Journal of Artificial Intelligence Research 36 (2009) 415-469 Submitted 03/09; published 12/09

Friends or Foes? On Planning as Satisfiability and Abstract CNF Encodings

Carmel Domshlak DCARMEL@IE.TECHNION.AC.IL
Technion – Israel Institute of Technology,
Haifa, Israel
Jörg Hoffmann JOERG.HOFFMANN@INRIA.FR
INRIA,
Nancy, France
Ashish Sabharwal SABHAR@CS.CORNELL.EDU
Cornell University,
Ithaca, NY, USA

Abstract

Planning as satisfiability, as implemented in, for instance, the SATPLAN tool, is a highly competitive method for finding parallel step-optimal plans. A bottleneck in this approach is to *prove the absence* of plans of a certain length. Specifically, if the optimal plan has n steps, then it is typically very costly to prove that there is no plan of length $n-1$. We pursue the idea of leading this proof within solution length preserving abstractions (over-approximations) of the original planning task. This is promising because the abstraction may have a much smaller state space; related methods are highly successful in model checking. In particular, we design a novel abstraction technique based on which one can, in several widely used planning benchmarks, construct abstractions that have exponentially smaller state spaces while preserving the length of an optimal plan.

Surprisingly, the idea turns out to appear quite hopeless in the context of planning as satisfiability. Evaluating our idea empirically, we run experiments on almost all benchmarks of the international planning competitions up to IPC 2004, and find that even hand-made abstractions do not tend to improve the performance of SATPLAN. Exploring these findings from a theoretical point of view, we identify an interesting phenomenon that may cause this behavior. We compare various planning-graph based CNF encodings ϕ of the original planning task with the CNF encodings ϕ^σ of the abstracted planning task. We prove that, in many cases, the shortest resolution refutation for ϕ^σ can never be shorter than that for ϕ. This suggests a fundamental weakness of the approach, and motivates further investigation of the interplay between declarative transition-systems, over-approximating abstractions, and SAT encodings.

1. Introduction

The areas of model checking and AI planning are well-known to be closely related as they both develop tools for automatic behavior analysis of large-scale, declaratively specified transition systems. In particular, both in planning and in model checking of "safety properties"—checking reachability of non-temporal formulas—problems are given by a description of a transition system, by an initial system state, and by a target condition. A solution for such a problem corresponds to a legal path of transitions bringing the system from the initial state to a state satisfying the target condition.

For a model checking problem, a solution corresponds to an "error path" in the system, that is, to an unwanted system behavior. Proving absence of such error paths is the ultimate goal of system verification, and thus the traditional focus of the field is exactly on that. Besides clever symbolic representations of the state space, the key technique to accomplish this ambitious task is *abstraction*. System abstraction corresponds to an over-approximation of the considered transition system, and thus abstraction preserves all the transitions of the original system. Hence, if the abstract transition system does not contain a solution, then neither does the original system. The key to success in model checking is that, in many cases, one can prove the absence of solutions in rather coarse abstractions with a comparatively small state space. Techniques of this kind have been explored in depth for a long time. Arguably its most wide-spread instance in model checking is predicate abstraction (Graf & Saïdi, 1997), where system states form equivalence classes defined in terms of the truth values of a number of expressions (the "predicates"), such as linear expressions over integer system variables. Predicates can be learned by analyzing spurious error paths in too-coarse abstractions (Clarke, Grumberg, Jha, Lu, & Veith, 2003). Methods of this kind have been extremely successful in the verification of temporal safety properties (e.g., Ball, Majumdar, Millstein, & Rajamani, 2001; Chaki, Clarke, Groce, Jha, & Veith, 2003; Henzinger, Jhala, Majumdar, & McMillan, 2004).

In contrast to system verification, the focus in AI planning is on *finding* solutions in instances that are assumed to be solvable. In particular, in optimizing planning, the task is to find a solution that optimizes a certain criterion such as (the focus of our analysis here) the sequential/parallel length of the solution path. Unlike in general planning where any solution is good enough, the main bottleneck in length-optimizing planning is always to *prove the absence* of solutions of a certain length. In particular, if the optimal plan has n steps, then the hardest bit is typically to prove that there is no plan of length $n-1$. Note that this is where the plan is actually proved to be optimal, and no length-optimizing planner can avoid constructing this proof, no matter what computational techniques it is based on.

Our agenda in this research is to apply the above idea from model checking to length-optimizing planning. We lead the optimality proof—non-existence of a plan of length $n-1$—*within an abstraction*. In particular, our focus is on the interplay between abstraction and proving optimality in *parallel step-optimal planning as satisfiability*. This approach was originally proposed by Kautz and Selman (1992), who later developed the SATPLAN tool (Kautz & Selman, 1999; Kautz, 2004; Kautz, Selman, & Hoffmann, 2006). SATPLAN performs an iteration of satisfiability tests on CNF formulas ϕ_b encoding the existence of a parallel plan of length at most b, where b starts from 0 and is increased incrementally. If ϕ_n is the first satisfiable formula, then n equals the length of an optimal parallel plan, and hence SATPLAN is a parallel optimal, or *step-optimal*, planner. In that class of planners, SATPLAN is highly competitive. In particular, SATPLAN won 1st prizes for optimal planners in the International Planning Competition (IPC), namely in IPC 2004 (Kautz, 2004; Hoffmann & Edelkamp, 2005) and in IPC 2006 (Kautz et al., 2006). One property of the CNF encodings employed by SATPLAN that plays a key role in our analysis later on is that these are based on the planning graph structure (Blum & Furst, 1995, 1997).

Of course, our objective closely relates to the many approaches developed in planning for computing lower bounds based on over-approximations, (e.g., Haslum & Geffner, 2000;

Edelkamp, 2001; Haslum, Botea, Helmert, Bonet, & Koenig, 2007; Helmert, Haslum, & Hoffmann, 2007; Katz & Domshlak, 2008; Bonet & Geffner, 2008). The key difference, however, is our focus on *exact* lower bounds, that is, the attempt to actually prove optimality within the abstraction. If we want to be able to prove optimality within the abstraction, then the abstraction must be what we term *solution length preserving*: the abstraction must not introduce any solutions shorter than the optimal solution for the original problem. If the lower bound for step-optimal planning is not exact, then there will be no support for SATPLAN's iteration $n-1$, which constitutes the main bottleneck.

In what follows, we briefly explain our initial motivation behind this work, and we summarize our empirical and theoretical results.

1.1 Initial Motivation

It would of course be interesting to explore whether predicate abstraction can be applied to planning. Indeed, that had been our initial idea. However, while our discussions of this idea lead to nowhere,[1] we instead made a different discovery. Planning state spaces can often be dramatically reduced, without introducing any shorter solutions, based on an abstraction technique that we call *variable domain abstraction*. That technique essentially adapts the work by Hernadvölgyi and Holte (1999) to the propositional STRIPS formalism. We abstract the STRIPS task by not distinguishing between certain values of the multiple-valued variables underlying the STRIPS encoding. That is, if p and q are propositions corresponding to non-distinguished values, the abstracted STRIPS task acts as if p and q were the same. Note that this generalizes the abstraction used by Edelkamp (2001) which, for each multi-valued variable, either abstracts it away completely or does not abstract it at all; see the details in Section 2.

The first example where we noticed the "compression power" of variable domain abstraction is the classical Logistics domain. In this domain, packages must be transported within cities using trucks and between cities using airplanes. Actions load/unload packages, and move vehicles. Importantly, there are no constraints on (either of) vehicle capacities, fuel, or travel links. As a consequence, if a package p starts off in city A and has its destination in city B, then all other cities $C \neq A,B$ are completely irrelevant to p. That is, one can choose an arbitrary location x in such a city C, and replace *all* facts of the form *at(p,l)*, where l is a location outside A and B, with *at(p,x)*. Also, *in(p,t)*, where t is a truck outside A and B, can be replaced with *at(p,x)*. One can completely abstract away the positions of packages that have no destination, and some other minor optimizations are possible. This way we lose many distinctions between different positions of objects—*without introducing a shorter solution!* An optimal plan will not rely on storing a package p in a city other than p's origin or destination. The state space reduction is dramatic: the abstracted state space contains at least $(((C-2)*S)-1)^P$ states less, where C, S, and P respectively are the number of cities, the city size (number of locations in each city), and the number of packages. Similar

1. We are still skeptical about the prospects. Software artifacts (rigid control structure, numeric expressions essential for the flow of control) have a rather different nature than planning problems (loose control structure, numeric expressions non-existent or mostly used to encode resource consumption). For example, a major advantage of predicate abstraction is that it can capture loop invariants—a great feature, but seemingly rather irrelevant to plan generation.

abstractions can be made, and similar state space reductions can be obtained, in other IPC domains such as Zenotravel, Blocksworld, Depots, Satellite, and Rovers (see Section 3).

1.2 Summary of Empirical Results

In our first experiment, we have implemented our Logistics-specific abstraction by abstracting a set of planning tasks at the level of their description, modifying their actions and their initial state. All these planning tasks feature 2 airplanes, 2 locations in each city, and 6 packages. The number of cities scales from 1 to 14. To account for the variance in the hardness of individual instances, we took average values over 5 random instances for each problem size. The increasing number of cities introduces an increasing amount of irrelevance, which we measure by the percentage *RelFrac* of facts considered relevant (not abstracted). Note here that the additional cities are irrelevant *only for some of the individual packages*—they can't be removed completely from the task like standard irrelevance detection mechanisms, e.g. RIFO (Nebel, Dimopoulos, & Koehler, 1997), would try to do.

We provided all the abstracted tasks to three optimizing planners, namely Mips.BDD (Edelkamp & Helmert, 1999), IPP (Hoffmann & Nebel, 2001), and SATPLAN'04,[2] in order to examine how the abstraction affects different approaches to optimizing planning. Mips.BDD searches blindly while exploiting a sophisticated symbolic representation of the state space. IPP is equivalent to a parallel state-space heuristic search with the widely used h^2 heuristic—the parallel version of h^2 as originally introduced in Graphplan (Blum & Furst, 1997; Haslum & Geffner, 2000). Thus, Mips.BDD, IPP, and SATPLAN'04 represent orthogonal approaches to optimizing planning.[3] For each abstract task and planner, we measured runtime, and compared the latter to the time taken by the same planner on the original task. Time-out was set to 1800 seconds, which was also used as the value for the average computation if a time-out occurred. We stopped evaluating a planner if it had 2 time-outs within the 5 instances of one size.

Figure 1(a), (b), and (c) respectively show our results for Mips.BDD, IPP, and SATPLAN'04. Comparing the performance on the original and abstracted tasks, it is apparent from Figure 1 that proving optimality within the abstraction dramatically improved the performance of Mips.BDD, and significantly improved the performance of IPP. At the right end of the scale (with 14 cities), Mips.BDD using the abstraction can find optimal sequential plans almost as fast as SATPLAN'04 can find step-optimal plans. Given that it is usually much harder to find optimal sequential plans than optimal parallel plans, especially in highly parallel domains such as Logistics, this performance improvement is quite remarkable. (In addition to the reduced state space size, Mips.BDD benefits from the small state *encoding*, which stops growing at some point because the maximal number of locations relevant to each package is constant.)

The above findings for Mips.BDD and IPP were in line with our original intuition and, for SATPLAN'04 as well, we expected to see much improved runtime behavior within the abstraction. To our surprise, *we did not.* As appears in Figure 1(c), the improvement obtained for SATPLAN'04 by proving optimality within the abstraction was hardly dis-

2. SATPLAN'04 is the version of SATPLAN that competed in the 2004 International Planning Competition.
3. Importantly, Mips.BDD is sequentially optimal while SATPLAN'04 and IPP are step-optimal. Hence one should not compare the performance of those planners directly, and in particular this is not our purpose here. We focus on how each of the planners reacts to the abstraction.

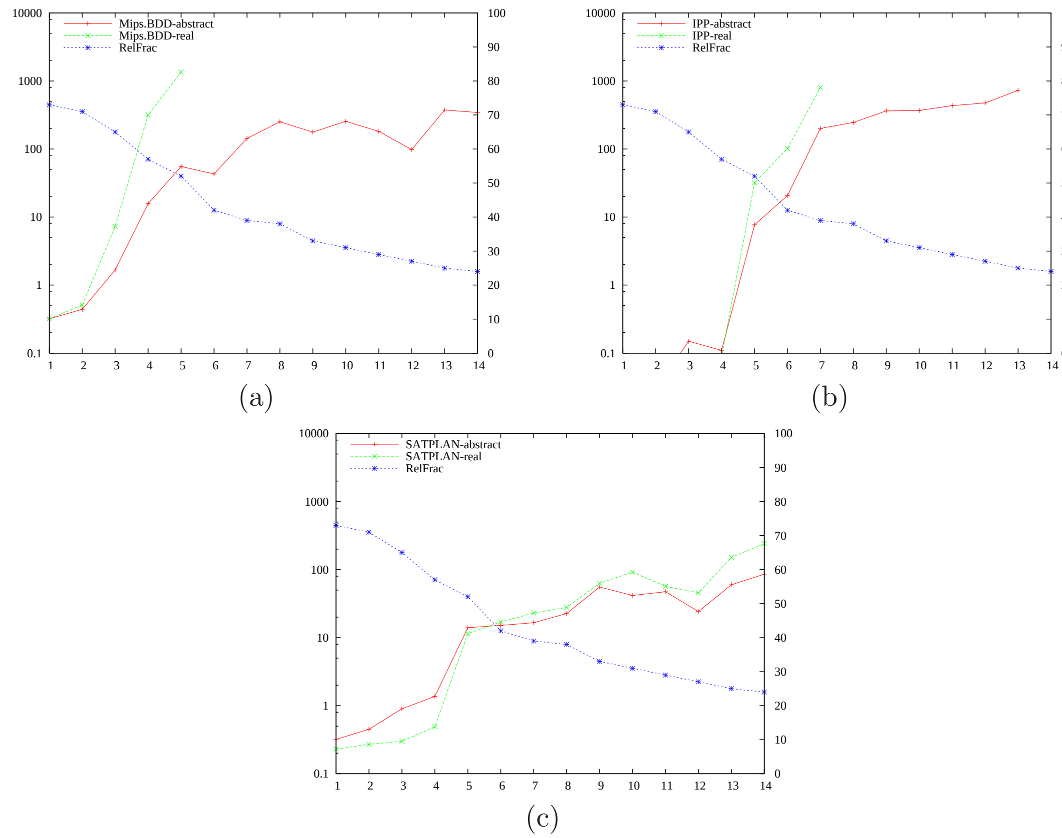

Figure 1: Runtime performance of (a) MIPS, (b) IPP, and (c) SATPLAN'04, with ("abstract") and without ("real") our hand-made variable domain abstraction, in Logistics instances explicitly scaled to increase the amount of irrelevance. Horizontal axis scales the number of cities, left vertical axis shows total runtime in seconds, right vertical axis shows the percentage *RelFrac* of relevant facts.

cernible. At the right end of the scale, abstraction yields a humble speed-up factor of 2.8. That is particularly insignificant since this speed-up is obtained at the drastically small RelFrac value of 24%—in the IPC 2000 Logistics benchmarks, RelFrac is 42% on average. The latter corresponds to 6 cities in Figure 1, where SATPLAN'04 has a slight advantage on the *original* tasks.

To investigate the above more broadly, we conducted experiments in almost all STRIPS domains used in all international planning competitions (IPC) up to IPC 2004. In many cases, we tailored the abstraction of the domain by hand. The results of this exhaustive evaluation (discussed in detail in Section 3) do not significantly depart from those for the Logistics-specific abstraction above. For Mips.BDD we almost consistently obtained a significant improvement. For IPP the improvements happened only rarely, and typically these were not substantial. (While IPP *is* improved in Figure 1, note that, at the IPC-average RelFrac of 42%, the improvement is not yet strong.) Finally, for SATPLAN'04, we hardly ever obtained an improvement.

What causes the difference in "profiting from abstraction" between these three different planning techniques? An intuitive interpretation of our results is that the informedness of the abstraction must compete with the informedness of the search itself. In other words, the better the planner is at exploiting the structure of a particular example, the more difficult it is for the abstraction to exploit structure that is not already being exploited. This intuition is in good correspondence with the Logistics results in Figure 1: while optimizing exactly the same measure, on the original examples, SATPLAN'04 is faster than IPP, while the inverse relation holds regarding which planner profits more from the abstraction. That said, such intuitive interpretations of our results are, at this point, mere speculation. It is left open for future research to determine more accurately what precisely causes the difference. Herein, we concentrate only on planning as satisfiability, and identify a fundamental weakness of this approach with respect to "profiting from abstraction".

1.3 Summary of Theoretical Results

Intrigued by our results for SATPLAN'04,[4] we wondered what kind of effect an abstraction actually has on a "CNF encoding" of the planning task formulated as a Boolean satisfiability problem instance. Recall here that our abstractions are over-approximations, that is, any action sequence applicable in the original task is applicable in the abstract task, and any plan in the original task is a plan in the abstract task. So, intuitively, the abstract task is "more generous" than the original task. With this in mind, consider the CNF formula ϕ_{n-1} encoding the existence of a plan one step shorter than the optimal plan, and consider the same formula, ϕ^{σ}_{n-1}, generated for the abstract task. We need to prove that ϕ^{σ}_{n-1} is unsatisfiable. (Note that ϕ^{σ}_{n-1} is, in fact, unsatisfiable when σ is a solution length preserving abstraction.) Intuitively, the more constrained a formula is, the easier it is to lead such a proof. But ϕ^{σ}_{n-1} is "more generous", and hence less constrained, than ϕ_{n-1}. Does this mean that it is actually *harder* to refute ϕ^{σ}_{n-1} than to refute ϕ_{n-1}?

For some abstraction methods, it is in fact trivial to see that the answer to that question is "yes". Say we abstract ϕ_{n-1} by ignoring some of its clauses. ϕ^{σ}_{n-1} is then a sub-formula of ϕ_{n-1}, immediately implying that any resolution refutation for ϕ^{σ}_{n-1} is also a resolution refutation for ϕ_{n-1}. In particular, the shortest possible refutation cannot be shorter for ϕ^{σ}_{n-1}. A similar situation sometimes occurs in the interplay between abstractions and CNF encodings of planning problems. For instance, suppose we abstract by ignoring a subset of the goals. In most CNF encodings of planning, and in particular in the planning-graph based CNF encodings (Kautz & Selman, 1999) underlying SATPLAN, each goal fact yields one clause in the CNF. Hence, with a "goal ignoring" abstraction, ϕ^{σ}_{n-1} is a sub-formula of ϕ_{n-1}, just as above.

A more complex example would correspond to abstraction by ignoring preconditions or delete effects. In the encodings used by SATPLAN, one or several clauses related to the ignored precondition/delete effect disappear. However, the CNF changes also in other ways because, with one precondition/delete effect less, more actions and facts become possible at later time steps. Intuitively, those additional actions and facts do not help proving

4. Translation: "Deeply frustrated by our results for SATPLAN'04, . . . "

unsatisfiability of ϕ^{σ}_{n-1}. The formal proof of this intuitive statement, however, is less obvious than the one for the "goal ignoring" abstraction above. Matters are most complicated, and much less intuitive, for Edelkamp's (2001) abstraction and for variable domain abstraction, where the changes made to the abstract task also affect the *add* effects of actions. Recall that variable domain abstraction is of special interest here because it is most likely to satisfy the constraint of solution length preservation.

To investigate these issues in detail, one has to consider various possible combinations of CNF encodings and abstraction methods. Many different encodings of planning into SAT have been proposed. We focus on planning-graph based encodings because those were used in the SATPLAN system, in all of its appearances in the international planning competitions (Kautz & Selman, 1999; Long, Kautz, Selman, Bonet, Geffner, Koehler, Brenner, Hoffmann, Rittinger, Anderson, Weld, Smith, & Fox, 2000; Kautz, 2004; Kautz et al., 2006). Indeed, according to Kautz and Selman (Kautz & Selman, 1999; Long et al., 2000), such CNF encodings—in particular the mutex relations computed by Graphplan—are vital to SATPLAN's performance. While recent results on more effective encodings may challenge this assessment (Rintanen, Heljanko, & Niemelä, 2006; Chen, Huang, Xing, & Zhang, 2009; Robinson, Gretton, Pham, & Sattar, 2008), even so the graphplan-based encodings are of interest simply because they have been widely used during almost a decade. It remains of course an important question whether and to what extent our results carry over to alternative CNF encodings. We discuss this issue in some depth when concluding the paper in Section 5.

We consider four different encodings, three of which have been used in some edition of the IPC; the fourth encoding is considered for the sake of completeness. The encodings differ in two parameters: whether they use only action variables, or action as well as fact variables; and whether they include all planning graph mutexes between actions or only the direct interferences. (The latter is motivated by the fact that there is often an enormous amount of action mutexes, seriously blowing up the size of the formula.)

On the abstractions side, we focus on abstraction methods that can be formulated as manipulating planning tasks at the language level, i.e., modifying the task's actions and/or initial/goal states. Many commonly used abstractions for propositional STRIPS can be formulated this way. We consider six such abstractions, namely (1) removing goals, (2) adding initial facts, (3) removing preconditions, (4) removing delete effects, (5) Edelkamp's (2001) abstraction (removing entire facts), and (6) variable domain abstraction.

For all 24 combinations of CNF encoding and abstraction method, we prove that the shortest possible resolution refutation can be exponentially longer for ϕ^{σ}_{n-1} than for ϕ_{n-1}. For all 20 combinations involving abstractions other than variable domain abstraction, we prove that the shortest possible resolution refutation cannot be shorter for ϕ^{σ}_{n-1} than for ϕ_{n-1}. For abstraction (1), this is trivial as outlined above. For abstractions (2)–(4), the proof exploits the fact that these abstractions lead to "larger" planning graphs containing more actions and facts. For abstraction (5), this reasoning does not work because some facts disappear from the planning graph. However, one can start by removing a fact from the goal and all action preconditions; afterwards the fact is irrelevant and one can remove it also from the initial state and action effects.

Matters are most complicated for abstraction (6), that is, variable domain abstraction. For the encoding with only action variables and full mutexes, we show that, as before, the shortest possible resolution refutation cannot be shorter for ϕ^{σ}_{n-1} than for ϕ_{n-1}. For the encoding with only action variables and only direct action mutexes, we show that the possible improvement is bounded from above by the effort it takes to recover the indirect action mutexes. For the two encodings with both action and fact variables, it remains an open question whether such bounds exist.[5]

Importantly, all our proofs are valid not only for general resolution, but also for many of the known restricted variants of resolution, in particular for the tree-like resolution refutations generated by DPLL (Davis & Putnam, 1960; Davis, Logemann, & Loveland, 1962). Naturally, our proofs are not separate for each of the combinations, but rather exploit and exhibit some of their common features.

The practical significance of our theoretical results is, to some extent, debatable as there is no direct connection between best-case resolution refutation size and empirical SAT solver performance. Even a very large refutation may be easy to find if it mostly consists of unit resolutions. Vice versa, just because a small refutation exists, that does not mean the SAT solver will find it. This notwithstanding, it appears unlikely that best-case resolution refutation size and practical SAT solver performance are completely unrelated (beyond the obvious lower bound). One example that indicates the opposite are planning graph mutexes. Mutexes do reduce the best-case refutation size by doing some of the work before the resolution is even invoked.[6] In other words, SAT solvers can exploit the mutexes to prune their search trees more effectively. We are not aware of explicit empirical proof that this tends to happen often, but there seems to be little doubt that it does. That is also suggested explicitly by Kautz and Selman (Kautz & Selman, 1999; Long et al., 2000) by ways of explaining the improved performacce of their system when run on graphplan-based encodings.

An interesting situation arises in (all) our experiments. We use variable domain abstraction on the encoding with only action variables and only direct action mutexes (as employed in SATPLAN's IPC'04 version). In this setting, resolution refutations can get shorter in principle, although only by the effort it takes to recover the indirect action mutexes. Further, we employ some trivial post-abstraction simplification methods (such as removing action duplicates) which, as we show, also have the potential to shorten resolution refutations. Still, as reported above, there is no discernible empirical improvement. The reason might be that the SAT solver does not find the shorter refutations, or that such shorter refutations do not actually appear on a significant scale. There is some evidence indicating the latter: mutex recovery becomes necessary only in rather special situations, where the abstraction turns an indirect mutex into a direct one. This will typically concern only a small fraction of the indirect mutexes. In addition, for both mutex recovery and simplifications, in a well-designed variable domain abstraction the affected actions will typically be irrelevant anyway. For example, with our hand-made Logistics abstraction,

5. The reason for complications is that answering this question requires determining, for planning-graph based encodings in general, whether fact variables are only syntactic sugar or may lead to more succinct refutations. Such a proof appears quite challenging; we say some more on this in Section 4.

6. General resolution can recover the mutexes effectively, c.f. the related investigations by Brafman (2001) and Rintanen (2008). It does not seem likely that the same is the case for tree-like resolution, but to the best of our knowledge this is not yet known.

the effect of the potential improvements is limited to actions appearing only in redundant plans. We get back to this in detail in Section 4.

In our view, the theoretical results would be of potential importance even with no evidence of empirical relevance, simply because they are quite surprising. After a moment of thought, it is clear that resolution refutation does not become easier by ignoring goals. However, variable domain abstraction in domains such as Logistics deflates state spaces immensely, up to a point where they have only a tiny fraction of their original size. Before performing this work, we would never have expected the best-case refutation size to remain the same.

The paper is organized as follows. Section 2 discusses preliminaries, covering the employed notions of planning, planning graphs, propositional encodings, resolution, and abstraction methods; in particular, it formally defines variable domain abstraction. Section 3 summarizes our empirical results. Section 4 presents our results regarding resolution refutations in abstract CNF encodings. Related work is discussed during the text where appropriate. We conclude in Section 5. Appendix A contains most proofs, which are replaced with brief proof sketches in the main body of the text. Additional empirical data can be found in an online appendix (see JAIR web page for this article).

2. Preliminaries

We begin with a discussion of various concepts needed in the rest of the paper: propositional STRIPS planning, planning graphs, propositional CNF encodings of planning problems, resolution proofs of unsatisfiability, and abstraction methods used in planning. As a general rule of notation, we will use variants of: $\mathcal{P}$ for planning tasks; F, A, and G for sets of facts, actions, and goals, respectively; σ for abstractions; PG for planning graphs; and ϕ for propositional formulas and encodings.

2.1 STRIPS and Planning Graphs

Classical planning is devoted to goal reachability analysis in state transition models with deterministic actions and complete information. Such a model is a tuple $\mathcal{M} = \langle S, s_0, S_G, A, \delta \rangle$ where S is a finite set of states, $s_0 \in S$ is the initial state, and $S_G \subseteq S$ is a set of alternative goal states, A is a finite set of actions, and $\delta : S \times A \to S$ is a transition function, with $\delta(s, a)$ specifying the state obtained by applying a in s. A solution, that is a *plan*, for a state transition model is a sequence of actions $a_1, \dots, a_m$ from A that generate a sequence of states $s_0, \dots, s_m$ such that, for $0 \leq i < m$, $\delta(s_i, a_{i+1}) = s_{i+1}$, and $s_m \in S_G$.

While AI planning targets large-scale state transition models with huge numbers of states, these models are assumed to be described in a concise manner via some intuitive declarative language. Here we use a propositional fragment of the STRIPS language (Fikes & Nilsson, 1971). For brevity, we will refer to this fragment as *STRIPS* herein. Informally, a *planning task* or *planning instance* in STRIPS consists of a set of propositional facts, some of which hold initially and some of which must hold simultaneously at the end of the plan execution. The state of the system at any time is defined by the set of propositional facts that hold at that time. The task specifies a set of actions, each of which is defined by a set of precondition facts, a set of facts that are added to the state, and a set of facts that are removed from the state, if the action is taken. Formally, a STRIPS planning task is given by a quadruple $\mathcal{P} = (P, A, I, G)$ with *fact* set P, *initial state* description $I \subseteq P$, *goal* description $G \subseteq P$, and *action* set A where for every action $a \in A$ we have $pre(a)$, $add(a)$, and $del(a)$, each of which is a subset of P. Such a planning task defines a state transition model $\mathcal{M} = \langle S, s_0, S_G, A, \delta \rangle$ with the state space $S = 2^P$, the initial state $s_0 = I$, and for each $s \in S$, we have $s \in S_G$ iff $G \subseteq s$. For each $s \in S$, $A(s) = \{a \in A \mid pre(a) \subseteq s\}$ are the actions applicable in s, and for each $a \in A(s)$, we have $\delta(s, a) = (s \setminus del(a)) \cup add(a)$. Here we assume that the actions are reasonable in the sense that $add(a) \cap del(a) = \emptyset$. This is satisfied in most known planning benchmarks; in particular it is satisfied in all benchmarks used in our experiments.[7]

Many planning algorithms, including SATPLAN, employ some form of approximate reachability analysis. One of the primary tools for this purpose is the *planning graph*, first introduced in the scope of the Graphplan planner (Blum & Furst, 1997). For a length bound b, the planning graph $PG(\mathcal{P})$ associated with $\mathcal{P}$ is a layered graph with two kinds of nodes: fact nodes and action nodes. The layers alternate between fact layers $F(0), F(1), \dots, F(b)$, and action layers $A(0), A(1), \dots, A(b-1)$, with each pair of layers $F(t), A(t)$ forming a "time step" t. The first vertex layer $F(0)$ contains the initial state. $A(t)$ and $F(t+1)$ for $0 \leq t < b$ are the action sets and fact sets, respectively, available at time step t and $t+1$. More precisely, each $A(t)$ includes all actions $a \in A$ where $pre(a) \subseteq F(t)$ and no pair of facts $p, p' \in pre(a)$ is mutex in layer t (c.f. below); further, $A(t)$ contains the standard NOOP action for every fact in $F(t)$.[8] Each $F(t+1)$ contains the union of the add effects of all actions in $A(t)$. Obviously, we have $A(t) \subseteq A(t+1)$ and $F(t) \subseteq F(t+1)$. The goal facts G label appropriate vertices in $F(b)$. $PG(\mathcal{P})$ has four kinds of edges:

(1) $E_{pre}(t) \subseteq F(t) \times A(t)$ connect the actions in $A(t)$ with their preconditions in $F(t)$,

(2) $E_{add}(t) \subseteq A(t) \times F(t+1)$ connect the actions in $A(t)$ with their add effects in $F(t+1)$,

(3) $E_{a\text{-}mutex}(t) \subseteq A(t) \times A(t)$ capture a pair-wise mutual exclusion relation between actions in $A(t)$; if $(a(t), a'(t)) \in E_{a\text{-}mutex}(t)$, then actions a and a' cannot be applied simultaneously at time t,

(4) $E_{f\text{-}mutex}(t) \subseteq F(t) \times F(t)$ capture a pair-wise mutual exclusion relation between facts in $F(t)$; if $(f(t), f'(t)) \in E_{f\text{-}mutex}(t)$, then facts f and f' cannot hold together at time t.

Note that $PG(\mathcal{P})$ does not have explicit edges for the deletion effects of actions; these effects are captured in the mutual exclusion relation (e.g., if $p \in add(a_1) \cap del(a_2)$, then $(a_1, a_2) \in E_{a\text{-}mutex}$ at all times). The mutex edges $E_{mutex} = E_{a\text{-}mutex} \cup E_{f\text{-}mutex}$ are computed by an iterative calculation of interfering action and fact pairs (Blum & Furst, 1997). Namely, two actions (directly) *interfere* if the effects of one contradict the effects of the other, or if one deletes a precondition of the other. Two actions have *competing*

7. In the IPC-2002 domain Rovers, some operators add and delete the same artificial fact in order to prevent their parallel application. We implement this restriction via duplicating the respective operators and sequentializing original and duplicate via *two* artificial facts. Similar fixes have been implemented in a couple of other domains as well.

8. For a fact $p \in P$, the associated NOOP(p) has no delete effects, and has $\{p\}$ as both its preconditions and add "effects". These are dummy actions that simply propagate facts from one fact layer to the next.

needs if they have mutex preconditions. Combining these two scenarios together, we say that two actions are *mutex* if they either (directly) interfere or have competing needs. In a similar spirit, two facts are *mutex* if there is no non-mutex pair of actions (in the graph layer directly below) together achieving both facts. A variant that will be of interest is the planning graph in which, after the iterative computation, $E_{a\text{-}mutex}$ is reduced to contain only the directly interfering actions. We will call this the *reduced planning graph*, and denote it with $PG_{red}(\mathcal{P})$. The motivation for considering this is that, often, the reduced planning graph results in much smaller SAT encodings; we get back to this below.

2.2 Propositional Encodings

We consider three CNF encodings used by (one or the other version of) SATPLAN, as well as a fourth encoding that fits naturally into the picture. Each of the encodings takes as input a planning task $\mathcal{P}$ with length bound b, and creates a formula in the standard Conjunctive Normal Form (CNF). The CNF formula is then solved by an off-the-shelf SAT solver. This process constitutes the basic step in a SAT-based approach to planning as implemented in SATPLAN (Kautz & Selman, 1992, 1996, 1999), where one starts with $b = 0$ and iteratively increments b until the CNF becomes satisfiable for the first time. The plan corresponding to the satisfying assignment is then a plan with minimal b, and is hence optimal in that sense.[9]

A CNF formula ϕ is logically a conjunction (AND) of clauses, where a clause is a disjunction (OR) of literals, and a literal is a propositional (Boolean) variable or its negation. CNF formulas are often written as a set of clauses, and each clause written as a set of literals, the underlying logical conjunction and disjunction, respectively, being implicit. Our propositional encodings of bounded-length planning tasks are specified in terms of various kinds of clauses generated by the encoding method.

Encoding (A) is constructed from $PG(\mathcal{P})$ and uses the propositional "action" variables $\{a(t) \mid 0 \leq t < b, a \in A(t)\}$. For each goal fact g there is a *goal clause* of the form $\{a_1(b-1), \dots, a_l(b-1)\}$, where $a_1, \dots a_l$ are the actions in $A(b-1)$ that add g. Similarly, for every $a(t)$ with $t > 0$ and every $p \in pre(a)$ we have a *precondition clause* $\{\neg a(t), a_1(t-1), \dots, a_l(t-1)\}$, where $a_1, \dots a_l$ are the actions in $A(t-1)$ that add p. Finally, we have a *mutex clause* $\{\neg a(t), \neg a'(t)\}$ for every t and $(a, a') \in E_{a\text{-}mutex}(t)$. (Note here that the dependence on the initial state is taken into account already in terms of which actions are contained in the sets $A(t)$, and does not need to be stated explicitly in the CNF.)

Encoding (B) is similar to (A) except that it uses variables (and appropriate clauses) also for the facts. More specifically, in addition to the action variables, it has "fact" variables $\{f(t) \mid 0 \leq t \leq b, f \in F(t)\}$. For each goal fact g, the *goal clause* is simply a unit clause asserting $g(b)$. For $t > 0$ and each fact $f(t)$, we now have an *effect clause* of the form $\{\neg f(t), a_1(t-1), \dots, a_l(t-1)\}$, where $a_1, \dots a_l$ are the actions in $A(t-1)$ that add f. For every $a(t)$ and every $p \in pre(a)$ we have a *precondition clause*, which

9. While all versions of SATPLAN use this naive incremental update on b, it has been shown that there are more clever strategies, exploiting the typical distribution of runtime over different values of b (Rintanen, 2004; Streeter & Smith, 2007).

takes the form $\{\neg a(t), p(t)\}$. We have *action mutex clauses* $\{\neg a(t), \neg a'(t)\}$ for every t and $(a, a') \in E_{a\text{-}mutex}(t)$, and *fact mutex clauses* $\{\neg f(t), \neg f'(t)\}$ for every t and $(f, f') \in E_{f\text{-}mutex}(t)$. Finally, for each fact $f \in F(0)$, we have an *initial state clause* $\{f(0)\}$ (these are not strictly necessary but are implemented in SATPLAN which is why we include them here).

Encoding (C) is like (A) except that it is based on the reduced planning graph $PG_{red}(\mathcal{P})$, so that mutex clauses are present only for action pairs whose preconditions and effects interfere directly.

Encoding (D) is like (B) except that, as in (C), it is based on $PG_{red}(\mathcal{P})$, with mutex clauses only for action pairs whose preconditions and effects interfere directly. Note, however, that the fact mutexes are those of the full planning graph, $PG(\mathcal{P})$.

All these encodings are reasonable ways of turning a planning graph into a CNF formula. The encodings essentially underly the competition implementations of SATPLAN. We will detail this below. First, note that the different encodings have different benefits and drawbacks. First, observe that the encodings are characterized by two decisions: (1) *Should we include all action mutexes from Graphplan, or only the direct interferences?* (2) *Should we include only action variables, or both action and fact variables?* Regarding (1), the empirical observation that "mutexes help" was one of the major observations in the design of SATPLAN (then called Blackbox) (Kautz & Selman, 1999; Long et al., 2000), in particular in comparison to earlier encoding methods (Kautz, McAllester, & Selman, 1996). On the other hand, since mutexes talk about *pairs* of facts and actions, the encodings may become quite large—there will be one clause for every pair of mutex actions or mutex facts. This is particularly critical for actions, of which in many planning benchmarks there are thousands (compared to at most a few hundred facts). Indeed, it turns out that action mutexes often consume critically large amounts of memory. It is not uncommon to have CNF formulas with millions of clauses, most of which are action mutexes (Kautz & Selman, 1999; Kautz, 2004; Kautz et al., 2006). This motivates encodings (C) and (D). As for question (2), this does not make as much of a difference, empirically, in most planning benchmarks. We consider this distinction only because it was used in some versions of SATPLAN.

Let us say a few words to clarify exactly how encodings (A)–(D) relate to the SATPLAN literature and implementations. Due to the long history of SATPLAN, as well as a few imprecisions in the literature, this is a little complicated. Our foremost reference is the actual program code underlying SATPLAN'04 and SATPLAN'06, i.e., the most recent versions used in the 2004 and 2006 competitions. The encoding methods in these versions were implemented by one of the authors of this paper. There are four different encoding methods: *action-based*, *graphplan-based*, *skinny action-based*, and *skinny graphplan-based*. The action-based encoding is exactly (A), the graphplan-based encoding is exactly (B), and the skinny graphplan-based encoding is exactly (D).[10] The skinny action-based encoding is like (C) except that, to save some runtime, the planning graph implementation does not propagate mutexes (after all, only direct interferences are present in the final encoding), effectively computing a relaxed planning graph (Hoffmann & Nebel, 2001). We use a normal

10. In the 2004 version, the skinny graphplan-based encoding did not feature fact mutexes. This is of no consequence because that encoding was not used in the 2004 competition.

planning graph for (C) only for the sake of readability—the greater similarity to the other encodings significantly simplifies the write-up, and our theoretical results hold as stated also for relaxed planning graphs.

How did the SATPLAN encodings develop historically, how is this reflected in the literature, and which encodings were used in the competitions? We answer these questions to the extent necessary for explaining encodings (A)–(D). The original paper on SATPLAN (Kautz & Selman, 1992) introduced rather different encodings. Graphplan-based encodings with only direct action mutexes were introduced next, and observed to yield performance comparable to Graphplan itself (Kautz & Selman, 1996). Subsequently, it was observed that modern SAT solvers profit from full (fact and action) mutex relations and actually beat other planners, in several domains (Kautz & Selman, 1999).[11] Consequently, such a graphlan-based encoding, i.e., our encoding (B), was used in the 1998 and 2000 competitions (Long et al., 2000). Prior to the 2004 competition, the encoding methods were re-implemented, yielding the four methods explained above. The IPC'04 booklet paper on SATPLAN'04 (Kautz, 2004) describes these four encodings.[12] The version run in the competition is the skinny action-based encoding that is (for our theoretical results) equivalent to encoding (C). When running the planner in IPC 2006, it turned out that having full fact mutexes helped in some domains, and so encoding (D) was used instead (Kautz et al., 2006). We consider encoding (A) for the sake of completeness.

2.3 Resolution Refutations

Our theoretical results are with respect to the *resolution* proof system (Robinson, 1965), which forms the basis of most of the complete SAT solvers around today (cf. Beame, Kautz, & Sabharwal, 2004). It is a sound and complete proof system, and has been studied extensively for theoretical and practical reasons. It works on CNF formulas and has only one simple rule of inference: given clauses $\{A, x\}$ and $\{B, \neg x\}$, one can *derive* the clause $\{A, B\}$ by *resolving upon* the variable x. Here A and B are shorthands for arbitrary lists of literals. Note that the choice of clauses to resolve is arbitrary, as long as they share a variable, with opposite signs. A *resolution derivation* π of a clause C from a formula ϕ consists of a series of applications of the resolution rule starting from the clauses in ϕ such that one eventually derives C; when C is the (unsatisfiable) empty clause, $\{\}$, π is called a *resolution proof* (of unsatisfiability) or *refutation* of ϕ. The *size* of π is the number of applications of the resolution rule in π. When ϕ is unsatisfiable, $\mathcal{RC}(\phi)$ denotes the *resolution complexity* of ϕ, i.e., the size of the smallest resolution proof of unsatisfiability of ϕ. We will be interested in whether applying an abstraction to a planning task can convert its encoding into one that has smaller resolution complexity.

A commonly studied sub-class of (still sound and complete) resolution derivations is that of *tree-like resolution* derivations, where each derived clause is used at most *once* in the whole derivation; the underlying graph structure of the proof is then a tree. Other

11. Kautz and Selman (1999) cite graphplan-based encodings from their earlier work (which used only action mutexes). However, the Blackbox program code includes functions that generate full mutexes, and Kautz and Selman explicitly emphasize the importance of these mutexes.
12. The paper is only an abstract and is a little imprecise in this description: initial state, goal, and fact mutex clauses are not mentioned; the skinny action-based encoding is stated to be identical to our encoding (C), i.e., based on a full planning graph rather than a relaxed planning graph.

interesting sub-classes of resolution include *regular resolution*, which is provably exponentially more powerful than tree-like resolution and in which each variable is resolved upon at most once in any path from the root to a leaf in the underlying proof graph, and *ordered resolution*, where in addition variables respect a fixed ordering in any root-to-leaf path. Tree-like resolution captures proofs of unsatisfiability generated by SAT solvers that are based on the DPLL procedure (Davis & Putnam, 1960; Davis et al., 1962) but don't employ so-called "clause learning" techniques; the latter kind of SAT solvers can be provably exponentially more powerful than even regular resolution although still within the realm of general resolution (Beame et al., 2004).

We note that the arguments presented in this paper are for general (unrestricted) resolution. However, since most of aour constructions do not affect or rely on the structural properties of the resolution refutations under consideration, the results hold as stated (except for a slight weakening in the case of Lemma 4.14) for all known variants of resolution for which setting variables to True or False or replacing one variable with another preserves proof structure. These variants include tree-like (DPLL), regular, and ordered resolution.

We state a standard property of resolution proofs which we will use in our arguments, pointing out that certain modifications (such as "variable restrictions" and "shortening of clauses") of a given formula do not cause proofs to become longer with general resolution or any of its "natural" sub-classes, including those mentioned above. Let x be a variable of ϕ and y be True, False, or another (possibly negated) variable of ϕ. The *variable restriction* $x \leftarrow y$ on ϕ is a transformation that replaces x with y throughout ϕ, and simplifies the resulting formula by removing all clauses containing True or a variable and its negation, and by removing False or duplicate literals from clauses. In other words, a variable restriction involves fixing the value of a variable or identifying it with another literal, and simplifying the formula. If τ is a sequence of variable restrictions on ϕ, then by $\phi|_\tau$ we denote the outcome of applying τ to ϕ.

The following proposition combines two basic facts together in a form that will be useful for us: (1) variable restrictions cannot increase the resolution complexity of a formula, and (2) lengthening clauses and/or removing clauses cannot decrease the resolution complexity of a formula.

Proposition 2.1. *Let ϕ^σ and ϕ be CNF formulas. If there exists a sequence τ of variable restrictions on ϕ^σ such that every clause of $\phi^\sigma|_\tau$ contains as a sub-clause a clause of ϕ, then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.*

Some explanation of this proposition and how we will use it is in order. The notation used here is chosen to match the way we will eventually utilize this proposition in our proofs; more on this below. The conditions in the proposition imply that one may obtain ϕ from ϕ^σ by applying the restriction τ, possibly throwing away some literals from some clauses, and possibly adding new clauses. Intuitively, each of these three modifications to ϕ^σ can only reduce the number of solutions and cannot make it any harder to prove the formula unsatisfiable. This property of resolution refutations of propositional formulas has been previously used (at least indirectly) in various contexts. For completeness, we include a proof in Appendix A, based on "folklore" ideas from proof complexity literature. An alternative proof, with a somewhat different notation, may also be found in the appendix of a recent article by Hoffmann, Gomes, and Selman (2007).

The way we will use this proposition is the following: ϕ^σ will be the CNF encoding of an abstracted planning task, ϕ will be the encoding of the original planning task, and τ will be a carefully chosen restriction of ϕ^σ that will bring our focus to variables already appearing in ϕ. Proposition 2.1 will then imply that the original encoding is no harder to refute, using resolution or its natural sub-classes, than the abstracted encoding.

2.4 Abstraction in Planning

Abstraction methods of various kinds have been put to use in planning, often quite successfully. One line of work uses abstraction methods for problem decomposition (cf. Sacerdoti, 1973; Knoblock, 1990; Koehler & Hoffmann, 2000). To the best of our knowledge, our approach—examining the abstract state space in order to prove the absence of solutions—has not been pursued before. The line of work most relevant to ours is the work on domain-independent heuristic functions (cf. McDermott, 1999; Bonet & Geffner, 2001; Hoffmann & Nebel, 2001; Edelkamp, 2001; Haslum et al., 2007; Helmert et al., 2007; Katz & Domshlak, 2008). There, "abstraction" means over-approximation of the state space, as in our work; what differs is how the abstractions are used. Of course, the kinds of over-approximations that are useful for either purpose can differ a lot. To use abstraction as we do in this paper, one has to define over-approximations that preserve, to a very large extent, the real structure of the problem. In particular, our ideal goal is to find abstractions that preserve the length of an optimal solution—something one definitely wouldn't expect from the abstraction underlying a heuristic function, since that has to be solved in every search state.

We briefly review some of the over-approximation methods that have been used in planning so far; we then formally introduce our novel one, variable domain abstraction. We use the Logistics domain as an illustrative example.[13]

One wide-spread over-approximation method in planning is the "2-subset" relaxation underlying the computation made in a planning graph (Blum & Furst, 1997), which is generalized to an "m-subset" relaxation by Haslum and Geffner (2000). In a nutshell, one assumes that achieving a set of facts is only as hard as achieving its hardest m-subset. It is known that, in most domains, including Logistics, the 2-subset solution length (corresponding to the length of a planning graph constructed up to the first fact layer containing no mutexes between the goal facts) is typically strictly lower than the length of an optimal plan. For $m > 2$, on the other hand, computing m-subset solution length is typically too costly, and still, for no $m = o(|P|)$ can one typically guarantee solution length preservation (Helmert & Mattmüller, 2008).

A second wide-spread over-approximation method is "ignoring delete lists" (McDermott, 1999; Bonet & Geffner, 2001). For this approximation, one simply removes (some of) the negative effects of the actions. If all negative effects are removed, then the problem becomes solvable in time linear in the instance size. The latter is the basis of the heuristic functions used in many modern planners (cf. Bonet & Geffner, 2001; Hoffmann & Nebel, 2001; Gerevini, Saetti, & Serina, 2003). Ignoring deletes is very likely to introduce shorter solutions. For example, in the Towers of Hanoi problem, it leads to plans of length n instead of $2^n - 1$ (Hoffmann, 2005). In Logistics, if one ignores the deletes of moving actions then

13. As stated, an open topic is to explore model checking abstractions, in particular predicate abstraction (Graf & Saïdi, 1997; Clarke et al., 2003), instead of planning abstractions.

the plans may get shorter because vehicles never have to "move back" in the abstraction. Interestingly, ignoring the deletes of load/unload does *not* decrease plan length, since these actions never have to be undone in an optimal plan. We use this observation in some of our experiments.

A third abstraction was introduced by Edelkamp (2001) for his "pattern database" heuristic. For this approximation, one completely removes some facts from the problem description, notably facts corresponding to *all* values of some multi-valued variables. If enough facts are removed, the task becomes sufficiently simple. Obviously, this approximation will hardly be solution length preserving. In Logistics, if we remove, for example, a fact *at(package1,airport2)* then, in particular, *package1* can be loaded onto an airplane at *airport2 without actually being there* because that precondition is removed. An optimal plan can now just make the package "pop up" anywhere.

A fourth abstraction, finally, involves removing some preconditions (Sacerdoti, 1973) and/or goal facts. As with Edelkamp's abstraction, this cannot be expected to be solution length preserving in interesting cases.

The above calls for a new abstraction method, which we designed following Hernadvölgyi and Holte (1999). Considering STRIPS-like state transition systems with multiple-valued (instead of Boolean) variables, they propose to reduce variable domains by not distinguishing between certain values. For example, if the content of a cell in the 8-puzzle can be in $\{blank, 1, 2, 3, 4, 6, 7, 8\}$ then that may be replaced with $\{blank, 1, 2, 3\}$ where $\{3, \dots, 8\}$ are all mapped onto 3. Our observation is that, in many planning benchmarks, this can be done without introducing shorter plans. For example, in Logistics it is unnecessary to distinguish the positions of packages in irrelevant cities. Therefore, we can replace the domain of $at(p)$, $\{A_1, A_2, B_1, B_2, C_1, C_2, \dots\}$, where A and B are the initial and goal cities of p and $A_i, B_i, \dots$ are locations in cities $A, B, \dots$, with an abstract domain $\{A_1, A_2, B_1, B_2, C_1\}$. In STRIPS, this amounts to replacing a set of irrelevant facts $at(p, l)$ with the single fact $at(p, C_1)$. We now formalize this idea.

Let *persistently mutex* denote the standard notion that two facts are mutex in the fixpoint layer of a planning graph: typically, different values of a multiple-valued variable.

Definition 2.2. Let $\mathcal{P} = (P, A, I, G)$ be a STRIPS planning task, $p, p' \in P$ a pair of persistently mutex facts such that, for all $a \in A$, we have $(\{p, p'\} \cap del(a)) \subseteq pre(a)$. Then $(\xi(P), \{\xi(a) | a \in A\}, \xi(I), \xi(G))$ is called a **variable domain abstraction** of $\mathcal{P}$, where ξ is defined as follows:

1. For a fact set F, $\xi(F) = F$ if $p' \notin F$; otherwise, $\xi(F) = (F \setminus \{p'\}) \cup \{p\}$.
2. For an action $a = (pre, add, del)$, $\xi(a) = (\xi(pre), \xi(add), \xi(del))$ if $p \notin \xi(add) \cap \xi(del)$; otherwise, $\xi(a) = (\xi(pre), \xi(add), \xi(del) \setminus \{p\})$.

In words, Definition 2.2 simply says that we replace p' with p. If p now appears both in the add list and in the delete list of an action, we remove it from the delete list.[14] This situation will arise, for instance, if the action moves a package from one irrelevant position

14. The reader may wonder why p remains in the add list, although by prerequisite $p \in \xi(pre)$. The reason is that we distinguish between "abstractions" and "simplifications": both change the planning task; abstractions, but not simplifications, do so in a way that may alter the task's semantics. However, simplifications may as well affect resolution complexity. We will get back to this later in the paper.

to another irrelevant position. After the operation, p is equivalent to what was originally $p \vee p'$: p is True after an abstracted action sequence if and only if $p \vee p'$ is True after the corresponding real action sequence. In particular, Proposition 2.3 states that variable domain abstraction is an over-approximation in the usual sense.

Proposition 2.3. *Let $\mathcal{P} = (P, A, I, G)$ be a STRIPS planning task, and let $\sigma(\mathcal{P}) = (\xi(P), \{\xi(a) \mid a \in A\}, \xi(I), \xi(G))$ be a variable domain abstraction of $\mathcal{P}$. Then, whenever $\langle a_1, \ldots, a_n \rangle$ is a plan for $\mathcal{P}$, $\langle \xi(a_1), \ldots, \xi(a_n) \rangle$ is a plan for $\sigma(\mathcal{P})$.*

Proof. Let $\mathcal{M}$ and $\mathcal{M}_\sigma$ be the state models induced by $\mathcal{P}$ and $\sigma(\mathcal{P})$. First, let us show that, for each state $s \in \mathcal{M}$, and each action $a \in \mathcal{M}$, if a is applicable in s, then (i) $\xi(a)$ is applicable in $\xi(s)$, and (ii) $p \in \delta(\xi(s), \xi(a))$ if and only if $\{p, p'\} \cap \delta(s, a) \neq \emptyset$. Note that, since the abstraction σ has no effect on facts other than $\{p, p'\}$, (ii) implies $\delta(\xi(s), \xi(a)) = \xi(\delta(s, a))$. Thus, together, (i) and (ii) imply that (iii) $\mathcal{M}$ is homomorphic to $\mathcal{M}_\sigma$. Finally, it is straightforward from Definition 2.2 that (iv) the initial state in $\mathcal{M}_\sigma$ is $\xi(I)$, and that the goal states in $\mathcal{M}_\sigma$ are exactly $\{\xi(s) \mid s \in S_G\}$. Together, (iii) and (iv) imply the claim of Proposition 2.3.

Let $a = (pre, add, del)$. The applicability of $\xi(a)$ in $\xi(s)$ is straightforward; if $p' \notin pre$, then we have $\xi(pre) = pre$ and $\xi(s) \cap \xi(pre) = s \cap pre$, and otherwise $\xi(pre) = (pre \setminus \{p'\}) \cup \{p\}$ and $\xi(s) = (s \setminus \{p'\}) \cup \{p\}$. In both cases, $pre \subseteq s$ implies $\xi(pre) \subseteq \xi(s)$. Consider now the sub-claim (ii) on a case-by-case basis.

$\underline{\{p', p\} \cap add = \emptyset, \{p', p\} \cap del = \emptyset}$ We have $p \notin add(\xi(a))$ and $p \notin del(\xi(a))$, and thus $p \in \delta(\xi(s), \xi(a))$ iff $p \in \xi(s)$ iff $\{p, p'\} \cap s \neq \emptyset$ iff (see assumption on del in the case) $\{p, p'\} \cap \delta(s, a) \neq \emptyset$.

$\underline{\{p', p\} \cap add \neq \emptyset, \{p', p\} \cap del = \emptyset}$ We have $p \in add(\xi(a))$ and $p \notin del(\xi(a))$, and thus $p \in \delta(\xi(s), \xi(a))$. On the other hand, $\{p, p'\} \cap \delta(s, a) \neq \emptyset$ also trivially holds here.

$\underline{\{p', p\} \cap add = \emptyset, \{p', p\} \cap del \neq \emptyset}$ We have $p \notin add(\xi(a))$ and $p \in del(\xi(a))$, and thus $p \notin \delta(\xi(s), \xi(a))$. On the other hand, since p, p' are persistently mutex facts in $\mathcal{P}$, and $\{p', p\} \cap del = \{p', p\} \cap pre$, we also have $\{p, p'\} \cap \delta(s, a) = \emptyset$.

$\underline{\{p', p\} \cap add \neq \emptyset, \{p', p\} \cap del \neq \emptyset}$ We have $p \in add(\xi(a))$ and $p \notin del(\xi(a))$, and thus $p \in \delta(\xi(s), \xi(a))$. On the other hand, from $add \cap del = \emptyset$ and $\{p', p\} \cap add \neq \emptyset$ we immediately have $\{p, p'\} \cap \delta(s, a) \neq \emptyset$.

This completes the proof of (ii). $\square$

Arbitrarily coarse variable domain abstractions may be generated by iterating the application of Definition 2.2. Note that variable domain abstraction is a refinement of Edelkamp's (2001) abstraction—instead of acting as if the irrelevant positions could be totally ignored, we do distinguish whether or not the package currently *is* at such a position. This makes all the difference between preserving optimal solution length or not.[15]

As hinted above, after variable domain abstraction we may be able to apply further *simplifications*. A simplification, in our terminology, is similar to an abstraction in that it

15. A topic for future work is to explore whether the refined abstraction can lead to better pattern database heuristics for STRIPS problems.

manipulates a planning task at the language level. However, while abstractions may alter the task's semantics, simplifications do not; i.e., simplifications do not introduce any new transitions or goal states. Concretely, we consider two simplifications. A planning task $\mathcal{P} = (P, A, I, G)$ has *duplicate actions* if there exist $a, a' \in A$ so that $pre(a) = pre(a')$, $add(a) = add(a')$, and $del(a) = del(a')$.[16] The simplified planning task is like $\mathcal{P}$ except that a' is removed. A planning task $\mathcal{P} = (P, A, I, G)$ has *irrelevant add effects* if there exists $a \in A$ so that $pre(a) \cap add(a) \neq \emptyset$. The simplified planning task is like $\mathcal{P}$ except that we remove $pre(a)$ from $add(a)$.

Obviously, duplicate actions and irrelevant add effects may arise as an outcome of variable domain abstraction. An example of the latter is an action moving a package from an irrelevant location into an irrelevant truck. An example of the former are two actions loading a package onto an airplane from distinct but irrelevant locations.[17] In our implementation, we have a simple post-abstraction processing in which we perform all these simplifications.

As we shall see in Section 4.3, the simplifications *can* lead to decreased resolution complexity, thereby offsetting our result that abstractions as such, in many cases, cannot. It may seem a little artificial to distinguish abstractions and simplifications in this way, seeing as many abstractions are bound to enable us to simplify. However, note that this distinction only serves to identify the borderline between what can, and what cannot, reduce resolution complexity. Anyhow, as we shall see in the next section, abstraction does not tend to help empirically with the performance of SATPLAN even *with* post-abstraction simplifications.

3. Empirical Results

We have performed a broad empirical evaluation of the effect of abstractions on the efficiency of optimizing planning algorithms. We mostly focus on variable domain abstraction, as in Definition 2.2, since it is clearly most promising for obtaining solution length preserving abstractions.

Section 3.1 explains the specific variable domain abstractions we employ in our experiments. Section 3.2 explains the experimental setting and how we chose to present the huge data set of the results. Section 3.3 describes our experiments with variable domain abstraction in the IPC benchmarks, and Section 3.4 discusses our results with domain-specific abstractions of hand-made instances in certain benchmark domains where the amount of irrelevance can be controlled. Section 3.5 briefly summarizes our findings with abstraction methods other than variable domain abstraction.

3.1 Variable Domain Abstractions

We designed three different methods to automatically generate variable domain abstractions. The methods as listed below are based on increasingly conservative approximations

16. Note that we define actions not to *be* triples of pre, add, del, but to *have* these components; hence two actions with identical pre, add, del may be contained in the set A. This reflects practical planner implementations, where actions have names and/or unique IDs, and checks for duplicate actions are not usually performed.
17. In a similar fashion, duplicate actions may arise as an outcome of Edelkamp's (2001) pattern database abstraction.

of relevance. As is common in relevance approximations (cf. Nebel et al., 1997), the algorithmic basis is, in all cases, a simplified backchaining from the goals.

1Support starts in the first layer of a planning graph that contains all goal facts (possibly with mutexes between them). For each goal fact in that layer, it selects one achiever in the preceding action layer and marks the preconditions of that action as new sub-goals; then the process is iterated for the created sub-goals.

AllSupports proceeds similarly to *1Support* except that it selects *all* achievers for each (sub-)goal.

AllSupportsNonMutex proceeds similarly to *AllSupports* except that it starts the backchaining at the first plan graph layer that contains the goals *without* mutexes.

In all three methods, the selected set of "relevant" facts $R \subseteq P$ is taken to be the set of goals and sub-goals created during the backchaining. This set of facts is turned into a variable domain abstraction as follows. First, we compute a partition of the problem's fact set P into subsets $P_1, \dots, P_k$ of pairwise persistently mutex facts. We take these subsets to correspond to the underlying multiple-valued variables (e.g., the position of a package). Then we perform abstraction within those P_i where not all values are relevant, i.e., $P_i \setminus R \neq \emptyset$. Within each such subset P_i, we arbitrarily choose one irrelevant fact p, i.e., $p \in P_i \setminus R$. We then replace all other irrelevant facts, i.e., all $q \in P_i \setminus R$ where $q \neq p$, with p.

As an example, in Logistics, *1Support* abstracts away all $in(p, v)$ facts for each package p except for those vehicles v that were selected as a support—in particular, a single airplane. In contrast, *AllSupports* will mark $in(p, v)$ as relevant for *all* airplanes v unless some special case applies (e.g., p must be transported within its origin city only). Finally, *AllSupportsNonMutex* is even more conservative and covers some of the special cases in which *AllSupports* abstracts an $in(p, v)$ fact away. Note that identifying positions inside airplanes with positions outside airplanes may well affect the length of an optimal plan.

In addition to the domain-independent, automatic variable domain abstractions, for six IPC domains we have designed domain-specific solution length preserving variable domain abstractions by hand. For Logistics, the domain-specific abstraction was explained in the introduction. For Zenotravel, we use a similar abstraction exploiting irrelevant object positions. In Blocksworld, $on(A, B)$ is considered irrelevant if B is neither the initial nor the goal position of A, and B is initially clear.[18] For Depots, which is a combination of Logistics and Blocksworld, our abstraction is a combination of the two individual abstractions. For Satellite, our abstraction performs a simple analysis of goal relevance to detect directions that are irrelevant for a satellite to turn to. A direction is relevant only if it is the satellite's initial direction, its goal direction, or a potential goal or camera calibration target. Similarly, in Rovers, a waypoint (location) is considered relevant for a rover only if it is either the initial position, or it is relevant for a needed rock sample/soil sample/image, or it necessarily lies on a path the rover must traverse to reach some other relevant location.

18. The last of these conditions is necessary to avoid the possibility of "clearing" a block C by moving A away from C although A is actually placed on some third block.

3.2 Experiment Setup and Presentation

The presented data were generated on a set of work stations running Linux, each with a Pentium 4 processor running at 3 GHz with 1 GB RAM. We used a time cutoff of 30 minutes. We experimented with the plan-length optimizing planners SATPLAN'04, IPP (Koehler, Nebel, Hoffmann, & Dimopoulos, 1997), and Mips-BDD (Edelkamp & Helmert, 1999).[19] Our choice of SATPLAN'04 rather than SATPLAN'06 is arbitrary, except that, by using the naive encoding (C), the resolution best case of SATPLAN'04 *can* be improved by variable domain abstraction—making our "bad" empirical results below even more significant. Note also that, although SATPLAN'06 could be considered "more recent", it contains no developments beyond SATPLAN'04, other than switching back to an older version of the encoding method.

As test examples, we took, with few exceptions listed below, all STRIPS domains used in all international planning competitions until and including IPC-2004. Precisely, we use

(IPC-2004) Airport, Dining Philosophers, Optical Telegraph, Pipesworld NoTankage, Pipes- world Tankage, and PSR.

(IPC-2002) Depots, Driverlog, Freecell, Rovers, Satellite, and Zenotravel.

(IPC-2000) Blocksworld and Logistics. (Miconic-STRIPS is just a very simple version of Logistics, Freecell is part of our IPC-2002 set.)

(IPC-1998) Grid, Mprime, and Mystery. (Movie is trivial, in Gripper variable domain abstraction cannot preserve solution length, Logistics is part of our IPC-2000 set.)

Our measurements are aimed at highlighting the potential that abstraction in principle has of speeding up the computation of information about a task. Concretely, given a planning task T, we create an abstract version T^σ of T, and run a planner X on it. There are three possible outcomes:

(1) X finds a plan for T^σ, an *abstract plan*, and it happens to be a real plan (that is, a plan for T). We record the time taken to find the plan, along with the time taken by X to find a plan given the original task T.

(2) X finds a plan for T^σ that is not a real plan. Since all our planners optimize plan length, the information we still gain is the length of the optimal abstract plan, which is a lower bound on the length of the real plan. We record the time taken to compute that bound (for example, for SATPLAN'04, the time taken up to the last unsatisfiable iteration), along with the time taken by X to compute the *same* lower bound given the original task T.

(3) X runs out of time or memory. In this case, one could record the time taken up to the last lower bound proved successfully. For the sake of readability, we omit this here and consider only cases (1) and (2) above.

19. While SATPLAN'04 and IPP optimize step-length of the plan, Mips-BDD optimizes sequential plan length. However, again, as the performance of the planners does not stand for a comparative evaluation here, we refer to all three simply as plan-length optimizing planners.

Note that, in the spirit of being optimistic about the usefulness of abstraction, we do *not* include the time taken to create the abstract task T^σ. Note also that we actually obtain several results for each pair T and X, namely one result for every particular variable domain abstraction. For the sake of readability, we do not include these distinctions in the results (the distinctions are mostly inconclusive and uninteresting anyway), and instead present the results from the following "best abstraction" perspective. We skip abstract tasks that were either not solved, or that are not "abstract" since all facts are considered relevant. If no abstract task remains, we skip the instance. Otherwise, we select the abstract task providing the best information about the instance: the best case is that the abstract plan is real, else we select the highest lower bound.[20] If there are several abstractions providing the same best information, we choose the one with lowest runtime.

3.3 IPC Benchmarks

Due to the sheer size of our experiments—3 planners multiplied with 17 domains—discussing the entire result set is neither feasible nor would it be useful. An online Appendix (see JAIR web page for this article) contains detailed data for the three optimal planners. Herein, we provide a summary analysis showing the main points, with a particular focus on SATPLAN'04.

Detailed data for SATPLAN'04 on 4 of our 17 IPC domains is given in Table 1:

- Depots and Satellite are selected into the table because they are the only 2 of our 17 domains where the abstraction brings a somewhat significant advantage.
- Logistics is selected because it is our illustrative example.
- PSR is selected due to being an interesting case—unusually, the current optimal planners do just as well (or badly) on PSR as the current sub-optimal (satisficing) planners.

In each domain, we selected the 13 most challenging instances, where "challenging" is measured as the runtime taken in the original task. Note that this problem-instance selection criterion for our presentation is also "optimistic" from the point of view of abstraction. For each instance, Table 1 first specifies whether the found abstract plan was a real plan or not. This characterizes the problem instance in terms of cases (1) and (2) explained above, and the corresponding runtimes of SATPLAN'04 on abstract and real tasks are then given by the rows t_a and t_r. The table then specifies the lower bound l_g proved for the real task by its planning graph (that is, t if $F(t)$ is the first fact layer to contain all goal facts with no mutexes between them), the lower bound l_a proved by SATPLAN'04 in the abstract task, and, finally, the actual length l_r of the optimal plans for the real task. The last row *RelFrac* in the table specifies the percentage of facts considered relevant.

For Depots, the best-case data shown in Table 1 is scattered across all four kinds of variable domain abstractions, with the automatic abstractions being sometimes more and sometimes less aggressive than our handmade abstraction. For example, instances numbers 11 and 15 have their best case with the very aggressive *1Support* strategy. Most of the time

20. Note here that the quality of the information is essential. If the abstraction only tells us that the plan must have at least $n-1$ steps, and the real plan length is n, then we must still prove the bound n, which typically takes more time than all other bounds together.

Depots													
Index	3	4	7	8	10	11	13	14	15	16	17	19	21
IsReal?	Y	Y	Y	N	Y	N	Y	N	N	Y	Y	Y	N
t_a	73.29	429.74	10.44	228.30	47.77	49.73	9.07	118.90	56.83	13.81	18.4	460.75	342.20
t_r	45.77	472.01	12.25	22.52	33.11	2.13	12.42	12.72	4.45	6.54	17.41	–	55.70
l_g, l_a, L	11,12,12	12,14,14	7,10,10	9,13,14	8,10,10	13,10,?	9,9,9	9,9,?	10,8,?	8,8,8	6,7,7	8,10,10	7,7,7
RelFrac	88%	88%	77%	76%	87%	27%	85%	50%	20%	89%	58%	92%	51%
Abs	AS	AS	ASnm	AS	ASnm	1S	ASnm	HM	1S	ASnm	ASnm	AS	HM
Logistics													
Index	10	13	14	15	16	17	18	19	20	21	22	23	39
IsReal?	Y	Y	Y	N	N	N	N	N	N	N	N	N	N
t_a	0.99	91.43	25.41	70.57	111.66	150.37	770.44	684.19	820.59	615.01	929.64	965.49	1.1
t_r	2.14	75.28	32.87	68.12	75.79	106.61	642.97	672.25	721.73	430.95	721.82	769.36	1.9
l_g, l_a, L	10,15,15	9,13,13	9,12,12	9,13,13	9,13,13	9,13,14	9,15,15	9,15,15	9,15,15	9,14,?	9,15,?	9,15,?	9,8,?
RelFrac	33%	48%	48%	55%	47%	44%	42%	30%	33%	48%	28%	43%	14%
Abs	1S	ASnm	ASnm	ASnm	ASnm	ASnm	HM	HM	HM	AS	HM	AS	1S
PSR													
Index	16	22	29	31	33	36	37	40	42	47	48	49	50
IsReal?	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	Y
t_a	0.63	64.29	11.16	1.04	0.93	2.44	2.4	0.9	0.73	17.75	125.49	3.02	0.59
t_r	0.8	71.86	12.05	8.38	1.33	4.96	2.26	6.39	0.87	17.41	131.94	3.11	1.03
l_g, l_a, L	5,15,15	7,25,25	7,18,18	5,16,16	5,16,16	8,16,16	7,19,19	5,14,15	5,16,16	4,23,23	7,26,26	8,19,?	4,16,16
RelFrac	47%	75%	79%	49%	48%	90%	60%	48%	53%	47%	80%	37%	30%
Abs	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	ASnm	1S	ASnm
Satellite													
Index	2	4	5	8	9	10	11	12	13	14	15	17	18
IsReal?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
t_a	64.43	112.30	53.06	310.76	34.35	168.03	84.47	874.84	931.17	256.16	282.79	65.79	112.50
t_r	74.26	104.05	51.53	250.08	40.65	176.47	160.03	–	–	425.44	429.81	152.37	217.76
l_g, l_a, L	6,12,12	6,10,10	4,7,7	4,8,8	4,6,6	4,8,8	4,8,8	6,14,14	4,13,13	4,8,8	4,8,8	4,6,6	4,8,8
RelFrac	37%	91%	95%	90%	84%	85%	74%	76%	76%	78%	75%	67%	74%
Abs	1S	HM	HM	HM	HM	HM	HM	HM	HM	HM	HM	HM	HM

Table 1: Full results, in some selected domains, for SATPLAN'04 and variable domain abstraction (best-of, see text). Notations: "Index": index (number) of instance in respective IPC suite; "IsReal": whether the abstract plan real (Y) or not (N); L: the length of the optimal plan (? if not known), l_g: lower bound on plan length proved by planning graph, l_a lower bound proved in abstract task; t_a: runtime (secs) needed to prove the lower bound l_a in abstract task; t_r runtime (secs) needed to prove the same lower bound l_a but in real task; "RelFrac": fraction of facts considered relevant; dashes: time-out; "Abs": is the corresponding form of variable domain abstraction, *1Support* (1S), *AllSupports* (AS), *AllSupportsNonMutex* (ASnm), and handmade (HM).

Domain	SATPLAN'04 Index	t_a	t_r	*IsReal?*	l_g, l_a, l_r	*RelFrac*
Airport	20	33.9	29.5	Y	25,32,32	73%
Blocksworld	7	118.3	11.2	Y	16,20,20	47%
Depots	19	460.8	–	Y	8,10,10	92%
Dining Philosophers	29	6.2	5.5	Y	7,11,11	71%
Driverlog	13	342.0	113.7	N	9,11,12	61%
Freecell	1	0.8	0.8	Y	4,5,5	82%
Grid	2	96.4	3.1	N	19,13,?	10%
Logistics	23	965.5	769.4	N	9,15,15	43%
Mprime	5	6.2	5.2	Y	6,6,6	78%
Mystery	20	180.7	112.4	Y	7,7,7	76%
Optical Telegraph	13	43.6	32.0	N	11,13,13	53%
Pipesworld NoTankage	12	521.7	455.5	Y	8,14,14	86%
Pipesworld Tankage	8	393.8	143.4	N	5,6,?	89%
PSR	48	125.5	131.9	Y	7,26,26	80%
Rovers	8	74.6	97.8	Y	5,9,9	87%
Satellite	12	874.8	–	Y	6,14,14	76%
Zenotravel	13	338.4	244.8	N	4,7,7	67%

Table 2: Results for SATPLAN'04 with the best-case variable domain abstraction on the most challenging successful instances of each domain. Notation as in Table 1.

the runtime is better on the original task, yet there are a few cases where the abstraction brings a quite significant advantage. Most notably, in instance number 19 SATPLAN'04 runs out of time on the original task, but solves the abstract task, finding a real plan, within a few minutes. In Logistics, the best-case data is mostly, though not exclusively, due to the conservative *AllSupportsNonMutex* and our handmade abstractions. The abstract runtime is worse in all but three cases (nos. 10, 14, 39), where it is slightly better. In PSR, the best cases are almost exclusively due to the conservative *AllSupportsNonMutex* abstraction. As for runtimes, abstraction is usually faster, but only marginally. Satellite is the only one of our 17 domains where abstraction brings a significant (and largely consistent) runtime advantage. The best cases are almost exclusively due to our hand-made abstraction. All abstract plans are real plans, often found significantly faster than for the original task. It is unclear to us why the results are good in Satellite, but, for example, not in Logistics, where the state space reduction is much larger.

Next, Table 2 provides an overview of the results for SATPLAN'04 in our 17 IPC domains. To make data presentation feasible, we select just one instance per domain—the "most challenging successful" instance. By successful, we mean that at least one abstract task of that instance was solved (abstract plan found), and this abstract task was indeed abstract (not all facts relevant). By challenging, we mean maximum runtime on the original task.[21]

21. Another strategy would be to select the task that maximizes $t_r - t_a$, the time advantage given by abstraction. However, in most cases this strategy would select a trivial instance: namely, because $t_r - t_a$ is consistently negative, and maximal in the easiest tasks.

Domain	IPP					
	Index	t_a	t_r	*IsReal?*	l_g, l_a, l_r	*RelFrac*
Airport	8	67.9	0.3	Y	25,26,26	77%
Blocksworld	7	3.1	0.0	Y	16,20,20	47%
Depots	17	254.4	268.4	Y	6,7,7	58%
Dining Philosophers	5	170.4	138.0	Y	7,11,11	71%
Driverlog	9	1.1	0.7	N	7,10,10	84%
Grid	1	0.1	0.2	N	14,7,14	43%
Logistics	8	0.2	1.0	Y	9,11,11	49%
Mprime	2	0.6	1.0	N	5,4,5	62%
Mystery	2	0.4	0.7	N	5,4,5	60%
Optical Telegraph	2	15.7	5.2	N	11,13,13	53%
Pipesworld NoTankage	5	0.0	0.0	Y	4,6,6	88%
PipesworldTankage	7	32.2	0.6	N	4,5,6	82%
PSR	10	0.0	0.0	N	5,4,5	37%
Rovers	6	592.5	375.7	N	7,12,12	90%
Satellite	7	100.3	2010.7	Y	4,6,6	87%
Zenotravel	12	344.3	322.4	Y	4,6,6	67%

Table 3: Similar to Table 2, but for the IPP planner.

It is useful to discuss the 17 domains in groups with similar behavior. Depots, Logistics, PSR, and Satellite have already been discussed. In each of Airport, Dining Philosophers, Driverlog, Mystery, Mprime, Optical Telegraph, Pipesworld NoTankage, and Pipesworld Tankage, SATPLAN'04 runtimes are consistently lower on the original tasks, with few exceptions mostly among the easiest instances. The picture is less consistent but qualitatively similar in Zenotravel. The degree of the advantage varies. It is relatively moderate in Dining Philosophers (up to 7% less runtime on original task), Optical Telegraph (up to 23%), Airport (up to 28%), Pipesworld Tankage (up to 28%), and Mprime (up to 36%); it is much stronger in Zenotravel (up to 75%), Mystery (up to 80%), Driverlog (up to 89%), and Pipesworld NoTankage (up to 92%).

In Rovers, the runtime results are inconclusive, with minor advantages for abstract or real depending on the instance. In Blocksworld, SATPLAN'04 solves abstract tasks with up to 7 blocks only, independently of the abstraction used; we don't know what causes this bad behavior. In Freecell, most of the time *AllSupports* and *AllSupportsNonMutex* do not abstract anything, and in all abstractions generated with *1Support*, SATPLAN'04 runs out of time, leaving instance number 1 as the only "successful" case, shown in Table 2. In Grid, finally, the IPC 1998 test suite contains only 5 instances, which become huge very quickly. SATPLAN'04 can solve (abstract or real) only instances numbers 1 and 2, and the latter is shown in Table 2.

Tables 3 and 4 provide a similar snapshot on the results with IPP and Mips.BDD, respectively. The picture for IPP is, roughly, similar to that for SATPLAN'04. The main difference is, in fact, that IPP is a weaker solver than SATPLAN'04 in many domains, to the effect that some more domains contain no interesting data. Specifically, in Driverlog, Mprime, Mystery, Pipesworld NoTankage, and PSR, IPP either solves the instances in no time, or not at all. Like for SATPLAN'04, we see an advantage for abstraction in Depots

	Mips.BDD					
Domain	Index	t_a	t_r	*IsReal?*	l_g, l_a, l_r	*RelFrac*
Airport	–	–	–	–	–	–
Blocksworld	–	–	–	–	–	–
Depots	2	1.1	–	Y	7,15,15	81%
Dining Philosophers						
Driverlog	10	503.6	–	Y	5,17,17	84%
Freecell	–	–	–	–	–	–
Grid	1	1.8	–	N	14,7,?	43%
Logistics	12	7.2	–	Y	10,42,42	43%
Mprime	–	–	–	–	–	–
Mystery	–	–	–	–	–	–
Optical Telegraph						
Pipesworld NoTankage						
Pipesworld Tankage						
PSR	25	0.7	13.5	Y	4,9,9	37%
Rovers	7	142.6	340.6	Y	5,18,18	86%
Satellite						
Zenotravel	11	271.0	–	Y	4,14,14	66%

Table 4: Similar to Tables 2–3, but for the Mips.BDD planner.

and Satellite, and we note that for Satellite this difference is consistently huge. We also see a vague advantage for abstraction in Logistics. For Mips.BDD, even more domains gave no meaningful data. In the domains dashed out in Table 4, Mips.BDD runs out of time on even the smallest instances. In the domains left empty, we either could not run Mips.BDD for some technical reasons, or it stopped abnormally. In the remaining data set of 7 domains, however, our abstractions (as expected) bring a consistent advantage for Mips.BDD. In particular, consider the behavior in Logistics, Rovers, and Zenotravel—in these domains, Mips.BDD is vastly improved by abstraction while SATPLAN'04 and IPP are more or less inconclusive.

3.4 Constructed Benchmarks

The above has shown that the use of abstraction—of variable domain abstraction, at least—to speed up state of the art planning systems varies from quite promising for Mips.BDD to rather hopeless for SATPLAN'04. We ran a number of focused experiments to examine the more subtle reasons for this phenomenon. These experiments have been done on three IPC benchmarks—Logistics, Rovers, and Zenotravel—where the results on the IPC test suites are relatively bad, although we are in possession of hand-made abstractions. We wanted to test what happens when we scale the instances on *irrelevance*. The respective experiment for Logistics, Figure 1, was discussed in the introduction. For Rovers, we tried a large number of instance size parameters, and even minor modifications of the operators, but we could not find a setting that contained a lot of irrelevance *and* was challenging for SATPLAN'04 and IPP. In short, it appears that the Rovers domain is not amenable to abstraction techniques. For Zenotravel, we obtained the picture shown in Figure 2.

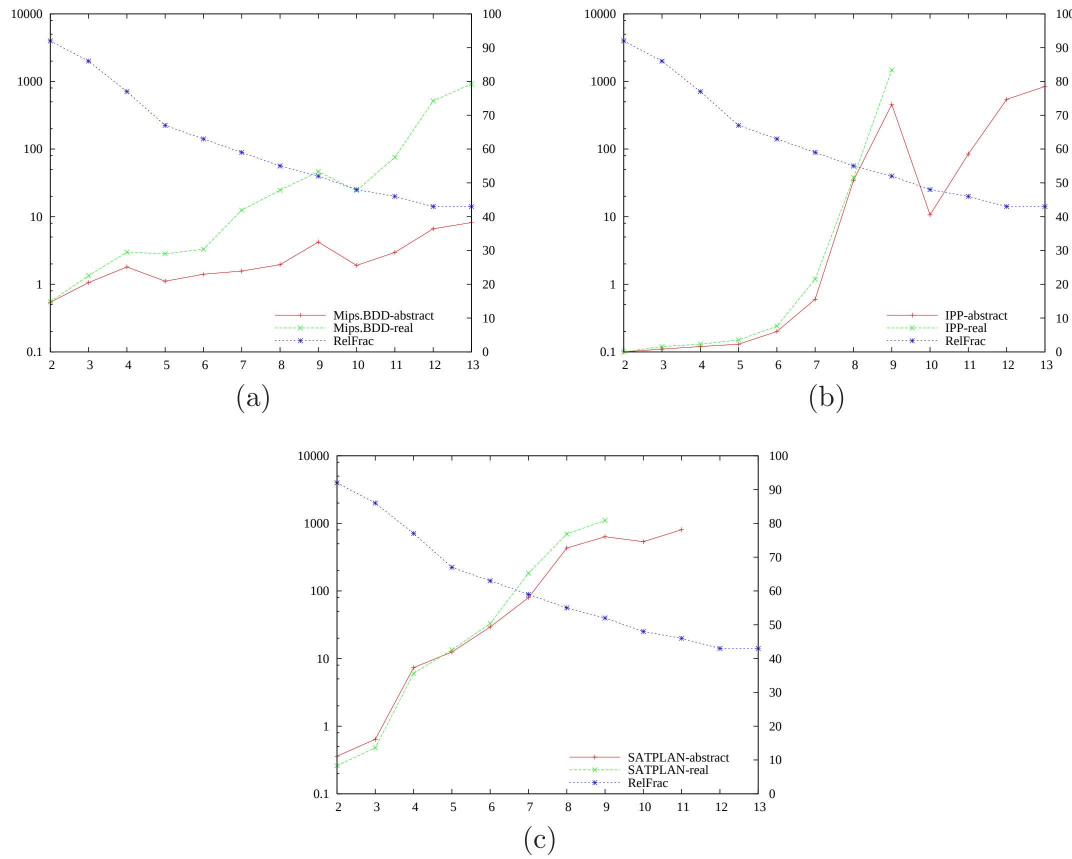

Figure 2: Runtime performance of Mips.BDD (a), IPP (b), and SATPLAN'04 (c), with ("abstract") and without ("real") our hand-made variable domain abstraction, in Zenotravel instances explicitly scaled to increase the amount of irrelevance. Horizontal axis scales the number of cities, left vertical axis shows total runtime in seconds, right vertical axis shows the percentage *RelFrac* of relevant facts.

The shown Zenotravel instances always feature 2 airplanes and 5 persons. The number of cities scales from 2 to 13. As in Logistics, we generated 5 random instances per size, and show average values with a time-out of 1800 seconds, stopping plots when 2 time-outs occurred at an instance size. All in all, the relative behavior of the abstract and real curves for each planner is quite similar to what we observed in Figure 1 with Logistics. For SATPLAN'04 and IPP, abstraction has a slight disadvantage with high *RelFrac*, and becomes much more efficient as *RelFrac* decreases. For Mips.BDD, the advantage brought by the abstraction is much more pronounced, and decreasing *RelFrac* consistently widens the gap between solving abstract and real tasks. The average value of *RelFrac* in the IPC 2000 Zenotravel benchmarks is 64%, lying in between 5 cities (67%) and 6 cities (63%) in Figure 2, where there is not yet much gained by the abstraction.

In summary, it appears that some planning benchmarks (like Rovers) do not have good abstractions, and most others (like Logistics and Zenotravel) do not have enough irrelevance in the IPC test suites.

It is important to note that the situation may not be quite as bad for *unsolvable* examples. Consider the IPC benchmarks Dining Philosophers and Optical Telegraph (Edelkamp, 2003). Dining Philosophers is an extremely basic benchmark that cannot be abstracted much further. In contrast, Optical Telegraph is essentially a version of Dining Philosophers with a complex "inner life" (exchanging data between the two "hands" of each philosopher). This inner life does not affect the *existence* of a solution (deadlock situation), which depends exclusively on the outer interfaces of the "philosophers", that is, taking and releasing "forks". However, the inner life does, of course, affect the *length* of a solution, if one exists. We constructed an unsolvable version of the domain (without deadlock situation) by giving the "philosophers" more flexibility in releasing forks. As one would expect, in this setting abstracting the inner life away gives huge savings, i.e., the tasks can be proved unsolvable much more efficiently. This suggests that it may be easier to abstract unsolvable tasks, without invalidating the property of interest. Exploring this is a topic for future work. While most planning benchmark domains do not naturally contain unsolvable instances, in over-subscription planning this issue may become relevant (Sanchez & Kambhampati, 2005; Meuleau, Brafman, & Benazera, 2006).

3.5 Other Abstractions

As discussed earlier, one cannot expect that removing preconditions, goals, or entire facts preserves plan length in interesting cases. There *are*, however, certain cases where some delete effects can safely be ignored. Specifically: in Driverlog, Logistics, Mprime, Mystery, and Zenotravel, one can ignore those deletes of "load" and "unload" actions which state that an object is no longer at its origin location (load) respectively that an object is no longer inside the vehicle (unload); in Rovers one can ignore some deletes of actions taking rock or soil samples, namely those deletes stating that the sample is no longer at its origin location. We ran each of our planners on the respective abstracted tasks. The results can be summarized as follows.

SATPLAN'04 has a clear loss in runtime from using the abstraction in Driverlog (e.g., task number 15 is solved abstract vs. real in 693.0 vs. 352.3 sec).

IPP has a vast gain from abstraction in Logistics (e.g., 52.8 vs 5540.1 sec in number 12), and a vast loss in Zenotravel (e.g., 318.5 vs 2.5 sec in number 12).

Mips.BDD has a vast loss from abstraction in Driverlog, Logistics, and Zenotravel (e.g., 163.8 vs. 8.3 sec in Zenotravel number 8).

The results are inconclusive for all other planner/domain pairs.

4. Resolution Complexity

As discussed in the introduction, we were surprised to see very little improvement to SATPLAN in our experiments, despite the dramatic state space reductions brought about by variable domain abstraction. We now shed some light on this issue, by examining resolution complexity in the original vs. the abstracted planning tasks. Throughout the section, we consider the situation where the plan length bound—the number of time steps in the CNF encoding—is too small, and thus the CNFs are unsatisfiable. Note that this is the case in all but one of the SAT tests performed by SATPLAN. In particular, it is the case in the SAT tests where SATPLAN proves optimality of the plan, that is, the non-existence of a plan with $n-1$ steps where n is the length of an optimal plan. This proof is typically very costly, accounting for a large fraction of the runtime taken by SATPLAN.

We consider all abstraction methods introduced in Section 2.4, plus (for completeness) a hypothetical abstraction method that adds new initial facts. We show in Section 4.1 that, in many cases, the resolution complexity cannot be improved by delegating the optimality proof to within the abstraction. In Section 4.2 we then show that, in all considered cases, the resolution complexity can become exponentially worse. Section 4.3 briefly examines the effect of post-abstraction simplifications. For the sake of readability, herein most proofs are replaced with proof sketches. The full proofs are available in Appendix A.

Recall that resolution complexity is defined as the length of the shortest possible resolution refutation. In all proofs, our arguments are for general (unrestricted) resolution. However, our constructions do not affect the structure of the resolution refutations, and hence *the results hold as stated (except for a slight weakening in the case of Lemma 4.14) for many known variants of resolution, including tree-like (DPLL), regular, and ordered resolution.* In general, the results hold for any variant of resolution for which setting variables to True or False or replacing one variable with another preserves proof structure (the slightly exceptional status of Lemma 4.14 will be explained below when we discuss that result).

In the remainder of the paper, if $\mathcal{P}$ is a planning task and σ is an abstraction, then by $\mathcal{P}^\sigma$ we denote the respective abstracted planning task, that is, the planning task that results from applying σ to $\mathcal{P}$.

4.1 Can Resolution Complexity Become Better?

We prove three main results, which are captured by Theorems 4.1–4.3 below. The first result holds for all four SAT encodings (A)–(D) as listed in Section 2.2; the other two results apply to encodings (A) and (C), respectively. For the respective encodings and abstraction methods, the results essentially say that resolution complexity cannot decrease by applying the abstraction. As outlined in the introduction, a catchy (if imprecise) intuition behind these results is that over-approximations (abstractions) result in "less constrained" formulas, which are harder to refute. For encoding (C), the result is offset by the effort required to recover all Graphplan mutexes; we will get back to this below. For the theorems that follow, recall from Section 2.3 that $\mathcal{RC}(\phi)$ denotes the resolution complexity of ϕ, i.e., the size of the smallest resolution proof of unsatisfiability of ϕ.

Theorem 4.1. *Let $\mathcal{P}$ be a planning task. Assume we use any of the encoding methods (A)–(D). Let σ be an abstraction of $\mathcal{P}$ that consists of any combination of:*

(a) adding initial facts;

(b) ignoring preconditions, goals, or deletes; and

(c) removing a fact completely.

Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.

Theorem 4.2. *Let $\mathcal{P}$ be a planning task. Assume we use encoding method (A). Let σ be an abstraction of $\mathcal{P}$ that consists of any combination of:*

(a) adding initial facts;

(b) ignoring preconditions, goals, or deletes;

(c) removing a fact completely; and

(d) variable domain abstraction.

Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.

Theorem 4.3. *Let $\mathcal{P}$ be a planning task. Assume we use encoding method (C). Let σ be an abstraction of $\mathcal{P}$ that consists of any combination of:*

(a) adding initial facts;

(b) ignoring preconditions, goals, or deletes;

(c) removing a fact completely; and

(d) variable domain abstraction.

Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Let M be the number of resolution steps required to infer from ϕ the additional mutex clauses that appear in ϕ_A, where ϕ_A is the encoding of b-step plan existence in $\mathcal{P}$ as per encoding (A). Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma) + M$.

Note in all these theorems that n, defined to be the length of a shortest plan for $\mathcal{P}^\sigma$, necessarily satisfies $n \leq m$ where m is the length of a shortest plan for $\mathcal{P}$. Hence, for $n \leq b \leq m$, ϕ^σ is satisfiable. Detecting this – finding out that $\mathcal{P}^\sigma$ has a plan of length b – does not give us any information about the length of a shortest plan for $\mathcal{P}$. For $0 \leq b < n$, however, ϕ^σ is unsatifiable, which tells us that $b + 1$ is a lower bound on plan length in $\mathcal{P}$. Hence, what the theorems say is this: either $b \geq n$ and σ is too coarse to disprove existence of a plan of length b; or $b < n$ and σ does not decrease the resolution complexity of that disproof—at least by no more than the complexity of deriving the additional mutexes, in the case of Theorem 4.3.

Let us first linger a bit on Theorem 4.3. The general intuition of our results is that abstractions induce less constrained formulas, and hence resolution complexity cannot decrease. So why does this hold for encoding (A) as stated in Theorem 4.2 but not, in a strict sense (see Proposition 4.13 later in this section), for encoding (C)? Basically, the answer is that the intuition is imprecise in this general formulation, and the devil is in the details. In this particular case, the issue is that *variable domain abstraction makes use of mutex* *relations which encoding (C) is not aware of.* Sometimes, an indirect mutex in the original task (omitted in encoding (C)) becomes a direct mutex in the abstraction (included in encoding (C)). Refuting ϕ might then involve recovering that mutex, which a refutation of ϕ^σ need not do. Hence, a potential improvement in resolution complexity may stem from the power of mutex relations. The upper bound specified in Theorem 4.3 shows that this is the only thing that an improvement can be due to. Proposition 4.13 below provides an example where a mutex must be recovered, and hence proves that an analogue of Theorem 4.2 does not hold for encoding (C).

It is an open question whether an analogue of Theorem 4.2 holds for encoding (B), and whether an analogue of Theorem 4.3 holds for encoding (D). As we will discuss further below, these open questions appear to be related to some intricate properties of Graphplan-based encodings with vs. without fact variables. What we do know is that mutexes may need to be recovered also in encoding (D): the example provided by Proposition 4.13 works for both encodings (C) and (D). Further, we establish a connection between the two open questions: if an analogue of Theorem 4.2 holds for encoding (B), then we immediately get that an analogue of Theorem 4.3 holds for encoding (D).

We now consider all this in detail. Note that, as far as the removal of goals is concerned, the theorems are actually trivial: for all four encoding methods, if σ only removes part of the goals, then ϕ^σ is a sub-formula of ϕ. For all other abstraction methods, the latter is not the case. We treat removal of goals together with the other methods since that treament does not cause any overhead, and the goal clauses need to be discussed anyway (the set of achievers of a goal may change).

For some of the proofs, we need a helper notion that captures over-approximated planning graphs. Assume a planning task $\mathcal{P}$ and its planning graph $PG(\mathcal{P})$, and assume that σ is an abstraction. Then $PG(\mathcal{P}^\sigma)$ will typically have many more vertices than $PG(\mathcal{P})$. This captures the fact that $\mathcal{P}^\sigma$ allows no fewer (and often more) facts and actions than $\mathcal{P}$ does. This will, in general, result in many *more constraints* in a propositional translation of the planning task. With more constraints, it may seem like the abstraction could, in principle, make it possible to derive an easier/shorter proof of the fact that no plan exists within the specified bound. However, a closer inspection *restricted to facts and actions already available in the original planning graph* reveals that one often ends up with *fewer and weaker constraints* than for the original task. We now introduce some notations to make this formal.

Definition 4.4. For a planning task $\mathcal{P}$ and an abstraction σ of it, $PG^\sigma(\mathcal{P})$ is defined to be the subgraph of $PG(\mathcal{P}^\sigma)$ induced by the vertices of $PG(\mathcal{P})$. Similarly, $PG^\sigma_{red}(\mathcal{P})$ is defined to be the subgraph of $PG_{red}(\mathcal{P}^\sigma)$ induced by the vertices of $PG_{red}(\mathcal{P})$.

Definition 4.5. Let $\mathcal{P}$ be a planning task. An abstraction σ is called a *planning graph abstraction* of $\mathcal{P}$ if $PG^\sigma(\mathcal{P})$ and $PG(\mathcal{P})$ have identical sets of vertices and the following conditions hold:

(1) $E_{add}(PG^\sigma(\mathcal{P})) \supseteq E_{add}(PG(\mathcal{P}))$,

(2) $E_{pre}(PG^\sigma(\mathcal{P})) \subseteq E_{pre}(PG(\mathcal{P}))$,

(3) $E_{mutex}(PG^\sigma(\mathcal{P})) \subseteq E_{mutex}(PG(\mathcal{P}))$, and

(4) $\sigma(G) \subseteq G$,

where G and $\sigma(G)$ are the goal states of $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. An abstraction σ is called a *reduced planning graph abstraction* if the above conditions hold for $PG^\sigma_{red}(\mathcal{P})$ and $PG_{red}(\mathcal{P})$ instead.

Lemma 4.6. *Let $\mathcal{P}$ be a planning task. Assume we use encoding method (A) or (B). Let σ be a planning graph abstraction of $\mathcal{P}$. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.*

Proof Sketch. Say we set to False all variables of ϕ^σ that do not appear in ϕ, i.e., we fix the value of those variables to be 0. Because of the way we defined $PG^\sigma(\mathcal{P})$, this yields precisely the propositional encoding of $PG^\sigma(\mathcal{P})$. We show that, after this variable restriction, the clauses surviving in ϕ^σ are also present in ϕ, either as is or in a stronger form (i.e., with fewer literals). For example, in encoding (A) each precondition clause C^σ of ϕ^σ has a corresponding clause C in ϕ due to condition (2) of Definition 4.5, which states that σ does not introduce new preconditions. We have $C^\sigma \supseteq C$ due to condition (1) of Definition 4.5, which states that σ preserves all add effects—hence the set of actions achieving the precondition in $\mathcal{P}^\sigma$ contains the corresponding set in $\mathcal{P}$. A similar observation holds for the effect clauses in encoding (B), and similar arguments apply to all the other kinds of clauses. The claim then follows with Proposition 2.1. □

Lemma 4.7. *Let $\mathcal{P}$ be a planning task. Assume we use encoding method (C) or (D). Let σ be a reduced planning graph abstraction of $\mathcal{P}$. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.*

Proof. The argument is identical to the proof of Lemma 4.6, except that the underlying planning graph for encodings (C) and (D) is the reduced planning graph, resulting in potentially fewer mutex clauses than in encodings (A) and (B), respectively. This, however, does not in any way affect the proof arguments. □

Lemma 4.8. *Let $\mathcal{P}$ be a planning task. Let σ by any modification of $\mathcal{P}$ that respects the following behavior:*

(a) σ does not shrink the list of initial facts,

(b) σ does not grow the set of goal facts,

(c) σ preserves the add lists unchanged, and

(d) σ does not grow the pre and del lists.

Then σ is a planning graph abstraction of $\mathcal{P}$ as well as a reduced planning graph abstraction of $\mathcal{P}$.

Proof Sketch. The proof is straightforward, but a little tedious in the details. Suppose σ is an abstraction for $\mathcal{P}$ satisfying the prerequisites. We must argue that $PG^\sigma(\mathcal{P})$ and $PG^\sigma_{red}(\mathcal{P})$ satisfy the conditions in Definition 4.5. Condition (4) involving goal states easily follows from property (b) of σ. Once $PG(\mathcal{P})$ and $PG^\sigma(\mathcal{P})$ (as well as their reduced counterparts) are shown to have the same set of vertices, conditions (1) and (2) involving precondition and effect relations follow directly from properties (c) and (d). It hence remains to prove that all facts and actions available in $PG(\mathcal{P})$ are also available in $PG^\sigma(\mathcal{P})$ (showing (1) and (2) with what we just said), and that no new mutex relations are created between facts and actions that are not mutex in $PG(\mathcal{P})$ (showing (3)). This proof is a little tedious, proceeding inductively over the construction of the planning graph. The underlying intuition, however, is simple: if $PG^\sigma(\mathcal{P})$ up to layer t abstracts $PG(\mathcal{P})$ up to layer t, and properties (a), (c) and (d) are respected by σ, then necessarily $PG^\sigma(\mathcal{P})$ up to layer $t+1$ abstracts $PG(\mathcal{P})$ up to layer $t+1$. This concludes the argument. □

The following is an immediate consequence of Lemmas 4.6, 4.7, and 4.8.

Corollary 4.9. *Let $\mathcal{P}$ be a planning task. Assume we use any of the encoding methods (A)–(D). Let σ be an abstraction of $\mathcal{P}$ that consists of any combination of:*

(a) adding initial facts; and

(b) ignoring preconditions, goals, or deletes.

Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.

This result essentially states the rather intuitive fact that, if the abstraction does anything that yields a "larger" planning graph, then the resulting Graphplan-based encodings will be less constrained and hence have a higher resolution complexity (if anything).

Matters become much less intuitive once we consider abstractions that remove entire facts—clearly, these no longer result in over-approximated planning graphs, since they remove some of the vertices. In other words, the condition in Definition 4.5 that $PG^\sigma(\mathcal{P})$ and $PG(\mathcal{P})$ have identical sets of vertices does *not* hold, and we need a slightly different line of reasoning that does not rely strictly on abstracted planning graphs. We first show that it is harmless to remove a fact if it does not appear in the goal and in any *pre* or *del* list. Then we rely on Corollary 4.9 to reason that this requirement on a fact can be easily achieved.

Lemma 4.10. *Let $\mathcal{P}$ be a planning task. Assume we use any of the encoding methods (A)–(D). Let p be a fact that does not appear in the goal and in any of the pre or del lists, and let σ be the abstraction of $\mathcal{P}$ that removes p from the initial facts and the add lists. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) = \mathcal{RC}(\phi^\sigma)$.*

Proof Sketch. The key point is that, if p does not appear in the goal and is never required or deleted by an action, then p is completely irrelevant to the planning task, and in particular to the resolution refutations we consider here. Concretely, we first prove that at every layer of the planning graph, the available facts and the mutex fact pairs remain the same, up to facts or fact pairs involving p. That is, the only thing that is lost in the fact layers of

$PG^\sigma(\mathcal{P})$ is p. Since p does not occur in any precondition, the action layers remain exactly the same; since p does not occur in any preconditions or delete effects, the action mutexes also remain exactly the same (they were not caused by p).

The above discussion implies that the precondition clauses in all the encodings are identical. Given that p does not appear in the goal, the same is true of the goal clauses. Since the action mutexes are unchanged, it follows that ϕ and ϕ^σ are actually identical for encodings (A) and (C). For encodings (B) and (D), the only difference between ϕ and ϕ^σ is that ϕ^σ does not contain the initial state, effect, and mutex clauses involving p. However, these clauses can never participate in a resolution refutation of ϕ: all effect and mutex clauses contain p in the same polarity (negative); while an initial state clause has the positive form $\{p(0)\}$, the time index is different from those of p in all effect and mutex clauses. Hence every variable corresponding to p occurs in only one polarity. This concludes the argument. $\square$

Corollary 4.11. *Let $\mathcal{P}$ be a planning task. Assume we use any of the encoding methods (A)–(D). Let σ be an abstraction of $\mathcal{P}$ that removes a fact completely. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.*

Proof. σ is equivalent to the following two steps. First, remove p from the goal facts (if present) and from all *pre* and *del* lists. By Corollary 4.9, this step cannot improve resolution complexity. Second, now that p has been removed from the goal and the *pre* and *del* lists, remove p from the problem completely by removing it from the initial facts and all *add* lists as well. By Lemma 4.10, this step as well cannot improve resolution complexity, and we are done. $\square$

Corollaries 4.9 and 4.11 together prove our first main result, Theorem 4.1. We now move on to variable domain abstraction, where matters are most complicated, and which is most interesting because that abstraction method enables us to construct solution length preserving abstractions with exponentially smaller state spaces, in many benchmarks. First we show that, in its original form, the result holds for encoding (A).

Lemma 4.12. *Let $\mathcal{P}$ be a planning task. Assume we use encoding method (A). Let σ be a variable domain abstraction of $\mathcal{P}$. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ and ϕ^σ be the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively. Then $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$.*

Proof Sketch. σ combines two persistently mutex facts p and p' into a single fact p. We first show that if an action pair (a, a') is mutex in $\mathcal{P}^\sigma$, then it will also be mutex in $\mathcal{P}$. The only way for (a, a') to become mutex per σ requires, w.l.o.g., that in $\mathcal{P}$, $p \in del(a) \subseteq pre(a)$ and $p' \in pre(a') \cup add(a')$. Supposing (a, a') is not mutex in $\mathcal{P}$, we have that $p \notin del(a')$ and p is not mutex with any fact in $pre(a')$. But then, $(\textsc{noop}(p), a')$ is not mutex in $\mathcal{P}$ and hence (p, p') is not a persistent mutex, in contradiction.

With this in hand, we can derive a property rather similar to that of planning graph abstractions given in Definition 4.5. By the above, we know that the abstract encoding ϕ^σ does not have any mutexes that do not appear in ϕ. Further, the set of actions achieving each fact only grows by applying this abstraction, and the goal can only shrink. The most subtle issue regards precondition clauses. If an action a has p' as its precondition in $\mathcal{P}$, then this is replaced by p in $\mathcal{P}^\sigma$, so there is no direct correspondence between the two. However, that lack of correspondence does not affect the precondition clause of encoding (A), which takes the form $\{\neg a, a_1, \dots, a_k\}$; this omits the actual precondition fact being achieved, so it does not matter whether that fact is p' or p.

Next, as in the proof to Lemma 4.6, we set all variables to False which appear in ϕ^σ but not in ϕ. With the above arguments, it is then not difficult to see that the clauses surviving in ϕ^σ are also present in ϕ, either as is or in a stronger form (i.e., with fewer literals). For mutex clauses, this is obvious. For goal clauses, the argument is exactly the same as in the proof sketch for Lemma 4.6 given above. For precondition clauses, observe that $a_1, \dots, a_k$ in the above will contain all achievers of p' plus all achievers of p. The claim now follows with Proposition 2.1. $\square$

Corollaries 4.9 and 4.11 together with Lemma 4.12 prove our second main result, Theorem 4.2. For encodings (B)–(D), matters are more complicated.

Consider first encodings (B) and (D), which differ from (A) in that they also have fact variables. This changes the precondition clauses. If an action a in $\mathcal{P}$ has p' as its precondition, but p in $\mathcal{P}^\sigma$, then we no longer get the clause $\{\neg a, a_1, \dots, a_k\}$ as in the proof sketch. Instead, we get the clause $\{\neg a, p\}$. For this clause, there is no correspondence in ϕ. In particular, consider the case where we have two actions in $\mathcal{P}$, action a with precondition p and action a' with precondition p'. This gives us the clauses $\{\neg a, p\}, \{\neg a', p'\}$ in ϕ and the clauses $\{\neg a, p\}, \{\neg a', p\}$ in ϕ^σ. Now, ϕ^σ does not distinguish between the achievers of p and those of p', so there is no problem in that regard. But can the fact that the two clauses now share a literal—which they don't in ϕ—be exploited to obtain shorter resolution refutations? This is an open question; we discuss its implications in a little more detail at the end of this sub-section.

Consider now encoding (C), which differs from (A) in that it includes only direct action mutexes. This invalidates a different argument in the proof of Lemma 4.12. It is still true that, if an action pair (a, a') is marked mutex in $\mathcal{P}^\sigma$, then it will also be mutex in $\mathcal{P}$. However, it can happen that (a, a') is mutex in $\mathcal{P}^\sigma$ due to a direct interference between a and a', while (a, a') is mutex in $\mathcal{P}$ due to mutex preconditions, rather than a direct interference. Since encoding (C) accounts only for direct interferences, we then have a mutex in ϕ^σ that does not appear in ϕ. This can result in improved resolution complexity for ϕ^σ. The following proposition proves this formally.

Proposition 4.13. *Assume we use encoding method (C). There exist a planning task $\mathcal{P}$, a variable domain abstraction σ of $\mathcal{P}$, and $b < n$ such that $\mathcal{RC}(\phi) > \mathcal{RC}(\phi^\sigma)$, where n is the length of a shortest plan for $\mathcal{P}^\sigma$, and ϕ and ϕ^σ are the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively.*

Proof Sketch. We construct $\mathcal{P}$, σ, and b as specified. The key property of the construction is that there are two actions, $getg_1$ and $getg_2$, that are both needed to achieve the goal facts g_1 and g_2, respectively. More precisely, $getg_1 = (\{x\}, \{g_1, p'\}, \{x\})$ and $getg_2 = (\{p, y\}, \{g_2\}, \{p\})$. The task is constructed, along with the help of a few other actions, in a way so that x, p, and p' are pairwise persistently mutex. The variable domain abstraction replaces p' with p, and b is set to 2. In the action layer directly beneath the goal layer, i.e., in action layer $A(1)$, the planning graph marks $getg_1$ and $getg_2$ as mutex because their preconditions are mutex. Encoding (C), however, does not include this mutex clause because there is no direct conflict. This situation changes after the abstraction. Now $getg_1$ adds p instead of p', and hence there is a direct conflict with the delete effect of $getg_2$. As a consequence, in the abstraction two resolution steps suffice: applying both $getg_1$ and $getg_2$ in $A(1)$ is the only option to achieve the goals, and the new mutex clause immediately excludes that option. This is not so in the encoding of the original task, where the required mutex must first be derived by reasoning over the preconditions x and p. $\square$

Note that the only reason why we get a shorter refutation for ϕ^σ is that variable domain abstraction turns an indirect action mutex (due to competing preconditions) into a direct interference. In doing so, the abstraction exploits the knowledge that p and p' are persistently mutex—a fact that is ignored in encoding (C). Hence *the positive result stated by Proposition 4.13 is less related to the power of abstraction than to the power of planning graph mutexes*. We now capture this formally. In what follows, note that using the same plan length bound, the CNF formula as per encoding (C) is a sub-formula of the CNF formula as per encoding (A), and all the additional clauses of (A) can be inferred from it.

Lemma 4.14. *Let $\mathcal{P}$ be a planning task. Let σ be a variable domain abstraction of $\mathcal{P}$. Let n be the length of a shortest plan for $\mathcal{P}^\sigma$, and let $b < n$. Let ϕ_A and ϕ_C be the encodings of b-step plan existence in $\mathcal{P}$ as per encoding (A) and (C), respectively. Let ϕ^σ_C be the encoding of b-step plan existence in $\mathcal{P}^\sigma$ as per encoding (C). Let M be the number of resolution steps required to infer from ϕ_C the additional mutex clauses that appear in ϕ_A. Then $\mathcal{RC}(\phi_C) \leq \mathcal{RC}(\phi^\sigma_C) + M$.*

Proof. Denote by ϕ^σ_A the encoding of b-step plan existence in $\mathcal{P}^\sigma$ as per encoding (A). We have:

(1) By the preconditions of the lemma, $\mathcal{RC}(\phi_C) \leq \mathcal{RC}(\phi_A) + M$: with M resolution steps, ϕ_C can be turned into ϕ_A, and hence from the shortest resolution refutation for ϕ_A we can construct one for ϕ_C that is M steps longer.

(2) From Lemma 4.12, $\mathcal{RC}(\phi_A) \leq \mathcal{RC}(\phi^\sigma_A)$.

(3) ϕ^σ_C is a sub-formula of ϕ^σ_A, and hence $\mathcal{RC}(\phi^\sigma_A) \leq \mathcal{RC}(\phi^\sigma_C)$.

Combining these observations, we have:

$$\begin{aligned}
\mathcal{RC}(\phi_C) &\leq \mathcal{RC}(\phi_A) + M && \text{from observation (1)}\\
&\leq \mathcal{RC}(\phi^\sigma_A) + M && \text{from observation (2)}\\
&\leq \mathcal{RC}(\phi^\sigma_C) + M && \text{from observation (3)}
\end{aligned}$$

This finishes the proof. $\square$

Clearly, this proof argument applies also when σ is a combination of variable domain abstraction with all the other abstractions. Hence Corollaries 4.9 and 4.11 together with Lemmas 4.12 and 4.14 prove our third main result, Theorem 4.3. Note that this latter result does not hold for all variants of resolution. In the claim of Lemma 4.14, M is the number of resolution steps it takes to derive the action mutexes not present in the original encoding. These are then used in the resolution refutation. If the variant of resolution under consideration is, say, DPLL or tree-like resolution, deriving a mutex clause once is not enough—it must be re-derived as many times as it is used in the tree-like resolution refutation. Hence the effective value of M for such variants of resolution would be larger. Note that this is not the case if the DPLL solver *learns* the mutex clauses by virtue of the wide-spread clause learning technique.

Lemma 4.14 is particularly relevant for our empirical results, because SATPLAN'04 uses encoding (C) and our experiments mostly focus on variable domain abstraction. While we have no explicit empirical proof (and such a proof would be difficult to come by, requiring a deep analysis of the SAT solver's search spaces), it seems reasonable to assume that, at least to some extent, the disappointing results for SATPLAN'04 are due to what's proved in Lemma 4.14. The abstraction cannot improve resolution complexity beyond the effort required to recover the indirect action mutexes. Note here that the bound M given in the lemma is rather pessimistic. A mutex (a, a') needs to be recovered only in the case where a and a' have competing needs in $\mathcal{P}$, and replacing p' with p results in a direct interference but does not incur simplifications. In the Logistics domain, for example, with our abstraction this happens only for actions loading a package onto an airplane in two different but irrelevant cities. Since these load actions are involved only in redundant solutions anyway, it seems doubtful that such mutexes play a role for resolution complexity.

More generally, it is interesting to consider upper bounds on M in Lemma 4.14. How many resolution steps does it take to recover the indirect action mutexes? For general resolution, the number of steps is polynomially bounded, since the inference process conducted by the planning graph can be simulated (for a related investigation, see Brafman, 2001). For restricted variants of resolution, matters are more complicated. Of particular interest is the behavior of DPLL+clause learning, c.f. the above. There is so far no known formula for which DPLL+clause learning proofs are provably substantially worse than general resolution proofs; it would be rather surprising if planning graph mutexes were to be the first. Also, Rintanen (2008) provides a related investigation, showing that mutexes can be recovered in polynomial time by a particular 2-step lookahead procedure, which is related (but not identical) to clause learning.

Concluding this sub-section, let us again turn our attention to encodings (B) and (D). As mentioned before, it is an open question whether an analogue of Theorem 4.2 holds for encoding (B), and whether an analogue of Theorem 4.3 holds for encoding (D). We are facing two problematic issues:

(I) **Fact variables.** In both encoding (B) and (D), there are fact variables in addition to the action variables used in encodings (A) and (C).

(II) **Mutexes.** Like for encoding (C), it may happen that variable domain abstraction converts an implicit mutex in encoding (D) of the original task into an explicit one in the abstraction.

We consider first issue (II). The situation is exactly as for encoding (C), in this regard. Proposition 4.13 holds as stated for encoding (D) as well; indeed it can be proved using exactly the same example and only very minor adaptations of the proof arguments.[22] Similarly, Lemma 4.14 holds for encoding (D), with exactly the same proof arguments—provided an analogue of Lemma 4.12 (and hence an analogue of Theorem 4.2) holds for encoding (B). Namely, the proof arguments of Lemma 4.14 all remain valid, except that we need to refer to encoding (B) rather than (A), and that accordingly we need a corresponding version of Lemma 4.12. This brings us to issue (I).

Variable domain abstraction can be perceived as "gluing sets of facts together". Recall here the example with clauses $\{\neg a, p\}, \{\neg a', p'\}$ in ϕ not sharing any literals, and clauses $\{\neg a, p\}, \{\neg a', p\}$ in ϕ^σ sharing the literal p; this was discussed above to explain why the proof of Lemma 4.12 does not work for encodings (B) and (D). If $k + 1$ facts are glued together, then groups of k clauses can become linked together in this fashion. The question is:

(*) Can resolution fruitfully exploit this increased linkage?

This issue appears to be related to some quite intricate properties of Graphplan-based encodings with vs. without fact variables. For encoding (A), which differs from encoding (B) only in that it does not use fact variables, Lemma 4.12 tells us that resolution cannot exploit variable domain abstraction. Now, it appears reasonable to think that adding fact variables does not help a lot, the intuition being:

(**) Whatever one can do with encoding (B), one can easily simulate with encoding (A).

If statement (**) is true, then the answer to question (*) has to be "no", because a "yes" answer would contradict Lemma 4.12. Hence, in an initial attempt to prove the "no" answer, we tried to prove statement (**). However, our initial investigation has indicated that by explicitly keeping fact variables (and non-trivial constraints on them) around, encoding (B) *might* facilitate significantly shorter resolution derivations in general, and hence statement (**) might be false. Namely, there appear to be families of formulas that can be suitably encoded into planning tasks to yield an exponential separation between encodings (A) and (B). If this is true, then it suggests that reasoning in the presence of fact variables might be more powerful and hence might indeed be able to exploit the linkage gain yielded by variable domain abstraction.

Since the purpose of this paper is not to compare the relative power of various Graphplan-based encodings (such as that of (A) and (B)), we do not detail our progress towards disproving statement (**). Besides, note that, if statement (**) is indeed false, then that does not have any immediate implications on the answer to question (*). A definite answer to (*) is left open for future research.

4.2 Can Resolution Complexity Become Worse?

The answer to the title of this sub-section is a definite "yes". With all four encodings, any of the abstractions we consider may exponentially deteriorate resolution complexity. Formally, we have the following theorem.

22. We include the full proof for both (C) and (D) in Appendix A.

Theorem 4.15. *Assume we use any of the encoding methods (A)–(D). There exist an infinite sequence of planning tasks $\mathcal{P}(i)$, abstractions $\sigma(i)$ of $\mathcal{P}(i)$, and $b(i) < n(i)$ such that $\mathcal{RC}(\phi^\sigma(i))$ is exponential in i while $\mathcal{RC}(\phi(i))$ is a constant independent of i, where $n(i)$ is the length of a shortest plan for $\mathcal{P}^\sigma(i)$, $\phi(i)$ and $\phi^\sigma(i)$ are the encodings of $b(i)$-step plan existence in $\mathcal{P}(i)$ and $\mathcal{P}^\sigma(i)$ respectively, and $\sigma(i)$ consists of any one of:*

(a) adding initial facts;

(b) ignoring preconditions, goals, or deletes;

(c) removing a fact completely; or

(d) variable domain abstraction.

Proof Sketch. The idea is to construct $\mathcal{P}(i)$ as a planning task that consists of two separate sub-tasks, and whose overall goal is to achieve the goals of both of these sub-tasks. Each of the sub-tasks themselves is infeasible within the given plan length bound $b(i)$. (The tasks and bounds are constructed such that their size grows polynomially with i.) However, while the first sub-task is constructed to require exponential size resolution refutations, the second allows constant size refutations. If an abstraction over-approximates the easy-to-refute sub-task in a way so that it becomes feasible within $b(i)$ steps, then the resolution refutation of the overall task must rely on the hard-to-refute sub-task. This leads to an exponential growth, over i, in resolution complexity for $\phi^\sigma(i)$, as opposed to constant resolution complexity for $\phi(i)$. With any single one of the listed abstractions, feasibility of the easy-to-refute sub-task can be accomplished in a simple manner, hence proving the theorem.

In order to construct planning tasks whose CNF encodings require exponential size resolution refutations, we resort to the "pigeon hole problem" formula *PHP(i)*. It is well known that any resolution proof of *PHP(i)* must be of size exponential in i (Haken, 1985). We construct a simple pigeon hole planning task $\mathcal{P}_{PHP}(i)$ to capture this problem. We show that, for any of the four encoding methods (A)–(D), the CNF encoding for $b(i) = 1$ is either identical to *PHP(i)*, or transforms into *PHP(i)* by variable restrictions. Hence, by Proposition 2.1, any resolution refutation must have size exponential in i. The final construction uses a combination of two such tasks: $\mathcal{P}_{PHP}(i)$ serves as the hard-to-refute sub-task, and $\mathcal{P}_{PHP}(1)$ on disjoint sets of pigeon and hole objects serves as the easy-to-refute sub-task. $\square$

Essentially, Theorem 4.15 states the intuitive fact that abstractions can make bad choices, approximating away the most concise reason for why a planning task cannot be solved in a particular number of steps. To illustrate the significance of this, consider once more the comparison to mutex relations. An analogue of Theorem 4.15 does not hold for them: adding a mutex clause to a CNF encoding can only improve resolution complexity. In that sense, mutex relations are considerably less "risky" than the abstractions we consider here.

While the pigeon hole problem used in the proof of Theorem 4.15 may seem artificial, it is indeed contained as a sub-problem in wide-spread domains such as some of those concerned with transportation. For example, in Gripper, the available time steps serve as "holes" and the actions picking/dropping balls are the "pigeons" (for a related investigation, see

Hoffmann et al., 2007). It also seems quite natural that a planning task may consist of two disconnected parts, one of which is complex while the other one is easy to prove unsolvable in the given number of steps. Just think of transporting two packages, one of which is close to many vehicles and requires just one more step than the bound allows, while the other one is already inside a vehicle and needs to be transported along a single path of road connections that is much longer than the bound (a concrete example for the latter situation is formalized by Hoffmann et al., 2007).

4.3 A Note on Simplifications

As pointed out in Section 2.4, there may be actions that after abstraction can obviously be simplified without altering the semantics of the planning task. In particular, an abstraction might create duplicate actions (that is, actions having identical preconditions and effects), as well as redundant add effects (that are contained in the respective action's precondition). It turns out that such a natural post-abstraction simplification of the planning task *can* lead to lower resolution complexity.

Proposition 4.16. *Assume we use any of the encoding methods (A)–(D). There exist a planning task $\mathcal{P}$, a planning task $\mathcal{P}'$ that is identical to $\mathcal{P}$ except that either an irrelevant add affect or a duplicate action have been removed, and $b < n$ so that $\mathcal{RC}(\phi) > \mathcal{RC}(\phi')$, where n is the length of a shortest plan in $\mathcal{P}$, and ϕ and ϕ' are the encodings of b-step plan existence in $\mathcal{P}$ and $\mathcal{P}'$, respectively.*

Proof Sketch. To show the claim for duplicate actions, we consider a task $\mathcal{P}'$ encoding the pigeon hole problem for 3 pigeons and 2 holes, with actions that put pigeon p into hole h. The plan length bound is 1. To point out that the proof works for solvable tasks $\mathcal{P}$, an extra action a, whose preconditions are two of the goals and which achieves the third goal, ensures solvability in two steps. There are three goals, one for each pigeon, and there are no mutex clauses other than direct action interference, because each pair of goals can be achieved – but not the three of them. In particular, every resolution refutation must resolve on all three goal clauses. We obtain $\mathcal{P}$ by adding a duplicate action for one of the pigeons and one of the holes. The respective goal clause then becomes one literal longer. Any refutation must get rid of that literal, hence necessitating one more step. The construction works for all four encodings.

To show the claim for removal of redundant add effects, we slightly modify $\mathcal{P}'$, replacing a's effect with a new fact x and including another action that achieves the third goal given the precondition x. The optimal plan length now is 3, and the length bound is 2. Any refutation must resolve on all three goal clauses. If, for $\mathcal{P}$, we give a one of its preconditions as an additional add effect, then the refutations become longer because the respective goal clause does. Again, the construction works for all four encodings. □

It is easy to modify the constructions used in the proof of Proposition 4.16 in a way so that the duplicate action, respectively the redundancy of the add effect, arise as an outcome of variable domain abstraction. Hence, via enabling such simplifications, variable domain abstraction may improve the resolution best-case behavior. For duplicate actions, this is also true for Edelkamp's (2001) pattern database abstraction. It is an open question whether the improvement may be exponential, or whether it is bounded polynomially. We conjecture that the latter is the case, at least for unrestricted resolution.

It is also notable that the examples in the proof to Proposition 4.16 are specifically constructed to include duplicate actions/redundant add effects for actions that are relevant to solving the problem – they form part of an optimal solution. In a well-constructed variable domain abstraction this is not likely to happen since abstraction should target only the facts that are irrelevant to solution length. Consider the Logistics domain as an example. The only actions on which simplifications apply are loads/unloads of packages to/from locations in cities other than the package's origin and destination. These actions are involved only in redundant solutions, and it seems doubtful that their simplification affects resolution complexity. Of course, these simplifications might help SAT solvers anyway. This, however, is not observed, at least not significantly, in our experiments.

5. Conclusion

Abstractions, as used here, have the power to allow proving certain properties within much smaller state spaces. In particular, if an abstraction preserves the length of an optimal solution, then optimality can be proved within the abstraction. We designed a novel abstraction method for STRIPS planning that is suitable for this purpose. Surprisingly, the approach yields little or no benefits for the planning-as-satisfiability approach as represented by SATPLAN, even in domains featuring hand-made abstractions with exponentially smaller state spaces. Towards explaining this, we have shown that, in many cases, our abstraction method (as well as some other commonly used abstractions) lacks the ability to introduce shorter resolution refutations—other than through exploiting mutexes, or enabling certain post-abstraction simplifications. In contrast, we have shown that these abstractions may exponentially increase the size of the shortest resolution refutations.

Several questions are left open by our theoretical results. We do not know whether variable domain abstraction can improve resolution complexity in combination with encoding (B), whether there is a polynomial upper bound on the improvement that variable domain abstraction can bring for encoding (D), and whether there is a polynomial upper bound on the improvement that can result from simplifications. Apart from answering these questions, most importantly it remains to be seen to what extent the results generalize. Bluntly stated, the intuition behind the results is that "over-approximations usually result in less constrained formulas which are harder to refute". However, the actual technicalities of the results depend quite a lot on the details—of both encoding method and abstraction—and hence it is largely unclear to what extent this intuitive statement captures reality. In particular: does it hold for other encodings of planning into SAT?

It would be interesting, e.g., to look at the alternative encodings described by Kautz and Selman (1992), Kautz et al. (1996), Kautz and Selman (1996), Ernst, Millstein, and Weld (1997). Many of these encodings are based on unit clauses for initial and goal state, and action clauses stating that an action implies each of its effects and preconditions. With such a structure—and the lack of a mechanism such as a planning graph that propagates changes—some of the properties proved herein are obvious. Removing goals or initial state facts corresponds directly to removing clauses; the same is true for preconditions. Removing a fact completely may in some cases simply correspond to removing all clauses that mention the fact. Hence, for these encodings, it seems that proving our results might indeed be comparatively easy. A more challenging subject may be some more recent developments, such as the encodings by Rintanen et al. (2006) which often give substantial speed-ups through novel notions of parallelity, the encodings by Chen et al. (2009) which introduce long-distance mutex relations, or the encodings by Robinson et al. (2008) which make use of effective operator splitting and factoring methods.

More generally: do our results hold for methods employed in other fields? In particular, do they hold in model-checking, where both abstraction (e.g., Graf & Saïdi, 1997; Clarke et al., 2003) and SAT encodings (e.g., Clarke, Biere, Raimi, & Zhu, 2001; Prasad, Biere, & Gupta, 2005) are highly successful? There is one example where it is actually obvious that our results hold. Gupta and Strichman (2005) abstract by ignoring clauses in a CNF encoding of the original transition system (the motivation is that the much smaller CNF formula causes less overhead in the SAT solver).

Most ambitiously: can we define a generic framework with formal notions of "declarative transition systems", "CNF encodings", and "abstractions" that are suitable to capture our results, and prove more generic statements? All these questions appear to be worthwhile research challenges. Indeed, we think that a key contribution of our work may lie in *asking* the question about resolution complexity with vs. without an abstraction.

From a more practical perspective, we see mainly four lines of further research. First, an important question is whether our observations carry over to modified/extended planning-as-SAT systems, such as as the one by Ray and Ginsberg (2008) which guarantees plan optimality through branching restrictions within a single SAT call, rather than through calling the SAT solver iteratively. Second, it remains open to explore whether very different abstraction techniques—based e.g. on predicate abstraction—can be suitably adapted to planning. Third, it is important to note that our empirical results are not entirely negative. Mips.BDD is often substantially improved, even up to the point where, as in Figure 1, this optimal sequential planner is highly competitive with a strong optimal parallel planner such as SATPLAN, and this in a highly parallel domain such as Logistics. This is a direction that may well be worth exploring in more depth. Finally, more effective abstraction methods may exist for unsolvable examples, and could potentially play a crucial role in over-subscription planning (Sanchez & Kambhampati, 2005; Meuleau et al., 2006).

Acknowledgments

We thank the anonymous reviewers, whose detailed comments helped greatly to improve the paper. A preliminary version of this work appeared at ICAPS'06, the 16^{th} International Conference on Automated Planning and Scheduling (Hoffmann, Sabharwal, & Domshlak, 2006). The work of Carmel Domshlak was supported by Israel Science Foundation (ISF) Grants 2008100 and 2009580, as well as by the C. Wellner Research Fund. For part of this work, Jörg Hoffmann was employed at Max Planck Institute for Computer Science, Saarbrücken, Germany, and at SAP Research, Karlsruhe, Germany. The work of Ashish Sabharwal was supported by IISI, Cornell University (AFOSR Grant FA9550-04-1-0151), the National Science Foundation (NSF) Expeditions in Computing award (Computational

Sustainability Grant 0832782), NSF IIS award (Grant 0514429), and the Defense Advanced Research Projects Agency (DARPA, REAL Grant FA8750-04-2-0216).

Appendix A. Proof Details

Proof of Proposition 2.1. Suppose the sequence τ of transformations consists of ℓ restrictions, $\tau_1, \tau_2, \ldots, \tau_\ell$. Further, let τ' be the "strengthening" transformation that replaces each clause of $\phi^\sigma|_\tau$ with a (not necessarily strict) sub-clause that is also a clause of ϕ; such a transformation exists because of the assumptions in the proposition. Observe that these $\ell + 1$ transformation steps together convert ϕ^σ into a (not necessarily strict) sub-formula of ϕ. We will show that each of these $\ell + 1$ transformation steps individually does not increase the resolution complexity of the underlying formula. Without loss of generality, we will prove this fact for a single restriction transformation and then for a generic strengthening transformation. Since each of these will individually be shown to not increase the resolution complexity of the formula, they can be applied in any sequence or combination, any number of times, without increasing the resolution complexity. This would prove, in particular, that the resolution complexity of a sub-formula of ϕ is no more than that of ϕ^σ, implying that the resolution complexity of ϕ itself is no more than that of ϕ^σ (as additional initial clauses cannot hurt a resolution refutation), as desired.

We start with a single *restriction transformation* $x \leftarrow y$. For ease of notation, we will assume that the initial formula is $F = \{C_1, C_2, \ldots, C_m\}$ and the resulting simplified formula after the transformation is $F' = \{C'_1, C'_2, \ldots, C'_{m'}\}$, with $m' \leq m$. Without loss of generality, we will assume that no C'_i equals the empty clause $\{\}$ and there are no duplicate clauses in F'. Let $\pi = (C_1, C_2, \ldots, C_m, C_{m+1}, \ldots, C_M = \{\})$ be a resolution refutation of F of the smallest possible size; note that π involves m initial clauses and $M - m$ resolution steps. From π, we construct a resolution refutation π' of F' which will be of size no larger than that of π. We do this in the following three steps.

Step 1. Transform π into $\hat{\pi} = (\hat{C}_1, \ldots, \hat{C}_m, \hat{C}_{m+1}, \ldots, \hat{C}_M = \{\})$, where $\hat{C}_i$ is defined as follows. If the application of the transformation $x \leftarrow y$ results in C_i containing True or a variable and its negation, then $\hat{C}_i$ equals True; if it results in C_i containing False or duplicate literals, $\hat{C}_i$ consists of C_i with False or a duplicate literal removed; otherwise $\hat{C}_i = C_i$. Note that $\hat{C}_i$ does not contain x and is either True, the empty clause, or a non-empty (not necessarily strict) sub-clause of C. The key property here is that $\hat{C}_i$ is still a logical implicant of $\hat{C}_j$ and $\hat{C}_k$ if C_i was derived by resolving C_j and C_k in the original proof π. $\hat{C}_i$ may not necessarily be a usual resolution resolvent of $\hat{C}_j$ and $\hat{C}_k$, which is what the next two steps will fix.

Step 2. Transform $\bar{\pi}$ into $\bar{\pi} = (\bar{C}_1, \ldots, \bar{C}_m, \bar{C}_{m+1}, \ldots, \bar{C}_M = \{\})$, where $\bar{C}_i$ equals $\hat{C}_i$ for $i \leq m$ and for $i > m$ is defined sequentially, for increasing i, as follows. Suppose C_i was derived in π by resolving clauses C_j and C_k, where $j < k < i$. Assume without loss of generality that we have already defined $\bar{C}_j$ and $\bar{C}_k$. If $\hat{C}_i$ equals True, then $\bar{C}_i$ equals True as well; otherwise, if one of the two clauses $\bar{C}_j$ and $\bar{C}_k$ equals True, then $\bar{C}_i$ equals the other clause; otherwise, $\bar{C}_j$ and $\bar{C}_k$ can be resolved together on some variable and $\bar{C}_i$ is the resolvent of these two clauses. The key property here, which can be seen easily by considering the sequential nature of the transformation, is that each $\bar{C}_i$ is either True or a (not necessarily proper) sub-clause of $\hat{C}_i$; in particular, $\bar{C}_M = \{\}$. Further, each $\bar{C}_i$ that does not equal True is either the resolution resolvent of $\bar{C}_j$ and $\bar{C}_k$, or equals $\{\}$ with one of $\bar{C}_j$ and $\bar{C}_k$ also being $\{\}$.

Step 3. Finally, transform $\bar{\pi}$ into π' by simply removing any "clauses" that equal True or occur previously in the clause sequence, and stopping the sequence as soon as the first empty clause is encountered. By construction, π, $\hat{\pi}$, and $\bar{\pi}$ have exactly M clauses each and π' has no more than M clauses. Further, the first m' clauses of π' are exactly the clauses of F' and π' is a resolution refutation starting from these initial clauses, as desired.

We now consider the *strengthening transformation* τ', which essentially replaces each clause of the formula with a sub-clause (thereby "strengthening" this clause). We will show that applying τ' does not increase the resolution complexity of the underlying formula. Again, for ease of notation, let the initial formula be $F = \{C_1, C_2, \ldots, C_m\}$ with a resolution refutation $\pi = (C_1, C_2, \ldots, C_m, C_{m+1}, \ldots, C_M = \{\})$ of the smallest possible size. The argument here is along the same lines as but simpler than for restriction transformations.

Transform π into $\bar{\pi} = (\bar{C}_1, \ldots, \bar{C}_m, \bar{C}_{m+1}, \ldots, \bar{C}_M = \{\})$, where for $i \leq m$, $\bar{C}_i$ equals the sub-clause of C_i that τ' maps C_i to, and for $i > m$, $\bar{C}_i$ is defined sequentially, for increasing i, as follows. Suppose C_i was derived in π by resolving clauses C_j and C_k on variable x, where $j < k < i$. Assume without loss of generality that we have already defined $\bar{C}_j$ and $\bar{C}_k$. If x is present in both $\bar{C}_j$ and $\bar{C}_k$, then $\bar{C}_i$ is simply the resolution resolvent of these two clauses; otherwise, if x is not present in $\bar{C}_j$, then $\bar{C}_i$ equals $\bar{C}_j$; otherwise x must not be present in $\bar{C}_k$ and we set $\bar{C}_i$ to equal $\bar{C}_k$. The key property here, which can again be seen easily by considering the sequential nature of the transformation, is that each $\bar{C}_i$ is a (not necessarily proper) sub-clause of C_i; in particular, $\bar{C}_M = \{\}$. Further, each $\bar{C}_i$ is either $\bar{C}_j$ or $\bar{C}_k$ or the resolution resolvent of the two. Now transform $\bar{\pi}$ into π' by simply removing any clauses that occur previously in the clause sequence. By construction, π and $\bar{\pi}$ have exactly M clauses each and π' has no more than M clauses. Further, the first m' clauses of π' are exactly the clauses of F' and π' is a resolution refutation starting from these initial clauses, as desired. □

Proof of Lemma 4.6. Let $\mathcal{P}$ be a planning task to which an abstraction σ that abstracts $PG(\mathcal{P})$ is applied. Let ϕ, ϕ^σ denote propositional encodings of $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively, where we use either the action-only encoding (A) or the action-fact encoding (B) for both ϕ and ϕ^σ. Let U denote the set of variables in ϕ^σ that are not variables of ϕ. Finally, let τ be the variable restriction that sets every variable in U to False. In this setting, the propositional formula $\phi^\sigma|_\tau$ is nothing but the CNF encoding of the planning graph $PG^\sigma(\mathcal{P})$ (using the same encoding method, (A) or (B), as used for ϕ and ϕ^σ). In particular, all clauses of ϕ^σ that correspond to actions and facts not in $PG(\mathcal{P})$ are trivially satisfied by τ, because they all contain the negation of a variable in U, which is set to False by τ. Call the remaining, yet unsatisfied clauses in $\phi^\sigma|_\tau$ the *surviving* clauses. We will argue that each surviving clause is already present, perhaps in a stronger but not a weaker form, in ϕ itself, showing that it can be no easier to prove $\phi^\sigma|_\tau$ unsatisfiable than it is to prove ϕ itself unsatisfiable.

First consider encoding (A). By conditions (2) and (4) of Definition 4.5, every surviving precondition and goal clause in $\phi^\sigma|_\tau$ has a corresponding clause in ϕ itself. Further, by condition (1) concerning which facts are added by which action, each such precondition or goal clause in $\phi^\sigma|_\tau$ contains as a sub-clause the corresponding clause in ϕ. Finally, each surviving mutex clauses in $\phi^\sigma|_\tau$, by condition (3), is also present as a mutex clause in ϕ. From these observations, it follows that every surviving clause in $\phi^\sigma|_\tau$ contains as a (possibly non-strict) sub-clause the corresponding clause in ϕ. Applying Proposition 2.1, we obtain $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$, finishing the proof for encoding (A).

Now consider encoding (B). First, because by Definition 4.5 $PG^\sigma(\mathcal{P})$ and $PG(\mathcal{P})$ have identical sets of vertices, in particular the fact vertices $F(0)$ are the same, and hence each surviving initial state clause in $\phi^\sigma|_\tau$ is also present as an initial state clause in ϕ. Further, precondition clauses are binary and, by condition (2) of Definition 4.5, each surviving precondition clause in $\phi^\sigma|_\tau$ is also present as a precondition clause in ϕ. Similarly, each goal clause is a unit clause and, by condition (4), each surviving goal clause in $\phi^\sigma|_\tau$ is also present as a goal clause in ϕ. In a similar vein, each surviving mutex clause in $\phi^\sigma|_\tau$ is also present as a mutex clause in ϕ. Finally, each surviving effect clause in $\phi^\sigma|_\tau$, by condition (1), contains as a sub-clause the corresponding effect clause in $\phi^\sigma|_\tau$. Hence we again see that every surviving clause in $\phi^\sigma|_\tau$ contains as a (possibly non-strict) sub-clause the corresponding clause in ϕ. Applying Proposition 2.1 as before, we obtain $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$, finishing the proof for encoding (B). $\square$

Proof of Lemma 4.8. Let $\mathcal{P}$ be a planning task to which an abstraction σ that respects the behavior specified in the lemma is applied. We will show that the four conditions in Definition 4.5 hold for both $PG^\sigma(\mathcal{P})$ and $PG^\sigma_{red}(\mathcal{P})$. Observe that condition (4) in Definition 4.5 trivially holds because of property (b) of σ. We therefore focus on showing that $V(PG(\mathcal{P})) = V(PG^\sigma(\mathcal{P}))$, $V(PG_{red}(\mathcal{P})) = V(PG^\sigma_{red}(\mathcal{P}))$, and that conditions (1)–(3) in Definition 4.5 hold. In fact, once we show that $PG^\sigma(\mathcal{P})$ and $PG^\sigma_{red}(\mathcal{P})$ have the same set of vertices as the original (reduced) planning graph, conditions (1) and (2) of Definition 4.5 would be immediately satisfied due to properties (c)[23] and (d) of σ, and all that would remain would be condition (3), saying that no new mutex clauses are added by applying σ.

Hence, our task is reduced to proving that the following four new properties hold at each step $t \in 0, 1, \ldots, b$ of the planning task:

(i) $F^\sigma(t) \supseteq F(t)$,

(ii) $A^\sigma(t) \supseteq A(t)$,

(iii) $E^\sigma_{f\text{-}mutex}(t)|_{F(t)} \subseteq E_{f\text{-}mutex}(t)$, and

(iv) $E^\sigma_{a\text{-}mutex}(t)|_{A(t)} \subseteq E_{a\text{-}mutex}(t)$.

where $F(t), A(t), E_{f\text{-}mutex}(t)$, and $E_{a\text{-}mutex}(t)$ denote the sets of facts, actions (including NOOPs), fact mutexes, and action mutexes generated for $\mathcal{P}$ at step t, and the σ-versions of these denote the corresponding sets for $\mathcal{P}^\sigma$. For $\Gamma \subseteq F^\sigma(t)$, $E^\sigma_{f\text{-}mutex}(t)|_\Gamma$ denotes $\{(f_1, f_2) \in$

23. What one needs here is only that *add* lists do not shrink by applying σ. However, another argument will shortly require that *add* lists do not grow either, justifying the strict requirement of property (c).

$E^\sigma_{f\text{-}mutex}(t) \mid f_1, f_2 \in \Gamma\}$, the subset of $E^\sigma_{f\text{-}mutex}(t)$ restricted to the facts in Γ. $E^\sigma_{a\text{-}mutex}(t)|_\Gamma$ is defined similarly for $\Gamma \subseteq A^\sigma(t)$.

In order to prove that these four properties hold, we give an inductive argument on t, alternating between $F(t)$ and $E_{f\text{-}mutex}(t)$ on one hand, and $A(t)$ and $E_{a\text{-}mutex}(t)$ on the other. For the base case of $t = 0$, note that $F^\sigma(0) \supseteq F(0)$ by property (a) of σ and $E^\sigma_{f\text{-}mutex}(0)|_{F(0)} \subseteq E_{f\text{-}mutex}(0)$ because both of these sets are empty.

In words, our goal is to prove that if certain facts and actions are available in $\mathcal{P}$ at a certain step, they remain available in $\mathcal{P}^\sigma$. Similarly, if two facts or actions are mutually compatible in $\mathcal{P}$, they remain mutually compatible in $\mathcal{P}^\sigma$. This seems intuitively justifiable given the properties of σ. The following argument formalizes this intuition. In terms of notation, we will use pre, del, and add to specify actions in $\mathcal{P}$ and pre^σ, del^σ, and add^σ to specify actions in $\mathcal{P}^\sigma$.

For the first part of the inductive step, suppose $F^\sigma(t) \supseteq F(t)$ and $E^\sigma_{f\text{-}mutex}(t)|_{F(t)} \subseteq E_{f\text{-}mutex}(t)$. We will show that $A^\sigma(t) \supseteq A(t)$ and $E^\sigma_{a\text{-}mutex}(t)|_{A(t)} \subseteq E_{a\text{-}mutex}(t)$, inductively proving conditions (ii) and (iv).

Let $a \in A(t)$. Then $pre^\sigma(a) \subseteq F(t) \subseteq F^\sigma(t)$. Further, $\forall f, f' \in pre^\sigma(a) \subseteq pre(a) \subseteq F(t)$, we have that $(f, f') \notin E_{f\text{-}mutex}(t) \supseteq E^\sigma_{f\text{-}mutex}(t)|_{F(t)}$ and hence $(f, f') \notin E^\sigma_{f\text{-}mutex}(t)$. Therefore $a \in A^\sigma(t)$, proving that $A^\sigma(t) \supseteq A(t)$.

Now let $a, a' \in A(t)$ be such that $(a, a') \notin E_{a\text{-}mutex}(t)$. For the reduced planning graph with only direct mutexes, we immediately have that $(a, a') \notin E^\sigma_{a\text{-}mutex}(t)$ due to properties (c) and (d) of σ, and we are done proving that $E^\sigma_{a\text{-}mutex}(t)|_{A(t)} \subseteq E_{a\text{-}mutex}(t)$. Otherwise, for more general mutexes, several things hold. First, we have $pre^\sigma(a) \subseteq pre(a) \subseteq F(t) \subseteq F^\sigma(t)$, and the same for a'. Likewise, we have $del^\sigma(a) \subseteq del(a)$ and $del^\sigma(a') \subseteq del(a')$. Finally, by property (c), we have $add^\sigma(a) \subseteq add(a)$. Hence (c.1) $(pre^\sigma(a) \cup add^\sigma(a)) \cap del^\sigma(a') \subseteq (pre(a) \cup add(a)) \cap del(a') = \emptyset$; the last equality holds because $(a, a') \notin E_{a\text{-}mutex}(t)$. Similarly, (c.2) $(pre^\sigma(a') \cup add^\sigma(a')) \cap del^\sigma(a) = \emptyset$. Finally, for all $f \in pre^\sigma(a) \subseteq pre(a) \subseteq F(t) \subseteq F^\sigma(t)$ and for all $f' \in pre^\sigma(a') \subseteq pre(a') \subseteq F(t) \subseteq F^\sigma(t)$, we have that $(f, f') \notin E_{f\text{-}mutex}(t) \supseteq E^\sigma_{f\text{-}mutex}(t)|_{F(t)}$, which implies (c.3) $(f, f') \notin E^\sigma_{f\text{-}mutex}(t)$. From (c.1), (c.2), and (c.3), we have $(a, a') \notin E^\sigma_{a\text{-}mutex}(t)$, proving that $E^\sigma_{a\text{-}mutex}(t)|_{A(t)} \subseteq E_{a\text{-}mutex}(t)$.

For the second part of the inductive step, suppose $A^\sigma(t) \supseteq A(t)$ and $E^\sigma_{a\text{-}mutex}(t)|_{A(t)} \subseteq E_{a\text{-}mutex}(t)$. We will show that $F^\sigma(t+1) \supseteq F(t+1)$ and $E^\sigma_{f\text{-}mutex}(t+1)|_{F(t+1)} \subseteq E_{f\text{-}mutex}(t+1)$, proving conditions (i) and (iii).

Let $f \in F(t+1)$. Then $f \in \bigcup_{a \in A(t)} add(a) \subseteq \bigcup_{a \in A^\sigma(t)} add^\sigma(a)$. (Recall that NOOP actions are included in $A(t)$, so that we need not explicitly include $F(t)$ in $F(t+1)$.) It follows that $f \in F^\sigma(t+1)$, proving that $F^\sigma(t+1) \supseteq F(t+1)$.

Now let $f, f' \in F(t+1)$ be such that $(f, f') \notin E_{f\text{-}mutex}(t+1)$. Then there must exist $a, a' \in A(t) \subseteq A^\sigma(t)$ such that (c.1) $f \in add(a) \subseteq add^\sigma(a)$, (c.2) $f' \in add(a') \subseteq add^\sigma(a)$, and $(a, a') \notin E_{a\text{-}mutex}(t) \supseteq E^\sigma_{a\text{-}mutex}(t)|_{A(t)}$, which implies that (c.3) $(a, a') \notin E^\sigma_{a\text{-}mutex}(t)$. From (c.1), (c.2), and (c.3), we have $(f, f') \notin E^\sigma_{f\text{-}mutex}(t+1)$, proving that $E^\sigma_{f\text{-}mutex}(t+1)|_{F(t+1)} \subseteq E_{f\text{-}mutex}(t+1)$.

This finishes the inductive argument, showing that conditions (i)–(iv) outlined above hold. By our earlier reasoning, this proves both that the vertices of $PG(\mathcal{P})$ and $PG^\sigma(\mathcal{P})$, as well as their reduced counterparts, are the same (so that conditions (1) and (2) of Definition 4.5 follow directly from properties (c) and (d) of σ) and the mutex relations of $\mathcal{P}^\sigma$ and $\sigma_{red}(\mathcal{P})$, restricted to the facts and actions of $\mathcal{P}$, are a subset of the mutex relations and the reduced mutex relations, respectively, of $\mathcal{P}$ (so that condition (3) of Definition 4.5 holds). Hence σ abstracts the planning graph as well as the reduced planning graph of $\mathcal{P}$. $\square$

Proof of Lemma 4.10. Let σ be the abstraction that removes p from the initial facts and the *add* lists of a given planning task $\mathcal{P}$ in which p does not appear in the goal facts and any of the *pre* or *del* lists. As in the proof of Lemma 4.6, let ϕ, ϕ^σ denote propositional encodings of $\mathcal{P}$ and $\mathcal{P}^\sigma$, respectively, where we use one of the encodings (A), (B), (C), and (D) for both ϕ and ϕ^σ. We will show that for encodings (A) and (C), ϕ^σ and ϕ are in fact identical, and for encodings (B) and (D), differ only in clauses that cannot be part of any resolution proof.

To this end, we use the planning graph notation from the proof of Lemma 4.8 and begin by arguing by induction that $F^\sigma(t) = F(t) \setminus \{p\}, E^\sigma_{\textit{f-mutex}}(t) = E_{\textit{f-mutex}}(t) \setminus \{(p, p') \mid p' \in F(t)\}, A^\sigma(t) = A(t)$, and $E^\sigma_{\textit{a-mutex}}(t) = E_{\textit{a-mutex}}(t)$. For the base case of $t = 0$, $F^\sigma(0) = F(0) \setminus \{p\}$ by the definition of σ, and $E^\sigma_{\textit{f-mutex}}(t) = E_{\textit{f-mutex}}(0) \setminus \{(p, p') \mid p' \in F(0)\}$ because both these sets are empty. For the first part of the induction, suppose that the inductive conditions on F and $E_{\textit{f-mutex}}$ hold at time step t. Since p is not in any *pre* list, this implies that $A^\sigma(t) = A(t)$ as well. Further, since p is not in any *del* list, we also have $E^\sigma_{\textit{a-mutex}}(t) = E_{\textit{a-mutex}}(t)$. Hence the conditions on A and $E_{\textit{a-mutex}}$ hold at time step t. For the second part of the induction, suppose that the inductive conditions on A and $E_{\textit{a-mutex}}$ hold at time step t. This implies that $F(t+1)$ consists of $F^\sigma(t+1)$ and possibly p. Further, $E_{\textit{f-mutex}}(t+1)$ and $E^\sigma_{\textit{f-mutex}}(t+1)$ are the same as far as mutexes not involving p are concerned. It follows that the conditions on F and $E_{\textit{f-mutex}}$ hold at time step $t+1$, finishing the induction.

To summarize, we have shown that at every step, the sets A, A^σ and $E_{\textit{a-mutex}}, E^\sigma_{\textit{a-mutex}}$ are exactly the same, and the sets F, F^σ and $E_{\textit{f-mutex}}, E^\sigma_{\textit{f-mutex}}$ are the same up to facts or pairs of facts involving p. In other words, no new actions or facts become available or are mutually excluded in the planning graph because of σ, and everything not involving p remains unchanged. Given this, observe that with any of the four encodings, the goal and precondition clauses of $\mathcal{P}^\sigma$ are exactly the same as those of $\mathcal{P}$ because p does not appear in the goal or the *pre* lists at all. Similarly, $E^\sigma_{\textit{a-mutex}}(t) = E_{\textit{a-mutex}}(t)$ and $E^\sigma_{\textit{f-mutex}}(t) = E_{\textit{f-mutex}}(t)$ implies that the action mutexes of $\mathcal{P}^\sigma$, and fact mutexes if present in the encoding, are the same as those of $\mathcal{P}$ as well. Therefore, for encodings (A) and (C), $\phi^\sigma = \phi$.

Finally, for encodings (B) and (D), we have initial state, effect, and mutex clauses in ϕ which do get removed by applying σ, i.e., which are not present in ϕ^σ. However, these are the only clauses that mention propositional variables corresponding to p, and each of these variables appears only with one polarity throughout ϕ. Namely, an initial state clause $\{p(0)\}$ is the only clause that may contain p in positive polarity; all effect and mutex clauses that contain p do so with a time index $t > 0$. Because of that, these clauses cannot be part of any resolution refutation of ϕ—every variable appearing in a resolution refutation must eventually be resolved away in order to derive the empty clause. It follows that ϕ^σ and ϕ are the same with respect to resolution refutations. $\square$

Proof of Lemma 4.12. Let $\mathcal{P}$ be a planning task to which σ, a variable domain abstraction, is applied. σ combines two persistently mutex facts p and p' into a single fact p. For brevity, let $\mathcal{G}$ denote $PG(\mathcal{P})$. Define $\mathcal{G}'$ to be the graph obtained by unifying any p and

p' fact vertices in each fact layer of $\mathcal{G}$ into a single vertex p in that layer, and similarly any $\textsc{noop}(p)$ and $\textsc{noop}(p')$ vertices in action layers. Finally, let $\mathcal{G}^\sigma$ denote the subgraph of $PG(\mathcal{P}^\sigma)$ induced by the vertices of $\mathcal{G}'$. We will show that $\mathcal{G}^\sigma$ is an abstracted planning graph in a sense very similar to Definition 4.5.

We begin by arguing that if an action pair (a, a') is mutex in $\mathcal{P}^\sigma$, then it is also mutex in $\mathcal{P}$. To see this, observe that the only way for (a, a') to become mutex per σ requires, w.l.o.g., that in $\mathcal{P}$, $p \in del(a) \subseteq pre(a)$ and $p' \in pre(a') \cup add(a')$. Suppose for the sake of contradiction that (a, a') is not already mutex in $\mathcal{P}$. In particular, this means that $p \notin del(a')$ and p is not mutex with any fact in $pre(a')$. This, however, implies that $(\textsc{noop}(p), a')$ is not mutex in $\mathcal{P}$ so that the fact pair (p, p') is not mutex in the next layer of $PG(\mathcal{P})$, a contradiction because p and p' are persistently mutex. It follows that edges $E_{a\text{-}mutex}$ in $\mathcal{G}^\sigma$ are a subset of those in $\mathcal{G}$.

Since $\mathcal{G}'$ and $\mathcal{P}^\sigma$ have the same initial facts, the above argument implies that all actions and facts available at any layer of $\mathcal{G}'$ are also available at that layer of $PG(\mathcal{P}^\sigma)$. In particular, $\mathcal{G}^\sigma$, by construction, has exactly the same set of vertices as $\mathcal{G}'$. Further, since σ is a variable domain abstraction, edges E_{add} and E_{pre} in $\mathcal{G}'$ and $\mathcal{G}^\sigma$ are exactly the same.

Define ϕ, ϕ^σ, U, and τ as in the proof of Lemma 4.6. $\phi^\sigma|_\tau$ is the CNF encoding (A) of the planning graph $\mathcal{G}^\sigma$, and all clauses of ϕ^σ corresponding to actions and facts not in $\mathcal{G}^\sigma$ are trivially satisfied by τ. Call the remaining clauses in $\phi^\sigma|_\tau$ the *surviving* clauses as before.

By our observation about edges $E_{a\text{-}mutex}$ in $\mathcal{G}^\sigma$ and $\mathcal{G}$, the surviving mutex clauses of $\phi^\sigma|_\tau$ are also mutex clauses of ϕ. The surviving precondition and goal clauses not involving p appear unchanged in ϕ. Since we are considering a variable domain abstraction, the actions achieving p in $\mathcal{G}^\sigma$ are precisely the actions achieving either of p and p' in $\mathcal{G}$. Hence, the surviving precondition and goal clauses of $\phi^\sigma|_\tau$ involving p contain as a sub-clause a precondition or goal clause of ϕ itself. It follows from Proposition 2.1 that $\mathcal{RC}(\phi) \leq \mathcal{RC}(\phi^\sigma)$. □

Proof of Proposition 4.13. The same example planning task, denoted $\mathcal{P}$, works for both encoding (C) and encoding (D). Let σ denote the variable domain abstraction to be applied. The example uses the following six facts: facts p and p', which will be glued together by σ; goal facts g_1 and g_2; and helper facts x, y. Based on these facts, the task $\mathcal{P}$ is defined as follows:

- Initial state $\{p\}$; Goal $\{g_1, g_2\}$
- Action set A containing five actions:
  $getx = (\{p\}, \{x\}, \{p\})$,
  $gety = (\emptyset, \{y\}, \emptyset)$,
  $getg_1 = (\{x\}, \{g_1, p'\}, \{x\})$,
  $getg_2 = (\{p, y\}, \{g_2\}, \{p\})$,
  $getp = (\{p'\}, \{p\}, \{p'\})$.

The plan length bound is 2, which makes the problem infeasible: the shortest (parallel) plan requires 4 steps: $\langle \{getx, gety\}, \{getg_1\}, \{getp\}, \{getg_2\} \rangle$. Observe that all pairs of actions—except $(getg_1, getg_2)$ and any pair that involves $gety$—directly interfere with each other and are therefore mutex. In $PG(\mathcal{P})$, we get the following fact and action sets up to step 2:

- $F(0) = \{p\}$, $A(0) = \{\textsc{noop}(p), getx, gety\}$
- $F(1) = \{p, x, y\}$, $A(1) = \{\textsc{noop}(p), \textsc{noop}(x), \textsc{noop}(y), getx, gety, getg_1, getg_2\}$
- $F(2) = \{p, x, y, g_2, g_1, p'\}$

It is easy to verify, iteratively, that, in $PG(\mathcal{P})$: p and x are mutex in $F(1)$; p and x are mutex in $F(2)$; p and p' are mutex in $F(2)$; x and p' are mutex in $F(2)$; all these mutexes we get also in $F(3)$, where the planning graph reaches its fixpoint. In particular, we have that $getg_1$ and $getg_2$ are always (indirectly) mutex because their preconditions x and p are persistently mutex. The variable domain abstraction σ will glue p and p', converting the conflict between $getg_1$ and $getg_2$ into a direct interference, thereby allowing a shorter resolution refutation.

Consider encoding (C) of $PG(\mathcal{P})$. It contains two goal clauses: $\{getg_1(1)\}$ and $\{getg_2(1)\}$. These clauses clearly must be used in any resolution refutation of the formula, because it is possible to achieve each goal individually within the given time bound, but not both together. Hence, a shortest refutation must involve at least two steps. We argue that such a shortest refutation *can* be achieved in the abstracted task $\mathcal{P}^\sigma$ but not in $\mathcal{P}$ itself.

In $\mathcal{P}^\sigma$, we get the same fact and action sets in the planning graph, except that $F(2) = \{p, x, y, g_1, g_2\}$, i.e., p' is of course not present, and both $A(0)$ and $A(1)$ contain also $getp$ that acts here similarly to $\textsc{noop}(p)$. The corresponding encoding (C) consists of exactly the same clauses as before (plus clauses with $\textsc{noop}(p)$ being mirrored with $getp$), *except that we get the additional clause* $\{\neg getg_1(1), \neg getg_2(1)\}$. This mutex clauses arises because $getg_1$ interferes directly with $getg_2$ (rather than only indirectly through incompatible preconditions), because now $getg_1$ adds p instead of p', and p is deleted by $getg_2$. This yields a trivial two-step (tree-like) resolution proof for $\mathcal{P}^\sigma$, using the two goal clauses and the mutex clause (namely, resolve the second goal clause and the mutex clause deriving $\{\neg getg_1(1)\}$, and then resolve this clause with the first goal clause). On the other hand, in the original task $\mathcal{P}$, $getg_1$ and $getg_2$ are *not* marked mutex in layer $A(1)$, because they don't directly interfere. Therefore, the corresponding mutex clause is not immediately available, and any resolution proof takes more than two steps because it must reason about and involve x.

Encoding (D) works similarly, and σ lets us derive the new mutex clause discussed above. The goal clauses in this case are simply $\{g_1(2)\}$ and $\{g_2(2)\}$. From these, using the two corresponding effect clauses, we can derive the two goal clauses of encoding (C) in two steps. From here, the two-step refutation discussed above derives the empty clause. Thus, we have a four-step resolution refutation for $\mathcal{P}^\sigma$ in encoding (D). There is no similarly small resolution refutation in $\mathcal{P}$ itself, because any such refutation must, as mentioned earlier, reason on x to figure out that $getg_1(1)$ and $getg_2(1)$ cannot both be True. □

Proof of Theorem 4.15. We construct a family of STRIPS tasks whose CNF encodings are very similar to the "pigeon hole problem" formula *PHP(i)*. It is well known that any resolution proof of *PHP(i)* must be of size exponential in i (Haken, 1985). Concretely, *PHP(i)* is an unsatisfiable formula encoding the fact that there is no way to assign $i + 1$

pigeons to i holes such that each pigeon is assigned to at least one hole and no hole gets more than one pigeon. The formula has $i(i+1)$ variables $x_{p,h}$ where $p \in \{1, \ldots, i+1\}, h \in \{1, \ldots, i\}$. For each pigeon p, there is a *pigeon clause* $(x_{p,1}, x_{p,2}, \ldots, x_{p,n})$, and for each pair of pigeons $\{p, q\}$ and hole h, there is a *hole clause* $\{\neg x_{p,h}, \neg x_{q,h}\}$.

The *pigeon hole planning task* $\mathcal{P}_{PHP}(i)$ is defined as follows. For each pigeon p, there is a fact *assigned*(p). For each hole h, there is a fact *free*(h). The initial state contains all i *free* facts but no *assigned* facts. The goal state contains all $i{+}1$ *assigned* facts. The only available actions (other than NOOPs) are $put(p, h)$, which puts pigeon p into a free hole h, after which h no longer remains free. Formally, $put(p, h) = (\{free(h)\}, \{assigned(p)\}, \{free(h)\})$. The plan length bound $b(i)$ is set to 1.

Consider any one of the four encoding methods (A)–(D), and let $\phi(i)$ be the encoding of $\mathcal{P}_{PHP}(i)$. Restrict $\phi(i)$ by setting all NOOP variables to False; this has no real "restrictive" implication in terms of planning since the plan length bound is 1 and none of the goal facts are available at time step 0. For the action-only encodings (A) and (C), identifying the action variables $put(p, h)$ with the *PHP(i)* variables $x_{p,h}$ immediately yields precisely the clauses of *PHP(i)*: the goal clauses of $\phi(i)$ become the pigeon clauses of *PHP(i)* and the action mutex clauses become the hole clauses. For the action-fact encodings (B) and (D), fix all *free* fact variables at time step 0 as well as all *assigned* fact variables at time step 1 to True, and identify $put(p, h)$ action variables as above with $x_{p,h}$. This again yields precisely the clauses of *PHP(i)*. It follows from the resolution hardness of *PHP(i)* and Proposition 2.1 that any resolution proof of the fact that the planning task $\mathcal{P}_{PHP}(i)$ does not have a plan of length 1 must require size exponential in i.

The claim now follows from a planning task $\mathcal{P}'(i)$ which consists of a combination of *two disconnected* pigeon hole planning sub-tasks, $\mathcal{P}_{PHP}(i)$ and $\mathcal{P}_{PHP}(1)$, over two separate sets of pigeon and hole objects. The goal for $\mathcal{P}'(i)$ is naturally defined as follows: put the first set of $i + 1$ pigeons into the first set of i holes *and* put the second set of two pigeons into the second set of holes (which consists of only a single hole). The overall CNF encoding $\phi'(i)$ of $\mathcal{P}'(i)$ is the logical conjunction of the encodings $\phi(i)$ and $\phi(1)$ (on disjoint sets of variables) of $\mathcal{P}_{PHP}(i)$ and $\mathcal{P}_{PHP}(1)$. Observe that $\phi'(i)$ can be proved unsatisfiable by proving unsatisfiability of either of the two pigeon hole problems. In particular, there is a *constant size* resolution refutation of $\phi'(i)$ which involves refuting the $\phi(1)$ component.

On the other hand, we argue that all of the listed abstractions can make the one-hole component of $\mathcal{P}'(i)$ trivially satisfiable, so that a resolution refutation of the abstracted task must resort to a proof of unsatisfiability of the i-hole component $\phi(i)$ of $\mathcal{P}'(i)$, which we have shown requires exponential size. Hence the single example $\mathcal{P}'(i)$ serves to show the claim for all combinations of abstraction method and CNF encoding.

It is easily verified that $\mathcal{P}_{PHP}(1)$ becomes solvable when ignoring the precondition *free*(1) of both $put(1, 1)$ and $put(2, 1)$: we can then put both pigeons into the single hole. The same happens when ignoring the delete effect *free*(1) of both $put(1, 1)$ and $put(2, 1)$.When ignoring the goal *assigned*(2), or when inserting *assigned*(2) into the initial state, or when completely removing *assigned*(2), the one-hole component of $\mathcal{P}'(i)$ requires to assign only one pigeon, which is of course possible. Finally, for variable domain abstraction, note that *assigned*(1) and *assigned*(2) are persistently mutex in $\mathcal{P}_{PHP}(1)$ because the only actions achieving them are $put(1, 1)$ and $put(2, 1)$, respectively. According to Definition 2.2, we can hence replace *assigned*(2) with *assigned*(1). In the resulting planning task, we have the single goal *assigned*(1) which can be achieved in 1 step by, for example, the *put*(1, 1) action. This concludes the argument. □

Proof of Proposition 4.16. We first consider removal of duplicate actions. The same example planning task, denoted $\mathcal{P}'$, works for all four encodings; $\mathcal{P}'$ is defined as follows:

- Fact set $\{r_1, r_2, g_1, g_2, g_3\}$
- Initial state $\{r_1, r_2\}$; Goal $\{g_1, g_2, g_3\}$
- Action set A containing seven actions:
  $1\_1 = (\{r_1\}, \{g_1\}, \{r_1\})$,
  $1\_2 = (\{r_1\}, \{g_2\}, \{r_1\})$,
  $1\_3 = (\{r_1\}, \{g_3\}, \{r_1\})$,
  $2\_1 = (\{r_2\}, \{g_1\}, \{r_2\})$,
  $2\_2 = (\{r_2\}, \{g_2\}, \{r_2\})$,
  $2\_3 = (\{r_2\}, \{g_3\}, \{r_2\})$,
  $help = (\{g_1, g_2\}, \{g_3\}, \emptyset)$.

In this planning task, all actions that are applicable in the initial state "consume" one of the two resources r_1 or r_2. Each of the actions achieves just one of the goals, so each pair of goals can be reached, but not all three of them. The only solution is to perform two steps, in the second of which the *help* action serves to accomplish g_3. We set the plan length bound to 1.

The planning graph $PG(\mathcal{P}')$ up to step 1 has no mutex relations other than the direct mutexes between actions competing for the same resource. Hence, encoding (A) is identical to encoding (C), and encoding (B) is identical to encoding (D). The same properties clearly hold also for the planning task $\mathcal{P}$ that is like $\mathcal{P}'$ except that it has an additional action $1\_1'$ identical to 1_1.

Consider encoding (A) of $\mathcal{P}'$. The goal clauses are $\{1\_1(0), 2\_1(0)\}$, $\{1\_2(0), 2\_2(0)\}$, and $\{1\_3(0), 2\_3(0)\}$. The only other clauses are mutex clauses of the form $\{\neg i\_j, \neg i\_k\}$. It is not difficult to verify that a shortest resolution refutation involves 12 steps. One such derivation proceeds via deriving $\{2\_2(0), 2\_1(0)\}$, $\{2\_3(0), 2\_1(0)\}$, $\{2\_1(0)\}$, $\{1\_2(0)\}$, $\{1\_3(0)\}$, $\{\}$; each of these can be derived, in this sequence, with 2 steps involving resolution against one mutex clause. For $\mathcal{P}$, the only thing that changes is that we now have the clause $\{1\_1(0), 1\_1'(0), 2\_1(0)\}$ instead of $\{1\_1(0), 2\_1(0)\}$, plus the additional mutex clauses. Now, obviously every resolution refutation must resolve on all three goal clauses. To end up with an empty clause, we hence additionally need to get rid of the literal $1\_1'(0)$. Clearly, there is no way to do this other than to resolve that literal away with an additional step involving one of the new mutex clauses. Hence the shortest possible resolution refutation now has ≥ 13 steps.

For encoding (B), the resolution proofs first need to make three steps resolving the goal clauses $\{g_1(1)\}$, $\{g_2(1)\}$, $\{g_3(1)\}$ with the respective effect clauses $\{\neg g_1(1), 1\_1(0), 2\_1(0)\}$, $\{\neg g_2(1), 1\_2(0), 2\_2(0)\}$, and $\{\neg g_3(1), 1\_3(0), 2\_3(0)\}$; thereafter, matters are the same as before.

To show the claim for removal of redundant add effects, we slightly modify the example, and define $\mathcal{P}'$ as follows:

- Fact set $\{r_1, r_2, g_1, g_2, g_3, x\}$
- Initial state $\{r_1, r_2\}$; Goal $\{g_1, g_2, g_3\}$
- Action set A containing eight actions:
  $1\_1 = (\{r_1\}, \{g_1\}, \{r_1\})$,
  $1\_2 = (\{r_1\}, \{g_2\}, \{r_1\})$,
  $1\_3 = (\{r_1\}, \{g_3\}, \{r_1\})$,
  $2\_1 = (\{r_2\}, \{g_1\}, \{r_2\})$,
  $2\_2 = (\{r_2\}, \{g_2\}, \{r_2\})$,
  $2\_3 = (\{r_2\}, \{g_3\}, \{r_2\})$,
  $help_1 = (\{g_1, g_2\}, \{x\}, \emptyset)$,
  $help_2 = (\{x\}, \{g_3\}, \emptyset)$.

In this task, the single *help* action from before is replaced with two help actions that need to be applied consecutively. We set the plan length bound to 2. As before, the planning graph $PG(\mathcal{P}')$ has no mutex relations other than the direct mutexes between actions competing for the same resource; encodings (A)/(C) and (B)/(D) respectively are identical. The same properties clearly hold also for the planning task $\mathcal{P}$ that is like $\mathcal{P}'$ except that $help_1$ has the additional add effect g_1.

Consider encoding (A) of $\mathcal{P}'$. The goal clauses are $\{1\_1(1), 2\_1(1), \textsc{noop}(g_1)(1)\}$, $\{1\_2(1), 2\_2(1), \textsc{noop}(g_2)(1)\}$, and $\{1\_3(1), 2\_3(1), \textsc{noop}(g_3)(1)\}$. Refuting this now involves showing that the three goals cannot all be achieved in step 1, nor in step 0, nor in a combination of the two. Any refutation needs to resolve on all three clauses. As before, for $\mathcal{P}$ we get an additional literal in the first clause, which is now $\{1\_1(1), 2\_1(1), help_1, \textsc{noop}(g_1)(1)\}$. Clearly, getting rid of that additional literal involves at least one more resolution step. For encoding (B), matters are essentially the same except that we first need to resolve the goal fact clauses against the respective effect clauses. □

References

Ball, T., Majumdar, R., Millstein, T., & Rajamani, S. (2001). Automatic predicate abstraction of C programs. In *PLDI'2001: Programming Language Design and Implementation*, pp. 203–213.

Beame, P., Kautz, H., & Sabharwal, A. (2004). Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, *22*, 319–351.

Beck, C., Hansen, E., Nebel, B., & Rintanen, J. (Eds.). (2008). *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08)*. AAAI Press.

Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 279–298.

Blum, A. L., & Furst, M. L. (1995). Fast planning through planning graph analysis. In Mellish, S. (Ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1636–1642, Montreal, Canada. Morgan Kaufmann.

Boddy, M., Fox, M., & Thiébaux, S. (Eds.). (2007). *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*. AAAI Press.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1–2), 5–33.

Bonet, B., & Geffner, H. (2008). Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *Artificial Intelligence*, *172*(12-13), 1579–1604.

Brafman, R. (2001). On reachability, relevance, and resolution in the planning as satisfiability approach. *Journal of Artificial Intelligence Research*, *14*, 1–28.

Chaki, S., Clarke, E., Groce, A., Jha, S., & Veith, H. (2003). Modular verification of software components in C. In *ICSE'2003: Int. Conf. on Software Engineering*, pp. 385–395.

Chen, Y., Huang, R., Xing, Z., & Zhang, W. (2009). Long-distance mutual exclusion for planning. *Artificial Intelligence*, *173*(2), 365–391.

Clarke, E. M., Biere, A., Raimi, R., & Zhu, Y. (2001). Bounded model checking using satisfiability solving. *Formal Methods in System Design*, *19*(1), 7–34.

Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., & Veith, H. (2003). Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery*, *50*(5), 752–794.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, *5*(7), 394–397.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, *7*(3), 201–215.

Edelkamp, S. (2001). Planning with pattern databases. In Cesta, A., & Borrajo, D. (Eds.), *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, pp. 13–24, Toledo, Spain. Springer-Verlag.

Edelkamp, S. (2003). Promela planning. In Ball, T., & Rajamani, S. (Eds.), *Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN-03)*, pp. 197–212, Portland, OR. Springer-Verlag.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S., & Fox, M. (Eds.), *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Lecture Notes in Artificial Intelligence, pp. 135–147, Durham, UK. Springer-Verlag.

Ernst, M., Millstein, T., & Weld, D. (1997). Automatic sat-compilation of planning problems. In Pollack, M. (Ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1169–1176, Nagoya, Japan. Morgan Kaufmann.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3–4), 198–208.

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, *20*, 239–290.

Graf, S., & Saïdi, H. (1997). Construction of abstract state graphs with PVS. In *CAV'1997: Computer Aided Verification*, pp. 72–83.

Gupta, A., & Strichman, O. (2005). Abstraction refinement for bounded model checking. In Etessami, K., & Rajamani, S. (Eds.), *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science, pp. 112–124, Edinburgh, UK. Springer-Verlag.

Haken, A. (1985). The intractability of resolution. *Theoretical Computer Science*, *39*, 297–308.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In Chien, S., Kambhampati, R., & Knoblock, C. (Eds.), *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pp. 140–149, Breckenridge, CO. AAAI Press, Menlo Park.

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-2007)*, pp. 1007–1012. AAAI Press.

Helmert, M., & Mattmüller, R. (2008). Accuracy of admissible heuristic functions in selected planning domains. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 938–943, Chicago, IL. AAAI Press.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning.. In Boddy et al. (Boddy, Fox, & Thiébaux, 2007), pp. 176–183.

Henzinger, T., Jhala, R., Majumdar, R., & McMillan, K. (2004). Abstractions from proofs. In *POPL'2004: Principles of Programming Languages*, pp. 232–244.

Hernadvölgyi, I., & Holte, R. (1999). PSVN: A vector representation for production systems. Tech. rep. 1999-07, University of Ottawa.

Hoffmann, J., & Edelkamp, S. (2005). The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, *24*, 519–579.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Hoffmann, J. (2005). Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.

Hoffmann, J., Gomes, C., & Selman, B. (2007). Structure and problem hardness: Goal asymmetry and dpll proofs in sat-based planning. *Logical Methods in Computer Science*, *3*(1:6).

Hoffmann, J., Sabharwal, A., & Domshlak, C. (2006). Friends or foes? an AI planning perspective on abstraction and search.. In Long, & Smith (Long & Smith, 2006), pp. 294–303.

Katz, M., & Domshlak, C. (2008). Structural pattern heuristics via fork decomposition.. In Beck et al. (Beck, Hansen, Nebel, & Rintanen, 2008), pp. 182–189.

Kautz, H., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In Pollack, M. (Ed.), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 318–325, Stockholm, Sweden. Morgan Kaufmann.

Kautz, H. (2004). SATPLAN04: Planning as satisfiability. In Edelkamp, S., Hoffmann, J., Littman, M., & Younes, H. (Eds.), *Proceedings of the 4th International Planning Competition (IPC-04)*, Whistler, BC, Canada.

Kautz, H., Selman, B., & Hoffmann, J. (2006). SATPLAN: Planning as satisfiability. In Gerevini, A., Dimopoulos, Y., Haslum, P., Saetti, A., Bonet, B., & Givan, B. (Eds.), *Proceedings of the 5th International Planning Competition (IPC-06)*, Ambleside, UK.

Kautz, H. A., McAllester, D., & Selman, B. (1996). Encoding plans in propositional logic. In Aiello, L. C., Doyle, J., & Shapiro, S. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th International Conference (KR-96)*, pp. 374–384, Cambridge, MA. Morgan Kaufmann.

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. In Neumann, B. (Ed.), *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pp. 359–363, Vienna, Austria. Wiley.

Kautz, H. A., & Selman, B. (1996). 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)*, pp. 1194–1201, Portland, OR. MIT Press.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the 8th National Conference of the American Association for Artificial Intelligence (AAAI-90)*, pp. 923–928, Boston, MA. MIT Press.

Koehler, J., Nebel, B., Hoffmann, J., & Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset.. In Steel, & Alami (Steel & Alami, 1997), pp. 273–285.

Koehler, J., & Hoffmann, J. (2000). On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, *12*, 338–386.

Long, D., Kautz, H. A., Selman, B., Bonet, B., Geffner, H., Koehler, J., Brenner, M., Hoffmann, J., Rittinger, F., Anderson, C. R., Weld, D. S., Smith, D. E., & Fox, M. (2000). The aips-98 planning competition. *AI Magazine*, *21*(2), 13–33.

Long, D., & Smith, S. (Eds.)., ICAPS-06 (2006). *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, Ambleside, UK. Morgan Kaufmann.

McDermott, D. (1999). Using regression-match graphs to control search in planning. *Artificial Intelligence*, *109*(1-2), 111–159.

Meuleau, N., Brafman, R., & Benazera, E. (2006). Stochastic over-subscription planning using hierarchies of MDPs.. In Long, & Smith (Long & Smith, 2006), pp. 121–130.

Nebel, B., Dimopoulos, Y., & Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation.. In Steel, & Alami (Steel & Alami, 1997), pp. 338–350.

Prasad, M. R., Biere, A., & Gupta, A. (2005). A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technlogy Transfer*, *7*(2), 156–173.

Ray, K., & Ginsberg, M. L. (2008). The complexity of optimal planning and a more efficient method for finding solutions.. In Beck et al. (Beck et al., 2008), pp. 280–287.

Rintanen, J. (2004). Evaluation strategies for planning as satisfiability. In Saitta, L. (Ed.), *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pp. 682–687, Valencia, Spain. Wiley.

Rintanen, J. (2008). Planning graphs and propositional clause-learning. In Brewka, G., & Doherty, P. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the 11th International Conference (KR-08)*, pp. 535–543, Sydney, Australia. AAAI Press.

Rintanen, J., Heljanko, K., & Niemelä, I. (2006). Planning as satisfiability: parallel plans and algorithms for plan search, artificial intelligence. *Artificial Intelligence*, *170*(12-13), 1031–1080.

Robinson, J. A. (1965). A machine oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, *12*(1), 23–41.

Robinson, N., Gretton, C., Pham, D.-N., & Sattar, A. (2008). A compact and efficient sat encoding for planning.. In Beck et al. (Beck et al., 2008), pp. 296–303.

Sacerdoti, E. (1973). Planning in a hierarchy of abstraction spaces. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 412–422, Stanford, CA. William Kaufmann.

Sanchez, R., & Kambhampati, S. (2005). Planning graph heuristics for selecting objectives in over-subscription planning problems. In Biundo, S., Myers, K., & Rajan, K. (Eds.), *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pp. 192–201, Monterey, CA, USA. Morgan Kaufmann.

Steel, S., & Alami, R. (Eds.). (1997). *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, Vol. 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France. Springer-Verlag.

Streeter, M., & Smith, S. (2007). Using decision procedures efficiently for optimization.. In Boddy et al. (Boddy et al., 2007), pp. 312–319.