Friends or Foes? An AI Planning Perspective on Abstraction and Search

Jörg Hoffmann
Programming Logics Group
Max Planck Institute for CS
66123 Saarbrücken, Germany
hoffmann@mpi-sb.mpg.de

Ashish Sabharwal
Dept. of Computer Science
Cornell University
Ithaca, NY 14853-7501, U.S.A.
sabhar@cs.cornell.edu

Carmel Domshlak
Fac. of Industrial Engr. & Management
Technion - Israel Institute of Technology
Haifa 32000, Israel
dcarmel@ie.technion.ac.il

Abstract

There is increasing awareness that planning and model checking are closely related fields. *Abstraction* means to perform search in an over-approximation of the original problem instance, with a potentially much smaller state space. This is *the* most essential method in model checking. One would expect that it can also be made successful in planning. We show, however, that this is likely to *not* be the case. The main reason is that, while in model checking one traditionally uses blind search to exhaust the state space and prove the absence of solutions, in planning *informed* search is used to *find* solutions. We give an exhaustive theoretical and practical account of the use of abstraction in planning. For all abstraction (over-approximation) methods known in planning, we *prove* that they cannot improve the best-case behavior of informed search. While this is easy to see for heuristic search, we were quite surprised to find that it also holds, in most cases, for the resolution-style proofs of unsolvability underlying SAT-based optimal planners. This result is potentially relevant also for model checking, where SAT-based techniques have recently been combined with abstraction. Exploring the issue in planning practice, we find that even hand-made abstractions do not tend to improve the performance of planners, unless the attacked task contains *huge* amounts of irrelevance. We relate these findings to the kinds of application domains that are typically addressed in model checking.

Introduction

Just like in planning, in model checking one wants to have tools that automatically analyze the behavior of declaratively specified transition systems. Particularly, both in planning and in model checking of "safety properties" – checking reachability of non-temporal formulas – problem instances are given by the description of a transition system, by an initial system state, and by a target condition. A solution is a legal path of transitions mapping the initial state into a state satisfying the target. In model checking, such a solution corresponds to an "error path" in the system, i.e., to an unwanted system behavior. Trying to find such error paths is useful for debugging purposes; proving their absence is the ultimate goal of verification. The traditional focus of the field is on the latter. The key technique to accomplish this ambitious task, besides clever (symbolic) representations of the state space, is *abstraction*: over-approximation of the considered transition system. If the abstracted instance does not contain a solution (an error), then neither does the real system. The key to success is that, in many cases, one can prove the absence of errors in rather coarse abstractions with a relatively small state space. Techniques of this kind have been explored in depth for a long time; just as an entry point, see (Clarke, Grumberg, & Peled 1999).

In planning, traditionally the focus is on *finding* solutions in instances that are assumed to be solvable. Our initial idea in this research was to adapt the concept of abstraction from model checking for use in step optimal planning. There, the main bottleneck is always to *prove the absence* of solutions (plans) of a certain length. Particularly, if the optimal plan has n steps, then the hardest bit is typically to prove that there is no plan of length $n-1$. Our hope was to be able to lead this proof within relatively coarse abstractions. To do so, the abstraction must be *solution length preserving*: not introduce any solutions shorter than the optimal one.[1] To design such abstractions, we developed a new abstraction technique for STRIPS, called *variable domain abstraction*, inspired by work on "pattern databases" (Culberson & Schaeffer 1998), in particular by Hernadvölgyi & Holte (1999): abstract the transition system by not distinguishing between certain values of the (multiple-valued) variables.[2]

Let us illustrate the approach and abstraction method with the Logistics domain. There are packages, which must be transported within cities using trucks, and between cities using airplanes. Actions are load/unload packages, or move vehicles; there are no constraints on (either of) space, fuel, and travel links. 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

[1] In practice, designing solution length preserving abstractions – for finding optimal solutions – often seems harder than designing "solution preserving" abstractions – for proving unsolvability; we get back to this below. Still, as we will see, many planning benchmarks do have solution length preserving abstractions that dramatically decrease the size of the state space.

[2] Edelkamp's (2001) abstraction is coarser than ours; see below.

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: by a factor of more than $((C-2)*S)^P$ where C, S, and P are the number of cities, the city size (number of locations in each city), and the number of packages. We implemented this Logistics-specific abstraction, and fed it into the state-of-the-art step optimal planner SATPLAN'04.[3] We expected to see much improved runtime behavior within the abstraction. To our surprise, *we didn't.* We also tried IPP (Koehler *et al.* 1997), and obtained no discernible improvement.

After some more experimentation, we started suspecting that the approach has some fundamental weakness. It turned out that this is indeed the case. By *informed search*, herein we mean heuristic search and DPLL-style search (the latter is known to be equivalent to restricted forms of resolution proofs, cf. Beame, Kautz, & Sabharwal 2004). Every current state-of-the-art automatic planning system we are aware of is based on a variant of either of these two kinds of informed search. We were able to prove that *abstraction can not improve the best-case behavior of informed search.* This is quite clear for heuristic search (see later). The surprise was to find that, at least in the planning context, it also holds for DPLL-style search, in most cases. Considering three CNF encoding methods and five abstraction methods, we prove for all but two combinations of abstraction and encoding that the shortest resolution refutation of an encoding never becomes shorter in the abstraction; for *all* combinations, it *can* become exponentially longer. We remark that SAT-based methods were also made quite successful in what is called "bounded" model checking (Biere *et al.* 1999). Recently, the combination of these techniques with abstraction methods is being explored (e.g. Gupta & Strichman 2005). Our result trivially holds also for the abstraction techniques currently used in this context (more later). In this sense, our theoretical observations are potentially relevant for model checking as well.

We also analyze abstraction and search in planning from a practical perspective. We investigate the three orthogonal approaches of constraint-based search, heuristic search, and symbolic blind search; we instantiate these with SATPLAN'04, IPP and FF (Hoffmann & Nebel 2001), and Mips.BDD (Edelkamp & Helmert 1999), respectively.[4] Note that all these planners, except FF, are optimal, which we consider to be the more promising case for abstraction, since proving optimality is closely related to proving unsolvability. We conducted experiments in, with few exceptions, all STRIPS domains used in all international planning competitions (IPC) so far. In many cases, we tailored the abstraction to the domain by hand. As one would expect, symbolic blind search is, most of the time, improved by abstraction. For informed search, however, in the IPC benchmark suites we almost never obtain improved behavior.

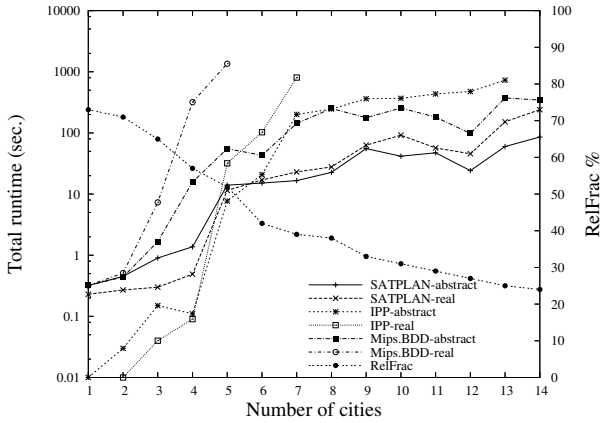

Figure 1: Runtime performance of SATPLAN'04, IPP, and Mips.BDD, with ("abstract") and without ("real") our hand-made variable domain abstraction, in Logistics instances explicitly scaled to increase the amount of irrelevance, as shown by the percentage *RelFrac* of relevant facts.

We constructed some benchmark suites ourselves to explore this issue further. Figure 1 gives a preview, which is instructive at this point. The shown Logistics instances constantly 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. 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 a plot if it had 2 time-outs within the 5 instances of one 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 away).[5] With a RelFrac value above 48% (up to 5 cities), SATPLAN'04 is faster on the real tasks; below RelFrac 42% (more than 6 cities), SATPLAN'04 is faster on the abstract tasks. We will see later (Table 1) that in the IPC 2000 Logistics benchmarks, *RelFrac is 42% on average* – and SATPLAN'04 has a slight but consistent advantage on the *real* tasks. For IPP, the abstract and real curves in Figure 1 change positions a little earlier than for SATPLAN'04, namely when stepping from RelFrac 57% (4 cities) to RelFrac 48% (5 cities). Mips.BDD is consistently much faster on the abstract tasks. *The degree to which SATPLAN'04, IPP, and Mips.BDD are improved by the abstraction clearly corresponds to the degree of quality of their respective search information*, as reflected by their behavior on the non-abstracted tasks. Indeed, the intuition behind our results is that the informedness of the abstraction must compete with the informedness of the search.

The observations in our constructed benchmarks show in particular that our abstraction techniques *can*, in practice,

[3] SATPLAN'04 is the newest version of Blackbox (Kautz & Selman 1999); it won the track for optimal planners at the international planning competition 2004.

[4] The latter does a BDD-based breadth first search, with the main feature that a clever preprocessing is used to obtain a compact state encoding. Graphplan/IPP is sometimes seen as a constraint-based search as well; we take Haslum and Geffner's (2000) view which we believe to be more appropriate.

[5] Note here that the additional cities are irrelevant only for some of *the individual packages* – they can't be removed completely from the task as standard irrelevance detection mechanisms, e.g. (Nebel, Dimopoulos, & Koehler 1997), would try to do.

speed up optimal planning – *provided there is a huge amount of irrelevance*. The problem is that, at least as far as the IPC benchmark suites suggest, the instances that planning people are usually interested in do not contain enough irrelevance. And herein lies another, more subtle, difference to model checking: the kinds of application domains that are typically addressed in model checking often *do* contain huge amounts of irrelevance. The reason is, intuitively, the highly "modular" nature of control structures such as automata networks and programs: often, only fragments of the control structure are relevant for the property to be checked. For example, Clarke et al. (1999) describe "Bit Abstraction", which abstracts logical circuits by considering only the values of some selected bits. If, as a simple example, the task is to prove that the sum of two input numbers, computed by an adding circuit, is odd iff exactly one of the inputs is odd, then it suffices to consider the last bits of the two inputs, and the fraction of the circuit dealing with them. Similarly, Clarke et al. (1999) describe a "Parity" example where it suffices to consider the last bit of a variable to ensure that parity computation works correctly. Note that this is *not* an oddity of the benchmarks used in model checking. Rather, such "modularity" is an essential typical property of human-created control structures – in difference to the more "physical" situations typically considered in planning.

Background

We use the STRIPS formalism, with *fact* set P, *action* set A where $a \in A$ has $pre(a)$, $add(a)$, and $del(a)$, *initial state* I, and *goal* G, all with the standard syntax and semantics. Tuples $\mathscr{P} = (P, A, I, G)$ are called *planning tasks*, or *instances*.

For a length bound b, the *planning graph* $PG(\mathscr{P})$ (Blum & Furst 1997) associated with $\mathscr{P}$ is a layered graph defined as follows. The vertex layers are the sets of nodes $F(0), A(0), F(1), A(1), \ldots, F(b)$, where $F(0)$ is the initial fact set, and $A(t)$ and $F(t+1)$ for $1 \leq t < b$ are the action sets and fact sets, respectively, available at the corresponding time step. The $A(t)$ include the standard *NOOP* actions, $A(t) \subseteq A(t+1)$, $F(t) \subseteq F(t+1)$, and the goal facts G label appropriate vertices in $F(b)$. $PG(\mathscr{P})$ has four kinds of edges: $E_{pre}(t) \subseteq A(t) \times F(t), E_{add}(t) \subseteq A(t) \times F(t+1), E_{a-mutex}(t) \subseteq A(t) \times A(t)$, and $E_{f-mutex}(t) \subseteq F(t) \times F(t)$. The mutex edges $E_{mutex} = E_{a-mutex} \cup E_{f-mutex}$ are computed by an iterative calculation of interfering action and fact pairs.

Propositional Encodings and Resolution

We consider three standard propositional CNF encodings of a planning task $\mathscr{P}$. Our CNF encoding (A) is constructed from $PG(\mathscr{P})$ 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 $\{a_1(b-1), \ldots, a_l(b-1)\}$, where $a_1, \ldots a_l$ are the actions in $A(b-1)$ that add g. Similarly, for every $a(t)$ and every $p \in pre(a)$ we have a *precondition clause* $\{\neg a(t), a_1(t-1), \ldots, a_l(t-1)\}$, where $a_1, \ldots a_l$ 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 $(a, a') \in E_{a-mutex}(t)$. Our CNF encoding (B) is the "Graphplan-based" encoding from Blackbox (Kautz & Selman 1999), which is similar to (A) except that it uses variables (and appropriate clauses) also for the facts. Our CNF encoding (C) is like (A) except that it is based on a "relaxed" planning graph (without mutex propagation), and mutex clauses are present only for action pairs whose preconditions and effects interfere directly. We used to believe that SATPLAN'04 uses encoding (A); it recently came to our attention that, for implementational reasons, actually the much more naive encoding (C) is used.

Our theoretical results are with respect to resolution, which forms the basis of most of the complete SAT solvers around today (cf. Beame, Kautz, & Sabharwal 2004). Our proofs are for (un-restricted) resolution; since our constructions do not affect the structure of the resolution refutations, the results hold as stated for all known variants of resolution, including tree-like (DPLL), regular, and ordered resolution.

For a CNF formula F, $RES(F)$ will denote the *resolution complexity* of F, i.e., the size of the smallest resolution proof of it. Let x be a variable of F and y be True, False, or another (possibly negated) variable of F. The *variable restriction* $x \leftarrow y$ on F is a transformation that symbolically replaces x with y throughout F and simplifies the resulting formula. We state a relatively simple property whose proof may be found in the TR (Hoffmann, Sabharwal, & Domshlak 2006).

Proposition 1. *Let F, G be CNF formulas and τ be a sequence of variable restrictions on G. If every clause of $G|_\tau$ contains as a subclause a clause of F, then $RES(F) \leq RES(G)$.*

Abstraction in Planning

The idea behind abstraction as used in model checking is to examine the state space of an over-approximation of the real task – the *abstract state space* – in order to gain information about the real task: prove the absence of solutions, or at least the absence of solutions of a certain length. By contrast, the over-approximation methods used in planning so far were mostly designed with a focus on computing heuristic functions, where they are called "relaxations". 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 the over-approximation methods that have been used in planning so far; we then formally introduce our new one, variable domain abstraction. We use the Logistics domain as an illustrative example. One widespread over-approximation method in planning is the "2-subset" relaxation underlying the computation made in a planning graph, which is generalized to an "m-subset" relaxation by Haslum & 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 length of a planning graph (its 2-subset solution length) is typically lower than the length of an optimal plan. For $m > 2$, on the other hand, computing m-subset solution length is typically too costly.

A second wide-spread over-approximation method is "ignoring delete lists" (McDermott 1999; Bonet & Geffner 2001). One simply removes (some of) the negative effects of the actions; if all 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 (e.g. Bonet & Geffner 2001; Hoffmann & Nebel 2001). Ignoring deletes is very likely to introduce shorter solutions. E.g., in Towers-of-Hanoi it leads to plans of length n instead of 2^n. In Logistics, if one ignores the deletes of moving actions then 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 relaxation was introduced by Edelkamp (2001) for his "pattern database" heuristic. One removes some facts completely. If enough facts are removed, the task becomes sufficiently simple. Obviously, this 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*, since that precondition is removed. An optimal plan can now just make the package "pop up" anywhere. A fourth relaxation, finally, involves removing some preconditions (Sacerdoti 1973) and/or goal facts. As with Edelkamp's abstraction, this can't be expected to be solution length preserving in interesting cases.

The above calls for a new abstraction method, which we designed following Hernadvölgyi & Holte (1999). Considering STRIPS-like transition systems with multiple-valued (instead of Boolean) variables, they propose to reduce variable domains by not distinguishing between certain values. Our observation is that, in many planning benchmarks, this can be done without introducing shorter plans. In Logistics it is unnecessary to distinguish the positions of packages in irrelevant cities. We can replace the domain of $at(p)$, $\{A_1, A_2, B_1, B_2, C_1, C_2, \ldots\}$, where A and B are the initial and goal cities of p and $A_i, B_i, \ldots$ are locations in cities $A, B, \ldots$, with $\{A_1, A_2, B_1, B_2, C_1\}$. In STRIPS, this amounts to replacing a set of facts $at(p,l)$ with the single fact $at(p,C_1)$.

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 1 (Variable Domain Abstraction). Let $\mathscr{P} = (P,A,I,G)$ be a planning task. Let $p, p' \in P$ be persistently mutex, and $\forall a \in A : (\{p,p'\} \cap del(a)) \subseteq pre(a)$. Then $(\xi(P), \{\xi(a) \mid a \in A\}, \xi(I), \xi(G))$ is called a *variable domain abstraction* of $\mathscr{P}$, where ξ is defined as follows:

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

In words, Definition 1 simply says that we replace p' with p. If, then, p appears both in the add list and in the delete list of an action, we skip it from both (which we can do safely because we know by prerequisite that $p \in \xi(pre)$). The latter will happen, e.g., if the action moves a package from one irrelevant position to another irrelevant position. After the operation, p is equivalent to $p \vee p'$: p is True after an abstracted action sequence iff $p \vee p'$ is True after the corresponding real action sequence. Arbitrarily coarse variable domain abstractions may be generated by iterating the application of Definition 1. Note that this 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. We developed a family of automatic processes to design variable domain abstractions; in several domains, e.g. Logistics, we also hard-coded domain specific abstractions. See the empirical section for details.

Theory

We start with a trivial observation.

Proposition 2. *In solvable instances, abstraction cannot improve the best-case behavior of heuristic search.*

The reason is simply that the best-case for heuristic search is a heuristic function that has oracle qualities and returns the precise goal state distance. As a result, the "search" explores nothing but the states on a shortest path to a goal state. (Obviously, this is not true in unsolvable instances; but there, heuristic search is useless anyway.) An abstraction may introduce invalid solutions. If we assume that the "oracle" heuristic function cannot distinguish between real and invalid solutions, then abstraction can be quite harmful, by leading search through arbitrarily many invalid solutions prior to finding a real one.

Since heuristic functions are not oracles in reality, Proposition 2 does *not* mean that abstraction is necessarily useless for heuristic search. What Proposition 2 says is only that *the informedness of the abstraction has to compete with the informedness of the heuristic function*. The better the heuristic function is, the harder it is for the abstraction to yield an advantage.[6] This is in stark contrast to *blind* search, where, clearly, an exponential reduction of the state space is bound to yield an exponential reduction of the search space (for BDDs, the picture may vary, but one would expect an advantage for smaller state spaces in the typical case).

Matters become a lot more subtle and interesting if one considers resolution-based searches, like DPLL. In what follows, we assume that the plan length bound – the number of time steps in the CNF encoding – is too small, i.e., the CNFs are unsatisfiable. This is the most relevant situation since proving unsatisfiability is the main bottleneck in practice.

Let σ be an (over-)abstraction of a planning task $\mathscr{P} = (P,A,I,G)$. We will denote the abstracted planning task as $\sigma(\mathscr{P})$. A reasonable requirement on σ is the following: if action sequence $\bar{a}$ is applicable in $\mathscr{P}$, then $\sigma(\bar{a})$ is applicable in $\sigma(\mathscr{P})$; if $\bar{a}$ achieves G then $\sigma(\bar{a})$ achieves $\sigma(G)$. For our theoretical analysis herein, we use variants of the following stronger, structural property based on $PG(\mathscr{P})$. Define

[6] This is especially relevant in planning, since some known heuristic functions, e.g. the one used in FF, are pretty close to oracles in relevant classes of planning domains (Hoffmann 2005).

$PG^{\sigma}(\mathscr{P})$ to be the subgraph of $PG(\sigma(\mathscr{P}))$ induced by the vertices of $PG(\mathscr{P})$. (Typically, $PG(\sigma(\mathscr{P}))$ will have many more vertices than $PG(\mathscr{P})$.)

Definition 2 (Planning Graph Abstraction). Let $H = PG(\mathscr{P})$ and $H^* = PG^{\sigma}(\mathscr{P})$ for succinctness. σ *abstracts the planning graph* of $\mathscr{P}$ if: (1) $E_{add}(H^*) \supseteq E_{add}(H)$, (2) $E_{pre}(H^*) \subseteq E_{pre}(H)$, (3) $E_{mutex}(H^*) \subseteq E_{mutex}(H)$, and (4) $\sigma(G) \subseteq G$.

Note that this is a strictly stronger requirement on σ – it may not abstract the planning graph while still obeying the above general conditions for an abstraction.

Can the Resolution Best-Case Get Better?

We first consider encoding (A) only, and get back to (B) and (C) below. We prove the following main result as an immediate consequence of Lemmas 1 to 5.

Theorem 1. *With encoding (A), any combination of the following abstractions cannot improve the best-case behavior of resolution-based search:*

(a) ignoring preconditions, goals, or deletes,
(b) removing a fact completely, and
(c) variable domain abstraction.

For lack of space, we omit the proofs to Lemmas 2, 3, and 4; they are available in the TR.

Lemma 1. *With encoding (A), any abstraction that abstracts the planning graph cannot improve the best-case behavior of resolution-based search.*

Proof. Let $\mathscr{P}$ be a planning task to which an abstraction σ that abstracts $PG(\mathscr{P})$ is applied. Let F, F^* denote the CNF encodings (A) of $\mathscr{P}$ and $\sigma(\mathscr{P})$, respectively. Let U denote the set of variables in F^* that are not variables of F. Finally, let τ be the variable restriction that sets every variable in U to False. $F^*|_\tau$ is the CNF encoding (A) of the planning graph $PG^{\sigma}(\mathscr{P})$. In particular, all clauses of F^* corresponding to actions and facts not in $PG(\mathscr{P})$ are trivially satisfied by τ (they all contain the negation of a variable in U). Call the remaining clauses in $F^*|_\tau$ the *surviving* clauses.

By conditions (2) and (4) of Definition 2, every surviving precondition and goal clause of $F^*|_\tau$ is also present in F itself. Further, by condition (1), each of these clauses contains as a subclause a precondition or goal clause in F. Finally, the surviving mutex clauses in $F^*|_\tau$, by condition (3), are a subset of the mutex clauses in F. It follows that every clause of $F^*|_\tau$ contains as a subclause a clause of F. By Proposition 1, $RES(F) \leq REF(F^*)$, finishing the proof. □

Lemma 2. *Any abstraction that respects the following behavior abstracts the planning graph:*

(a) it does not shrink the list of initial facts,
(b) it does not grow the set of goal facts,
(c) it preserves the add lists unchanged, and
(d) it does not grow the pre and del lists.

Lemmas 1 and 2 together imply that any combination of ignoring goals, ignoring preconditions, and ignoring deletes (as well as adding new initial facts) cannot improve the best-case behavior of resolution-based search.

Lemma 3. *With encoding (A), if a fact p does not appear in the goal or in any of the pre or del lists, then removing p from the initial facts and the add lists cannot improve the best-case behavior of resolution-based search.*

A fact can be removed completely by first removing it from the goal facts and from all *pre* and *del* lists, and then removing it from the initial facts and all *add* lists as well. By Lemmas 2 and 3, neither of these steps can improve the best-case behavior of resolution-based search.

Consider now an *additive variant* of variable domain abstraction where, for an action $a = (pre, add, del)$, $\xi(a) = (\xi(pre), \xi(add), \xi(del) \setminus \{p\})$ when $p \in \xi(add) \cap \xi(del)$. The only change from Definition 1 is that when action $\xi(a)$ is taken in the abstracted task, there is no need to also take the mutually consistent action $NOOP(p)$ to achieve p. Indeed, the following lemma shows that it suffices to consider this additive variant for our arguments.

Lemma 4. *With encoding (A), if $p \in pre(a) \cap add(a)$ and $p \notin del(a)$, then removing p from $add(a)$ cannot improve the best-case behavior of resolution-based search.*

Lemma 5. *With encoding (A), the additive variant of variable domain abstraction cannot improve the best-case behavior of resolution-based search.*

Proof. Let $\mathscr{P}$ be a planning task to which σ, an additive variant of variable domain abstraction, is applied. σ combines two persistently mutex facts p and p' into a single fact p. Let $H = PG(\mathscr{P})$ as before, and define H' to be the graph obtained by unifying any p and p' fact vertices in each layer of H into a single vertex p in that layer. Finally, let H^* denote the subgraph of $PG(\sigma(\mathscr{P}))$ induced by the vertices of H'. We will show that H^* is an abstracted planning graph in a sense very similar to Definition 2.

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

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

Define F, F^*, U, and τ as in the proof of Lemma 1. $F^*|_\tau$ is the CNF encoding (A) of the planning graph H^*, and all clauses of F^* corresponding to actions and facts not in H^* are trivially satisfied by τ. Call the remaining clauses in $F^*|_\tau$ the *surviving* clauses as before.

By our observation about edges $E_{a-mutex}$ in H^* and H, the surviving mutex clauses of $F^*|_\tau$ are also mutex clauses

of F. The surviving precondition and goal clauses not involving p appear unchanged in F. Since we are considering the additive variant of variable domain abstraction, the actions achieving p in H^* are precisely the actions achieving either of p and p' in H. Hence, the surviving precondition and goal clauses of $F^*|_\tau$ involving p contain as a subclause a precondition or goal clause of F itself. It follows from Proposition 1 that $RES(F) \leq RES(F^*)$. □

All the proofs are easy to modify for encodings (B) and (C), except the one of Lemma 5. It is an open question whether Lemma 5 holds for encoding (B); it does *not* hold for encoding (C). The reason is that, in its definition, variable domain abstraction makes use of knowledge about (persistently) mutex facts. This *knowledge about reachability* is not available in the naive encoding (C), and so the "abstraction" can actually, in cases particularly constructed in a way so that this happens, serve to usefully *restrict* the set of satisfying assignments. Generalizing these observations, we get:

Hypothesis 1. Abstraction cannot improve the best-case behavior of resolution-based search, unless it exploits reachability information not exploited in the encoding method.

We refer here to all possible combinations of transition system formalism, abstraction technique, and CNF encoding method. Note that any reachability information exploited during abstraction could just as well be exploited in the encoding itself – in our case, the propagated mutex relations are not used in encoding (C), but are used in (the typically much more efficient) encoding (A).

Hypothesis 1 is not a formal statement. Indeed, at the current stage of our research, it is mere speculation. Formalizing and proving/disproving the statement, in relevant situations, is an exciting topic for future work. We remark that Hypothesis 1 *does* hold for the abstractions currently used in the bounded model checking context (Gupta & Strichman 2005), where abstraction is done by *ignoring some of the clauses encoding the original bounded transition system*. To our surprise, this obvious observation is, to the best of our knowledge, not mentioned in a clear way in the model checking literature.

Can the Resolution Best-Case Get Worse?

All that is to be said in this matter is said in the following theorem, which holds also for encodings (B) and (C):

Theorem 2. *With encoding (A), any of the following abstractions can exponentially deteriorate the best-case behavior of resolution-based search:*

(a) ignoring preconditions, goals, or deletes,

(b) removing a fact completely, and

(c) variable domain abstraction.

Proof Sketch. We construct STRIPS tasks whose encoding (A) is very similar to the "pigeon hole problem" formula $PHP(n)$, resolution proofs for which must be of size exponential in n (Haken 1985). We have facts of the form $assigned(x)$ for $n+1$ pigeons x, and $free(y)$ for n holes y. The actions have the form $put(x,y) = (\{free(y)\},\{assigned(x)\},\{free(y)\})$. The goal is to assign all pigeons, the plan length bound is 1. Restricting, in the encoding, all NOOP variables to 0, one gets exactly $PHP(n)$. The claim now follows with a construction that has *two* disconnected pigeon hole problems, one parameterized by n and the other with constantly only 2 holes. The overall CNF encoding can be proved unsatisfiable by proving unsatisfiability of either of the two pigeon hole problems. If the abstraction makes the mistake to ease the *small* pigeon hole problem, making it solvable, we end up with an exponentially longer best-case resolution proof. For all of the listed abstractions, this can happen. □

The full proof is in the TR. One can construct similar situations in, e.g., Logistics-type domains, with two disconnected parts, one of which is complex while the other one is easy to prove unsolvable in the given number of steps.

Practice

Our most extensive experiments are with variable domain abstraction, which, as discussed in the background section, is clearly the most promising for obtaining solution length preserving abstractions. The first sub-section explains the variable domain abstractions we used, the second one explains the experimental setting and how we chose to present the (huge) data set. We then in turn describe our experiments with variable domain abstraction in the IPC benchmarks, and in artificial situations where the amount of irrelevance can be scaled. In a final sub-section, we discuss the behavior of the other abstraction methods known in planning.

Variable Domain Abstractions

We designed three different methods to automatically generate variable domain abstractions. The methods are based on increasingly accurate approximations of relevance. As is common in relevance approximations (e.g. Nebel, Dimopoulos, & Koehler 1997), the algorithmic basis is, in all cases, a simplified backchaining from the goals. Our first approximation, *1Support*, starts in the first layer of a planning graph that contains the goals (with mutexes, maybe); for each goal, it selects one achiever in the preceding action layer and marks the preconditions as new sub-goals; then the process is iterated for the created sub-goals. Our second approximation, *AllSuppports*, does the same except that it selects *all* achievers for each (sub-)goal. Our third approximation, *AllSupportsNonMutex*, is like AllSupports except that it starts the backchaining at the first plan graph layer that contains the goals *without* mutexes. In all cases, the set of "relevant" facts is turned into a variable domain abstraction by, first, computing a partition of the fact set into subsets of pairwise persistently mutex facts. We then take these subsets to correspond to the underlying multiple-valued variables (e.g. the position of a package). Within each subset, we arbitrarily choose one irrelevant fact and replace all other irrelevant facts with it. As an example, in Logistics, 1Support will abstract away all $in(p,v)$ facts for a package p except for those vehicles v that were selected as a support – in particular, a single airplane. AllSupports, by contrast, 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). AllSupportsNonMutex, finally, 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.

For six IPC domains, we designed solution length preserving, domain specific, variable domain abstractions by hand. For Logistics, this 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.[7] 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 into. 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 the initial position, relevant for a needed rock sample/soil sample/image, or necessarily lies on a path the rover must traverse to some other relevant location.

Experiment Setup and Presentation

The presented data are generated on a set of work stations running Linux, each with a Pentium 4 processor running at 3GHz with 1G RAM. We used a time cutoff of 30 min. We experimented with the optimal planners SATPLAN'04,[8] IPP, and Mips-BDD; to exemplify the effect of abstraction on *sub*-optimal planning, we also ran FF. As test examples, we took, with few exceptions, all STRIPS domains used in all international planning competitions so far. Precisely, we ran Grid, Mprime, and Mystery from IPC 1998 (Gripper and Movie are trivial, Logistics is part of our IPC 2000 set); Blocksworld and Logistics from IPC 2000 (Miconic-STRIPS is just a very simple version of Logistics, Freecell is part of our IPC 2002 set); Depots, Driverlog, Freecell, Rovers, Satellite, and Zenotravel from IPC 2002; Airport, Dining Philosophers, Optical Telegraph, Pipesworld NoTankage, Pipesworld Tankage, and PSR from IPC 2004.

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^* of T, and run a planner X on it. There are three possible outcomes:

1. X finds a plan for T^*, an *abstract plan*, and that is also a real plan (a plan for T). We record the time taken to find the plan, and the time taken by X to find a plan given T.
2. X finds a plan for T^* that is *not* a real plan. If X is an optimal planner, the information gained is the length of the abstract plan: a lower bound on the real plan length. We record the time taken to compute that bound (e.g., for SATPLAN'04, the time taken up to the last unsatisfiable iteration), and the time taken by X to compute the same lower bound given T. If X is not optimal, *no* information is gained, and we skip the task.
3. X runs out of time or memory. In this case, if X is optimal, 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.

Note that we do *not* include the time taken to create the abstract task T^*. For presentation, for each individual instance we select the "best" performing abstraction, of the possible three or four, as follows. 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. Else, 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.[9] If there are several abstractions providing the same best information, we choose the one with lowest (abstract) runtime.

We show data tables for the optimal planners only. In addition to the explained runtimes, the tables specify whether the found abstract plan was real or not. They specify the lower bound proved by a planning graph, the lower bound proved by X in the abstract task, and the optimal plan length. They finally specify *RelFrac*, the percentage of facts considered relevant. We consider SATPLAN'04 in more detail than the other optimal planners, since it is the most relevant solver: it is state of the art in step-optimal planning; in particular it is known to be much faster in most IPC benchmarks than either Graphplan-based or BDD-based approaches.

IPC Benchmarks

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 an interesting case since there, unusually, the current optimal planners do just as well (or bad) 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.

Consider the data for Depots in Table 1. The best-case data shown is half due to the hand-made abstraction; the other instances are distributed more or less evenly over AllSupports and AllSupportsNonMutex, which are, sometimes successfully, more aggressive. Instance nr. 8 has its best-case with the very aggressive 1Support strategy. Most of the time the runtime is better on the original task. But there are a few cases where the abstraction brings a quite significant

[7] 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.

[8] Through using the naive encoding (C), the resolution best-case of SATPLAN'04 *can* be improved by variable domain abstraction – making the "bad" empirical results below even more significant.

[9] 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	1	2	3	4	7	8	10	13	14	16	17	19	21
t_a	0.2	0.5	73.3	429.7	10.44	82.5	47.8	13.8	248.9	24.9	22.7	460.7	342.2
t_r	0.2	0.6	45.8	472.0	12.25	3.6	33.1	20.7	194.5	8.6	22.0	–	55.7
IsReal?	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	N
l_g,l_a,l_r	5,5,5	7,8,8	11,12,12	12,14,14	7,10,10	9,12,14	8,10,10	9,9,9	9,10,10	8,8,8	6,7,7	8,10,10	7,7,7
RelFrac	75%	88%	88%	88%	77%	27%	87%	85%	73%	81%	58%	92%	51%
Logistics													
Index	11	12	13	14	15	16	17	18	19	20	21	22	23
t_a	1.0	0.5	207.5	53.6	146.3	253.2	296.3	728.3	684.2	820.6	615.0	781.8	965.5
t_r	1.5	1.0	171.5	70.9	160.4	168.8	246.4	639.5	672.2	721.7	430.9	720.9	769.4
IsReal?	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
l_g,l_a,l_r	10,13,13	10,12,12	9,13,13	9,12,12	9,13,13	9,13,13	9,13,13	9,15,15	9,15,15	9,15,15	9,14,14	9,15,15	9,15,15
RelFrac	49%	43%	48%	49%	45%	37%	36%	47%	30%	34%	48%	39%	43%
PSR													
Index	22	24	27	29	31	33	35	37	38	40	42	44	48
t_a	64.3	0.2	0.6	11.1	1.5	1.4	0.6	2.4	0.3	1.2	1.1	0.7	125.5
t_r	71.9	0.2	0.6	12.5	10.3	2.1	0.8	2.3	0.3	7.5	1.3	0.7	131.9
IsReal?	Y	Y	Y	Y	N	N	N	Y	Y	N	N	Y	Y
l_g,l_a,l_r	7,25,25	3,9,9	4,16,16	7,18,18	5,16,16	5,16,16	5,13,16	7,19,19	6,12,12	5,14,15	5,16,16	7,15,15	7,26,26
RelFrac	75%	38%	51%	79%	49%	48%	47%	60%	56%	48%	53%	93%	80%
Satellite													
Index	5	6	7	8	9	10	11	12	13	14	15	17	18
t_a	53.1	22.3	16.4	310.8	34.4	168.0	84.5	874.8	931.2	256.2	282.8	65.8	112.5
t_r	51.5	28.2	27.7	250.1	40.7	176.5	160.0	–	–	425.4	429.8	152.4	217.8
IsReal?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
l_g,l_a,l_r	4,7,7	6,8,8	4,6,6	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	95%	81%	88%	90%	85%	86%	75%	76%	76%	79%	75%	67%	74%

Table 1: Full results, in some selected domains, for SATPLAN'04 and variable domain abstraction (best-of, see text). Notations: "Index": index (nr.) of instance in respective IPC suite; t_a runtime (secs) needed in abstracted task; t_r runtime (secs) needed in real task; "IsReal": is the abstract plan real (Y) or not (N); l_g lower bound on plan length proved by planning graph, l_a lower bound proved in abstract task, l_r real optimal plan length; "RelFrac": fraction of facts considered relevant; dashes: time-out.

advantage. Most notably, in instance nr. 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, all best-case data is due to the hand-made abstraction. The abstract runtime is worse in all but two cases (nrs. 14 and 15). In PSR, the best-cases are half due to AllSupports, and half due to AllSupportsNonMutex. The runtimes are mostly inconclusive, with a slight advantage for abstraction. Satellite is the only one of our 17 domains where abstraction brings a consistent runtime advantage. The best-cases are due to our hand-made abstraction. All abstract plans are real plans, found much faster than for the original task in almost all cases. It is unclear to us why the results are reasonably good in Satellite, but, e.g., not in Logistics, where the state space reduction is much larger.

We next provide an overview of the results in all 17 domains, including also IPP and Mips-BDD. To make data presentation feasible, we select just one instance per domain and planner: the "most challenging successful" instance. By successful, we mean that at least one abstract task was solved (abstract plan found), and indeed abstract (not all facts relevant). By challenging, we mean maximum runtime on the original task.[10] The data are in Table 2.

We discuss the 17 domains in groups with similar behavior; we first discuss SATPLAN'04 in some detail, and then briefly summarize the behavior of IPP and Mips.BDD. Depots, Logistics, PSR, and Satellite have already been discussed. In each of Airport, Dining Philosophers, Driverlog, Mystery, Mprime, Optical Telegraph, Pipesworld NoTankage, Pipesworld Tankage, and Zenotravel, SATPLAN'04 runtimes are consistently lower on the original tasks, with few exceptions mostly among the easiest instances. The degree of the advantage varies. It is relatively moderate in Airport (up to 28% less runtime on original task), Dining Philosophers (up to 7%), Mprime (up to 36%), Pipesworld Tankage (up to 28%), and Optical Telegraph (up to 23%); it is stronger in Driverlog (up to 89%), Mystery (up to 80%), Pipesworld NoTankage (up to 92%), and Zenotravel (up to 75%). 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 don't abstract anything; in the larger instances, and in all abstractions generated with 1Support, SATPLAN'04 runs out of time, leaving instance nr. 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 instance nr. 1, which is shown in Table 2.

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: 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 and Satellite; the latter 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 2, Mips.BDD runs out of time on even the smallest instances. In the domains left empty, we couldn't get Mips.BDD to run, i.e., it stopped ab-

[10] 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.

	SATPLAN'04						IPP						Mips-BDD					
Domain	ID	t_a	t_r	IR	l_g, l_a, l_r	RF	ID	t_a	t_r	IR	l_g, l_a, l_r	RF	ID	t_a	t_r	IR	l_g, l_a, l_r	RF
Air	20	73.7	53.5	Y	25,32,32	70%	8	67.9	0.3	Y	25,26,26	77%	–	–	–	–	–	–
Blocks	7	7.6	5.1	N	16,20,20	77%	7	0.0	0.0	Y	16,20,20	77%	–	–	–	–	–	–
Depots	19	460.7	–	Y	8,10,10	92%	17	254.4	268.4	Y	6,7,7	58%	2	1.1	10.6	Y	7,15,15	81%
Dining	29	8.0	7.5	Y	7,11,11	71%	5	170.4	138.0	Y	7,11,11	71%						
Driver	13	342.0	113.7	N	9,11,12	61%	9	1.1	0.7	N	7,10,10	84%	9	375.1	530.9	N	7,19,20	84%
Free	1	0.7	0.7	Y	4,5,5	83%	1	0.2	0.1	Y	4,5,5	83%	–	–	–	–	–	–
Grid	1	2.1	1.0	N	14,7,14	43%	1	0.1	0.2	N	14,7,14	43%	1	1.8	4.9	N	14,7,14	43%
Log	23	965.5	769.4	N	9,15,15	43%	12	394.9	416.6	N	10,11,11	52%	12	7.2	–	Y	10,42,42	43%
Mprime	5	7.5	6.4	Y	6,6,6	78%	2	0.6	1.0	N	5,5,5	62%	–	–	–	–	–	–
Mys	20	180.7	112.4	Y	7,7,7	77%	2	0.4	0.7	N	5,5,5	60%	–	–	–	–	–	–
Optic	13	58.3	45.1	N	11,13,13	53%	2	15.7	5.2	N	11,13,13	53%						
PipeNT	12	521.7	455.5	Y	8,14,14	86%	8	0.2	0.1	N	5,5,5	74%						
PipeT	7	93.8	67.3	Y	4,6,6	89%	10	278.9	268.1	N	6,7,7	98%						
PSR	48	125.5	131.9	Y	7,26,26	80%	10	0.0	0.0	N	4,4,5	37%	25	0.7	13.5	Y	4,9,9	38%
Rovers	8	83.4	84.1	N	5,9,9	75%	6	592.5	375.7	N	7,12,12	90%	7	142.6	340.6	Y	5,18,18	86%
Sat	12	874.8	–	Y	6,14,14	76%	7	100.3	1705.7	Y	4,6,6	88%						
Zeno	13	338.4	244.8	N	4,7,7	67%	12	344.3	322.4	Y	4,6,6	67%	11	271.0	–	Y	4,14,14	67%

Table 2: Results for (best-case) variable domain abstraction on the respectively most challenging successful instance (see text), of each domain, for each of SATPLAN'04, IPP, and Mips-BDD. Notations: "ID" Index, "IR" IsReal?, "RF" RelFrac; rest as in Table 1. Domain names abbreviated as obvious.

normally with a variety of error messages. In the remaining data set of 7 domains, however, our abstractions bring a consistent advantage for Mips.BDD. In particular, consider the behavior in Logistics, Rovers, and Zenotravel: Mips.BDD is vastly improved while SATPLAN'04 and IPP are more or less inconclusive.

We also ran FF on all the domains, and with all the abstractions. In PSR, none of the abstract plans works in reality, meaning that *no* information about the tasks is gained. In Airport and Dining Philosophers, FF is slowed down considerably, up to a factor of 100, by the abstractions. In Blocksworld, it is considerably improved, solving some abstract tasks that FF can't solve in their real representation. In all other domains, the results are completely inconclusive.

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 is rather hopeless. We ran a number of experiments to examine the more subtle reasons for the phenomenon. We ran experiments 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. We conclude from this that the Rovers domain is not amenable to abstraction techniques. For Zenotravel, we obtained the picture shown in Figure 2.

The shown Zenotravel instances constantly feature 2 airplanes and 5 persons. The number of cities scales from 2 to 13. Like in Logistics, we generated 5 random instances per size, and show average values with a time-out of 1800

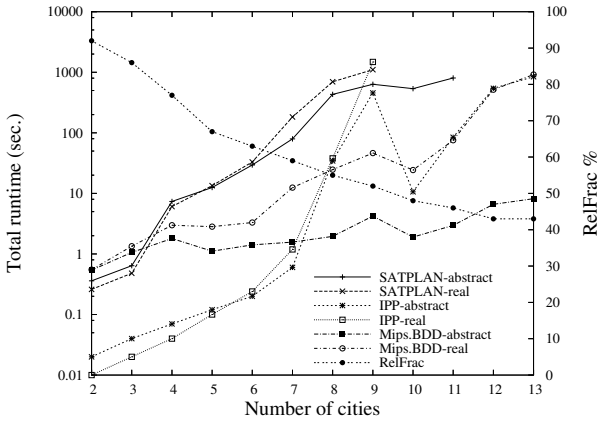

Figure 2: Runtime performance of SATPLAN'04, IPP, and Mips.BDD, with ("abstract") and without ("real") our hand-made variable domain abstraction, in Zenotravel instances explicitly scaled to increase the amount of irrelevance, as shown by the plot of RelFrac.

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: for SATPLAN'04 and IPP, abstraction has a slight disadvantage with high RelFrac, but becomes efficient as RelFrac decreases; for Mips.BDD, decreasing RelFrac consistently widens the gap between abstract and real. 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. Interestingly, in this domain, at least in this distribution of it, Mips.BDD and IPP are much more efficient than SATPLAN'04.

Some planning benchmarks (like Rovers) don't have good abstractions, and most others (like Logistics and Zenotravel) don't have enough irrelevance in the IPC test suites.

As discussed earlier, the situation is typically quite different in model checking benchmarks: unsolvable examples with a highly modular structure. The IPC domains Dining Philosophers and Optical Telegraph, which come from model checking (Edelkamp 2003), are exceptional: Dining Philosophers is an extremely basic benchmark that can't be abstracted much further. Optical Telegraph is essentially a version of Dining Philosophers with a complex "inner life" (exchanging data between the two "hands" of each philosopher). This is a modular structure in the sense that the inner life does not affect the *existence* of a solution (deadlock situation), which depends exclusively on the outer interfaces of the "philosophers" – 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. This highlights an important aspect of the difference between solvable and unsolvable tasks: it seems easier to abstract the latter without invalidating the property of interest. One might thus suspect that abstraction could be made successful for *unsolvable* examples also in planning. Exploring this is a topic for future work. Note that most planning (benchmark) domains don't naturally contain any unsolvable instances; the issue may become relevant in over-subscription planning, however.

Other Abstractions

As discussed earlier, one cannot expect that removing preconditions, goals, or entire facts preserves plan length in interesting cases. There *are* certain cases where some delete effects can safely be ignored: in Driverlog, Logistics, Mprime, Mystery, and Zenotravel, one can ignore the deletes of "load" and "unload" actions; in Rovers one can ignore the deletes of actions taking rock or soil samples. We ran each of our planners on the respective abstracted tasks. For SATPLAN'04, the results are inconclusive except in Logistics, where there is a slight but consistent gain in the abstraction, and Driverlog, with a clear loss in the abstraction (e.g. task nr. 15 is solved abstract vs. real in 693.0 vs. 352.3 sec). IPP behaves inconclusively except a vast gain in Logistics (e.g. 52.8 vs 5540.1 sec in nr. 12), and a vast loss in Zenotravel (e.g. 318.5 vs 2.5 sec in nr. 12). FF behaves inconclusively in all domains except a vast loss in Driverlog (e.g. time-out vs. 0.22 sec in nr. 14). Mips.BDD has a vast loss in Driverlog, Logistics, and Zenotravel (e.g. 163.8 vs. 8.3 sec in Zenotravel nr. 8), and behaves inconclusively in the other domains.

Conclusion

There are at least two important differences between model checking and planning: the typical kinds of application domains addressed, and a focus on unsolvable vs. solvable examples. Our empirical results strongly suggest that these differences make the use of abstraction, which is extremely successful in model checking, rather hopeless in (classical) planning. Moreover, even from a theoretical perspective, the use of abstraction with informed search does not seem very promising: apparently, the only hope to improve the best-case behavior is to exploit reachability knowledge in the abstraction – knowledge that could just as well be exploited directly in the CNF encoding. Whether this hypothesis is true in general or not is an exciting topic for future work.

References

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

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Proc. TACAS'99*, 193–207.

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

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

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. MIT Press.

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proc. ECP'99*, 135–147.

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

Edelkamp, S. 2003. Promela planning. In *Proc. SPIN'03*, 197–212.

Gupta, A., and Strichman, O. 2005. Abstraction refinement for bounded model checking. In *Proc. CAV'05*, 112–124.

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

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. AIPS'00*, 140–149.

Hernadvölgyi, I., and Holte, R. 1999. PSVN: A vector representation for production systems. Technical Report 1999-07, University of Ottawa.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Sabharwal, A.; and Domshlak, C. 2006. Friends or foes? An AI planning perspective on abstraction and search. Technical Report. Available at http://www.mpi-sb.mpg.de/∼hoffmann/tr-icaps06a.ps.gz.

Hoffmann, J. 2005. Where ignoring delete lists works: Local search topology in planning benchmarks. *JAIR*. To appear.

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

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. ECP'97*, 273–285.

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

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. ECP'97*, 338–350.

Sacerdoti, E. 1973. Planning in a hierarchy of abstraction spaces. In *Proc. IJCAI'73*.