*2^{i-1}-1)$ for $1 \leq i \leq \lceil log_2(n/3) \rceil$. See Table 1 for an illustration of BD_{top} up to $n = 25$.

The size of BD_{top} is, obviously, $2 + \lceil log_2(n/3) \rceil$. BD_{top} is a backdoor in SBW-L$_{n,n-2}$, implying Theorem 5.

Theorem 11 *For* $n \geq 3$, BD_{top} *is a backdoor in SBW-L*$_{n,n-2}$.

Proof:

First, observe that the claim is true for $n = 3$. There, BD_{top} contains only $movetot2$-g_1-$t_2(1)$ and $movetot2$-g_2-$t_2(1)$. If we set one of these to 1, then the move actions for the other g-block, as well as for the single b-block, get forced out at layer 1, so they get forced in (by their goal constraints) at the only other layer, time 2. This yields an inconsistency with the respective exclusion clause. On the other hand, if we set both $movetot2$-g_1-$t_2(1)$ and $movetot2$-g_2-$t_2(1)$ to 0, then the NOOPs for the respective goal constraints are out at layer 2, and both moves get forced in at that layer, yielding again an inconsistency.

For the rest of the proof, we assume $n > 3$. We next make two observations: (we use the convention that $movetot2$-b_1-b_0 means $movetot2$-b_1-t_2)

- **(O1)** $movetot2$-b_{i+1}-$b_i(i+1)$ **(**$i \geq 1$**) set to 1 implies, by UP,** $movetot2$-b_{j+1}-$b_j(j+1)$ **set to 1 for all** $0 \leq j < i$**.** This is because, from i downwards, the only precond achiever in the level below is the next lower of these $movetot2$ actions.

- **(O2)** $movetot2$-b_{i+1}-$b_i(i+1)$ **set to 0 implies, by UP,** $movetot2$-b_{j+1}-$b_j(j+2)$ **set to 1 for all** $i \leq j \leq n-3$**.** This is because, with the loss of the add support of $movetot2$-b_{i+1}-$b_i(i+1)$, $movetot2$-b_{i+2}-$b_{i+1}(i+2)$ (as well as $NOOP$-$aboves$-$b_{i+1}(i+2)$) is forced out and so on until $movetot2$-b_{n-2}-$b_{n-3}(n-2)$ and $NOOP$-$aboves$-$b_{n-3}(n-2)$ as well as $NOOP$-$aboves$-$b_{n-2}(n-1)$. Then, $movetot2$-b_{n-2}-$b_{n-3}(n-1)$ is forced in [goal constraint], which propagates downwards until $movetot2$-b_{i+1}-$b_i(i+2)$ because all the NOOPs in the respective levels are already forced out.

Let u be the number so that $3*2^{u-1} \leq n \leq 3*2^u$, i.e. u is the i-index of n's BD_{top} equivalence class as illustrated in Table 1 ($u \geq 1$ since $n > 3$).

Note that $u = \lceil log_2(n/3) \rceil$. Keep in mind that $3 * 2^u$ is the largest n in the equivalence class, and that this largest n has a CNF with $n-1$ layers, numbered $1, \dots, n-1 = 3 * 2^u - 1$.

We prove that, for all $1 \leq i \leq u$,

(**) $movetot2$-$b_{3*2^{i-1}-1}$-$b_{3*2^{i-1}-2}(3 * 2^{i-1} - 1)$ **must be set to** 1 **or else UP yields a contradiction.**

This suffices because setting $movetot2$-$b_{3*2^{u-1}-1}$-$b_{3*2^{u-1}-2}(3*2^{u-1}-1)$ to 1 yields a contradiction. Let us first show the latter. With **(O1)**, below $3 * 2^{u-1} - 1$ every layer has a $movetot2$ set to 1 so all other $movetot2$ actions are out, in particular $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2. To re-insert these actions, one needs, by sequencing precondition clauses, i.e. by removing the $b_j, j \leq 3*2^{u-1}-1$ from t_2 again (g-blocks can not be stacked onto b-blocks), to insert $3*2^{u-1}-1$ steps – one for each of the b_j. So $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 re-appear first in layer $3*2^{u-1}+3*2^{u-1}-1 = 3*2^u-1$. But either there aren't that many layers in the CNF, namely if $n < 3 * 2^u$, yielding the g_1 and g_2 goals unachievable; or we get an inconsistency in layer $3 * 2^u - 1$ since it is the only unoccupied (by a movetot2 action) layer left and both $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 are forced to be 1 by their goal constraints.

We now prove **(**)** by induction over i. Base case, $i = 1$, $movetot2$-b_2-$b_1(2)$. If we set that to 0, then by **(O2)** all $movetot2$-b_{j+1}-$b_j(j+2)$ are forced in for $1 \leq j \leq n-3$. This means, in particular, that all layers $t \geq 3$ are occupied, and only layers 1 and 2 are left to achieve the goals for g_1 and g_2. This reduces to exactly the same case as $n = 3$ discussed at the start of the proof.

Inductive case, $i \to i+1$. We assume that $movetot2$-$b_{3*2^{i-1}-1}$-$b_{3*2^{i-1}-2}(3 * 2^{i-1} - 1)$ is set to 1, and by **(O1)** we get that $movetot2$-b_{j+1}-$b_j(j+1)$ is set to 1 for all $0 \leq j < 3 * 2^{i-1} - 2$. So $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 are out for all layers $t \leq 3 * 2^{i-1} - 1$. Say we set $movetot2$-b_{3*2^i-1}-$b_{3*2^i-2}(3 * 2^i - 1)$ to 0. Then by **(O2)** we get $movetot2$ actions at all layers $t > 3 * 2^i - 1$, so $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 are also out for all layers $t > 3 * 2^i - 1$. After applying $movetot2$-$b_{3*2^{i-1}-1}$-$b_{3*2^{i-1}-2}(3 * 2^{i-1} - 1)$, i.e. starting from layer $3 * 2^{i-1}$, the number of steps we need to make the $movetot2$-$g_{1|2}$-t_2 preconditions achievable, i.e. to remove all the b_i from s again, is $3 * 2^{i-1} - 1$ (this is basically the same argument as used further above). So $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 first re-appear in layer $t = 3 * 2^{i-1} + 3 * 2^{i-1} - 1 = 3 * 2^i - 1$. But as observed above all layers $t' > t = 3 * 2^i - 1$ are already occupied (by **(O2)** and since we set $movetot2$-b_{3*2^i-1}-$b_{3*2^i-2}(3 * 2^i - 1)$ to 0), so both $movetot2$-g_1-t_2 and $movetot2$-g_2-t_2 get forced in at $t = 3*2^i-1$ by their goal constraints, yielding an inconsistency with the respective exclusion clause. This concludes the argumentation.

■

The reader who is confused by all the $(3 * 2^{foo} - bar)$ indexing in the proof to Theorem 11 is advised to have a look at Table 1, and instantiate the indices in the proof with the numbers 1 ($= 3 * 2^{1-1} - 2$), 4 ($= 3 * 2^{2-1} - 2$), and 10 ($= 3 * 2^{3-1} - 2$).

I N F O R M A T I K

MPI-I-2005-4-005	M. Goesele	?
MPI-I-2005-4-004	C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A.,. Magnor, H. Seidel	Joint Motion and Reflectance Capture for Creating Relightable 3D Videos
MPI-I-2005-4-003	T. Langer, A.G. Belyaev, H. Seidel	Analysis and Design of Discrete Normals and Curvatures
MPI-I-2005-4-002	O. Schall, A. Belyaev, H. Seidel	Sparse Meshing of Uncertain and Noisy Surface Scattered Data
MPI-I-2005-4-001	M. Fuchs, V. Blanz, H. Lensch, H. Seidel	Reflectance from Images: A Model-Based Approach for Human Faces
MPI-I-2005-2-001	J. Hoffmann, Carla Gomes	Bottleneck Behavior in CNF Formulas
MPI-I-2005-1-007	I. Katriel, M. Kutz	A Faster Algorithm for Computing a Longest Common Increasing Subsequence
MPI-I-2005-1-002	I. Katriel, M. Kutz, M. Skutella	Reachability Substitutes for Planar Digraphs
MPI-I-2005-1-001	D. Michail	Rank-Maximal through Maximum Weight Matchings
MPI-I-2004-NWG3-001	M. Magnor	Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae
MPI-I-2004-NWG1-001	B. Blanchet	Automatic Proof of Strong Secrecy for Security Protocols
MPI-I-2004-5-001	S. Siersdorfer, S. Sizov, G. Weikum	Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning
MPI-I-2004-4-006	K. Dmitriev, V. Havran, H. Seidel	Faster Ray Tracing with SIMD Shaft Culling
MPI-I-2004-4-005	I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel	Neural Meshes: Surface Reconstruction with a Learning Algorithm
MPI-I-2004-4-004	R. Zayer, C. Rssl, H. Seidel	r-Adaptive Parameterization of Surfaces
MPI-I-2004-4-003	Y. Ohtake, A. Belyaev, H. Seidel	3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs
MPI-I-2004-4-002	Y. Ohtake, A. Belyaev, H. Seidel	Quadric-Based Mesh Reconstruction from Scattered Data
MPI-I-2004-4-001	J. Haber, C. Schmitt, M. Koster, H. Seidel	Modeling Hair using a Wisp Hair Model
MPI-I-2004-2-007	S. Wagner	Summaries for While Programs with Recursion
MPI-I-2004-2-002	P. Maier	Intuitionistic LTL and a New Characterization of Safety and Liveness
MPI-I-2004-2-001	H.d. Nivelle, Y. Kazakov	Resolution Decision Procedures for the Guarded Fragment with Transitive Guards
MPI-I-2004-1-006	L.S. Chandran, N. Sivadasan	On the Hadwiger's Conjecture for Graph Products
MPI-I-2004-1-005	S. Schmitt, L. Fousse	A comparison of polynomial evaluation schemes
MPI-I-2004-1-004	N. Sivadasan, P. Sanders, M. Skutella	Online Scheduling with Bounded Migration