A Planning Heuristic Based on Causal Graph Analysis

Malte Helmert
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 052, 79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

In recent years, heuristic search methods for classical planning have achieved remarkable results. Their most successful representative, the FF algorithm, performs well over a wide spectrum of planning domains and still sets the state of the art for STRIPS planning. However, there are some planning domains in which algorithms like FF and HSP perform poorly because their relaxation method of ignoring the "delete lists" of STRIPS operators loses too much vital information.

Planning domains which have many dead ends in the search space are especially problematic in this regard. In some domains, dead ends are readily found by the human observer yet remain undetected by all propositional planning systems we are aware of. We believe that this is partly because the STRIPS representation obscures the important *causal structure* of the domain, which is evident to humans.

In this paper, we propose translating STRIPS problems to a planning formalism with multi-valued state variables in order to expose this underlying causal structure. Moreover, we show how this structure can be exploited by an algorithm for detecting dead ends in the search space and by a planning heuristic based on hierarchical problem decomposition.

Our experiments show excellent overall performance on the benchmarks from the international planning competitions.

Introduction

Many current algorithms for STRIPS-style planning are based on heuristic forward search in the space of world states. The first successful domain-independent planning system using this approach was Bonet and Geffner's HSP (Bonet, Loerincs, & Geffner 1997; Bonet & Geffner 2001). Other researchers adopted the general concept and applied various modifications to it, leading to the development of planners like GRT (Refanidis & Vlahavas 1999), MIPS (Edelkamp & Helmert 2000), which uses heuristic forward search among other planning strategies, and most prominently Hoffmann's very successful FF system (Hoffmann & Nebel 2001). All these planners can be seen as using some approximation of the h^+ heuristic (Hoffmann 2002). The h^+ value of a world state is the length of a shortest plan for a *relaxed* planning task, where all operator effects that make state variables false are ignored.

As an illustration, consider the transportation task depicted in Fig. 1. The nodes in the figure correspond to locations, and solid arcs correspond to roads. Note that location D has four incoming but no outgoing roads, and hence can be entered but not left. The objective is to move a cargo item located at E to location B, as indicated by the dashed arrow. Two trucks initially located at C and F may be used to achieve this goal (gray nodes). This task is similar in structure to several MYSTERY tasks from the planning competition benchmark suite, e. g. MYSTERY #22 and #23.[1]

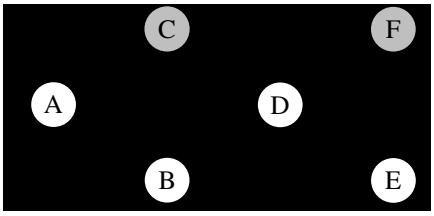

Figure 1: An unsolvable transportation task.

We see that this task has no solution. Even if the cargo item could traverse the roadmap by itself, without help of the trucks, there would be no path from its initial location to its destination. However, using a standard STRIPS encoding of the planning task, the h^+ value of the depicted state is 8, not ∞. This is because the relaxed task allows vehicles to be at several locations simultaneously: When they are moved, the effect of no longer being at the original location is ignored. This can be interpreted as an ability of the truck to "teleport" to any previously visited location without cost and regardless of the existence of a path to that location.

We thus obtain the following optimal relaxed plan: Move the right truck to the cargo item and pick it up, then move to D and drop it. Then move the left truck to D, pick up the cargo item, "teleport" back to C (not counted as a plan step) and finally move to B and drop the cargo.

Unrealized dead ends like this are not the only source of problems for heuristic forward planners. Consider the example in Fig. 2. Again, the objective is to transport the cargo

[1] In MYSTERY tasks, the roadmap is undirected in principle, but lack of fuel at a node can lead to situations like the one above. This is the case in the tasks mentioned, where locations "cherry" and "hotdog" start with empty fuel depots and lie on all paths from the initial location to the goal location of some cargo item.

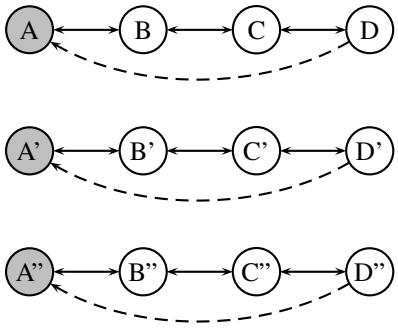

Figure 2: A simple transportation task.

items as indicated by the dashed arrows, using the trucks indicated by the gray vertices. The task is solved by first moving the trucks to the initial cargo locations to pick up everything, then moving them back to drop everything.

However, the task is troubling for h^+-based planners. The h^+ heuristic does not favor moving a truck towards a cargo item over moving it in the opposite direction, as the heuristic anticipates the eventual need to move the truck back to its origin. More precisely, all states in which no cargo items have been picked up form a single plateau with the same heuristic evaluation. Indeed, if we scale the number of trucks and cargo items and the diameter of the graph to higher values, current heuristic forward planners fail quickly.

We should emphasize that we do not want to imply that the h^+ heuristic is inherently bad; in fact it yields exceedingly good estimates in many planning domains. The point we want to make is that the two examples are not at all trivial to analyze at the propositional level of `at-truck-A` and `in-package-truck`, on which h^+ depends. In our opinion, understanding the tasks is much easier if we realize that they are really about *multi-valued state variables* like "the position of the truck" (with values like A, B, C) or "the state of the cargo item" (with values like *at-A* or *in-truck*), and that these state variables only interact in specific, very limited ways.

This paper discusses how such higher-level concepts and their interactions can be found and used automatically by a domain-independent planning system. In the sections to come we will formalize these notions, analyze an important special case of the resulting planning problem, and show how solutions to a number of simplified tasks can serve as a heuristic for the general case.

The SAS$^+$ Planning Formalism

The SAS$^+$ planning formalism is an alternative to propositional STRIPS that has been analyzed in depth by several researchers. Unlike STRIPS, it allows state variables to have non-binary (finite) domains. For example, the position of a truck in our first example can be encoded as a single state variable with six possible values.

Definition 1 ***SAS$^+$ planning task***
*A **SAS$^+$ planning task** or **SAS$^+$ task** for short is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ with the following components:*

- *$\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of **state variables**, each with an associated finite domain $\mathcal{D}_v$.*
  *A **partial variable assignment** over $\mathcal{V}$ is a function s on some subset of $\mathcal{V}$ such that $s(v) \in \mathcal{D}_v$ wherever $s(v)$ is defined. If $s(v)$ is defined for all $v \in \mathcal{V}$, s is called a **state**.*
- *$\mathcal{O}$ is a set of **operators**, where an operator is a pair $\langle pre, eff \rangle$ of partial variable assignments called **preconditions** and **effects**, respectively.*
- *s_0 is a state called the **initial state**, and $s_\star$ is a partial variable assignment called the **goal**.*

An operator $\langle pre, eff \rangle$ is applicable in a state s iff $s(v) = pre(v)$ whenever $pre(v)$ is defined. Applying the operator changes the value of v to $eff(v)$ if $eff(v)$ is defined. For a precise definition of the semantics of SAS$^+$ planning, we refer to the paper by Bäckström and Nebel (1995). Our definition differs slightly from the reference in order to more closely resemble STRIPS. Like propositional STRIPS planning, SAS$^+$ planning is **PSPACE**-complete.

Translating STRIPS to SAS$^+$

Planning tasks specified in propositional STRIPS can be straightforwardly translated into SAS$^+$ planning tasks with an identical set of (binary) state variables. However, we can do better than that using non-binary variables.

A typical STRIPS encoding of the example in Fig. 1 uses 18 propositional variables $at_{o,l}$ to describe the location of the trucks and cargo on the map, plus two state variable in_{t_1} and in_{t_2} to model cargo inside either truck.

An alternative SAS$^+$ encoding uses only three state variables. Two variables v_1 and v_2 specify the truck locations; they can assume six different values corresponding to the six locations. The third variable v_c defines the state of the cargo item and may assume eight different values corresponding to being at any of the six locations or inside either truck. This encoding is more concise in the sense that it only allows for $6^2 \cdot 8 = 288$ states, whereas the STRIPS encoding allows for $2^{20} = 1048576$ states including ones where trucks are at several locations simultaneously or nowhere at all. Of course, the number of *reachable* states is the same for both encodings, and there is a straightforward bijection between reachable states that can be used for translating operators, shown in Fig. 3.

STRIPS state S	SAS$^+$ state s
$at_{o,l} \in S$	$s(v_o) = l$
$in_t \in S$	$s(v_\mathrm{c}) = t$

Figure 3: STRIPS encoding vs. SAS$^+$ encoding.

While it seems like human insight is required to come up with this translation, several existing algorithms are capable of doing the kind of mutual exclusion reasoning among state variables that leads to the more concise encoding. Examples include TIM (Fox & Long 1998) and the inference methods of DISCOPLAN (Gerevini & Schubert 1998). In this paper, we have used the preprocessing method applied by the MIPS planner, described in detail in the paper by Edelkamp and

Helmert (1999). Fig. 4 shows the translation result for the example.

$$
\begin{aligned}
\mathcal{V} &= \{v_1, v_2, v_c\} \\
\mathcal{D}_{v_1} &= \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\} \\
\mathcal{D}_{v_2} &= \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\} \\
\mathcal{D}_{v_c} &= \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{t}_1, \mathrm{t}_2\} \\
\mathcal{O} &= \{\langle\{v_1 \mapsto \mathrm{A}\}, \{v_1 \mapsto \mathrm{B}\}\rangle, \\
&\quad \langle\{v_1 \mapsto \mathrm{B}, v_c \mapsto \mathrm{B}\}, \{v_c \mapsto \mathrm{t}_1\}\rangle, \\
&\quad \langle\{v_2 \mapsto \mathrm{E}, v_c \mapsto \mathrm{t}_2\}, \{v_c \mapsto \mathrm{E}\}\rangle, \ldots\} \\
s_0 &= \{v_1 \mapsto \mathrm{C}, v_2 \mapsto \mathrm{F}, v_c \mapsto \mathrm{E}\} \\
s_\star &= \{v_c \mapsto \mathrm{B}\}
\end{aligned}
$$

Figure 4: SAS$^+$ encoding of the running example (Fig. 1). Three operators are shown, one instance each of a movement, pick-up and drop action.

Subtasks and Causal Graphs

The basic idea of our approach is to decompose a planning task into subtasks with limited interaction. Let us first formalize our notion of subtask.[2]

Definition 2 ***SAS$^+$ subtask***
Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a SAS$^+$ task and $\mathcal{V}' \subseteq \mathcal{V}$ be a subset of its state variables. The ***subtask induced by*** *$\mathcal{V}'$ is the 4-tuple $\Pi' = \langle \mathcal{V}', \mathcal{O}|\mathcal{V}', s_0|\mathcal{V}', s_\star|\mathcal{V}' \rangle$, where we define $\mathcal{O}|\mathcal{V}' = \{\ \langle pre|\mathcal{V}', eff|\mathcal{V}' \rangle \mid \langle pre, eff \rangle \in \mathcal{O} \wedge eff|\mathcal{V}' \neq \emptyset\ \}$.*

Of course, subtasks of a SAS$^+$ task are also SAS$^+$ tasks. It is easy to see that if Π has a solution, then so has Π', although the converse is often not the case. This implies that if Π' does *not* have a solution, then Π is also unsolvable. This property leads to a method for detecting dead ends in the search space of a planning task.

However, this method is fairly limited: In Fig. 1, *all* subtasks except for the original task itself are solvable. For instance, consider the subtask induced by $\{v_1, v_c\}$. It can be solved in two steps by first moving the cargo item into the second truck and then from there to the goal location B. Note the fact that the absence of the state variable for the second truck from the subtask does *not* lead to a situation where the second truck is not available. Instead, it leads to a situation where that truck is *everywhere at the same time*, since all preconditions involving v_2 are projected away.

Clearly, simplifying the planning task in such a way that important preconditions disappear should be avoided. To formalize this notion, we introduce the concept of *causal graphs*. The idea is by no means new; early work discussing it (although under a different name) includes the papers on hierarchical problem solving by Knoblock (1994) and Bacchus and Yang (1994). More recently causal graphs have been applied to planning in unary STRIPS domains (Domshlak & Brafman 2002)[3].

Definition 3 ***causal graph***
Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a SAS$^+$ task. Its ***causal graph*** *$\mathrm{CG}(\Pi)$ is the digraph $(\mathcal{V}, A)$ containing an arc (u, v) iff $u \neq v$ and there exists an operator $\langle pre, eff \rangle \in \mathcal{O}$ such that $eff(v)$ is defined and either $pre(u)$ or $eff(u)$ are defined.*

Informally, the causal graph contains an arc from a source variable to a target variable if changes in the value of the target variable can depend on the value of the source variable. Note that we have decided to include an arc in the causal graph even if this dependency is only of the form of an *effect* on the source variable.

This is a conscious decision. With our definition, a state s can be transformed into a state s' if and only if this is possible in the task induced by the causal graph ancestors of all variables on which s and s' differ. This important property would not hold if arcs between variables affected by the same operator were not part of the causal graph.

This separability property very much facilitates solving tasks where variables have few ancestors in the causal graph, which is one of the reasons why many papers focus on planning tasks with *acyclic* causal graphs (Domshlak & Brafman 2002; Williams & Nayak 1997). An acyclic causal graph implies that all operators are unary, because operators with k effects introduce k-cliques in the causal graph.

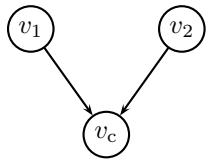

Figure 5: Causal graph of the running example (Fig. 1).

STRIPS tasks with acyclic causal graphs are rare, but the same is not necessarily true of SAS$^+$. For example, LOGISTICS tasks fall into this class, although more general transportation tasks involving capacity or fuel constraints do not. The causal graph of our first example is acyclic, as seen in Fig. 5. In fact, the example belongs to a more restricted class which we will now define.

Definition 4 ***SAS$^+$-1***
A ***SAS$^+$-1 task*** *is a SAS$^+$ task Π with a designated variable v such that $\mathrm{CG}(\Pi)$ has an arc from all other variables to v, and no other arcs. This variable v is called the* ***high-level variable****, whereas all other state variables are called* ***low-level variables****. A goal must be defined for the high-level variable, and goals must not be defined for the low-level variables.*[4]

[2] We write $f|A$ for the restriction of function f to set A, i. e. for the function $\{\ x \mapsto f(x) \mid x \in A \text{ and } f(x) \text{ is defined}\ \}$.

[3] The reference discusses STRIPS with negative preconditions, as deciding plan existence for plain propositional STRIPS is trivial if all operators are unary.

[4] The restriction on goals is only for convenience of presentation and could easily be lifted.

In SAS$^+$-1 tasks, all low-level variables can be changed independently, using operators that have no preconditions on other variables. The subtask induced by the set of low-level variables can thus be easily solved by basic graph search techniques. However, solving SAS$^+$-1 tasks is not easy, as we shall see in the following section.

We are interested in SAS$^+$-1 planning tasks because they are comparatively simple in structure, yet still expressive enough to capture complicated interactions between state variables. In the following sections, we will discuss the theoretical properties of SAS$^+$-1 planning and present some algorithmic techniques. These techniques form the building blocks for a planning heuristic for general SAS$^+$ tasks, which is presented afterwards.

For the following discussion, it is useful to introduce the notion of *domain transition graphs*. The domain transition graph of a state variable is a representation of the ways in which the variable can change its value, and of the conditions that must be satisfied for such value changes to be allowed. Apart from minor notational differences, our definition is identical to the one provided by Jonsson and Bäckström (1998).

Definition 5 ***domain transition graph***
Consider a SAS$^+$ task with variable set $\mathcal{V}$, and let $v \in \mathcal{V}$.

*The **domain transition graph** G_v is the labeled digraph with vertex set $\mathcal{D}_v$ which contains an arc (d, d') iff there is an operator $\langle pre, eff \rangle$ where $pre(v) = d$ or $pre(v)$ is undefined, and $eff(v) = d'$. The arc is labeled by $pre|(\mathcal{V} \setminus \{v\})$.*

*For each arc (d, d') with label L, we say that there is a **transition** of v from d to d' under the condition L.*

Multiple transitions between the same values using different conditions are possible. The domain transition graphs for our running example are shown in Figs. 6 and 7 (the graphs for v_1 and v_2 are identical). As the example indicates, arcs in domain transition graphs of low-level variables of SAS$^+$-1 tasks always have empty labels.

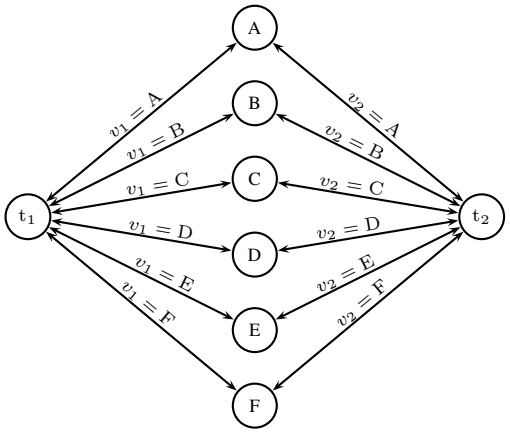

Figure 6: Domain transition graph for v_c.

If all operators of a task are unary, there is a strong correspondence between domain transition graphs, states, and

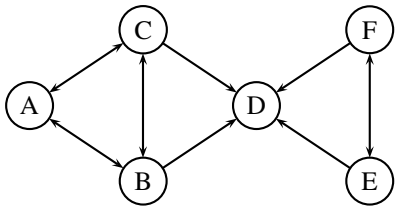

Figure 7: Domain transition graph for v_1 and v_2.

state transitions. A state is characterized by the values of the state variables, which in turn correspond to vertices in their respective domain transition graphs. For a given state s, we call $s(v)$ the *active vertex* of G_v. An operator application in a unary SAS$^+$ task changes the value of exactly one state variable, by changing the active vertex of some domain transition graph to one of its successors. Such a transition is allowed iff its conditions are satisfied, defining constraints on the active vertices of other domain transition graphs.

Plan execution can thus be viewed as simultaneous traversal of domain transition graphs, where a transition on a given graph can be allowed or forbidden depending on the active vertices of the other graphs. This is the view expressed in the paper by Domshlak and Dinitz (2001a). In the case of SAS$^+$-1 tasks, only transitions of the high-level variable can be constrained.

Theoretical Properties of SAS$^+$-1

Intuition suggests that SAS$^+$-1 planning should be easier than SAS$^+$ planning in general. This is in fact the case. While SAS$^+$ planning is **PSPACE**-complete, SAS$^+$-1 planning is easily seen to be in **NP**. However, the problem is not easy, as the following result shows. The proof is based on an earlier result by Domshlak and Dinitz (2001a; 2001b) for the closely related STS-$\mathbb{F}_n^{\vee^\pi}$ problem.

Theorem 6 **PLANEX-SAS$^+$-1** ***is* NP-*complete***
The plan existence problem for SAS$^+$-1 is **NP***-complete.*
Proof: Membership in **NP** follows from a polynomial plan length argument. In a minimal plan, the high-level variable does not assume the same value twice, and a low-level variable does not assume the same value twice unless the high-level variable was changed in between. Together, this leads to a quadratic length bound allowing a non-deterministic guess-and-check algorithm to run in polynomial time.

For hardness, we reduce from the **NP**-complete PFP (path with forbidden pairs) problem (Garey & Johnson 1979, problem GT54): Given a digraph (V, A), vertices $v_0, v_\star \in V$, and a set of arc pairs $P \subseteq A \times A$, is there a path from v_0 to $v_\star$ containing at most one arc from each pair in P?

The high-level variable v_H of the corresponding SAS$^+$-1 task has the domain V, initial value v_0 and goal value $v_\star$. For each forbidden pair $(a, b) \in P$, there is a low-level variable $v_{(a,b)}$ with domain $\{\bot, a, b\}$ and initial value $\bot$.

Low-level transitions allow changing the value of $v_{(a,b)}$ from $\bot$ to either a or b. For each arc $a = (u, v) \in A$, a high-level transition changes the value of v_H from u to v if all low-level variables which have a in their domain currently

assume that value. Thus, low-level transitions correspond to selecting the usable arc of each forbidden pair, and high-level transitions correspond to traversing the usable arcs.

This mapping is a polynomial reduction: If the planning task is solvable, then the high-level transitions of a solution plan define a legal path for the PFP instance. Conversely, if there is a path for the PFP instance, the low-level variables can be set accordingly so that the high-level variable can follow that path to solve the planning task. ∎

The result even holds if we restrict operators to define at most one precondition on low-level variables, because PFP is **NP**-complete even if all pairs are required to be disjoint.

However, some classes of SAS$^+$-1 tasks can be solved efficiently. For example, solutions are found in polynomial time by explicit state space search if the number of low-level variables is bounded by a constant.[5] Unfortunately, typical planning tasks, even those with acyclic causal graphs, do not have this property. For example, in the LOGISTICS domain, the causal graph indegree of the variables representing packages equals the number of vehicles, which can be fairly high.

Still, plan generation for LOGISTICS tasks is easy because of an additional property: The domain transition graphs of all state variables are strongly connected. Under this restriction, SAS$^+$-1 plans can be generated in polynomial time, since all values of the low-level variables can be achieved from any configuration, and thus all high-level transition conditions can be satisfied at will. The following result shows that the corresponding optimization problem is still hard.

Theorem 7 PLANLEN-SAS$^+$-1 *is* NP-*complete*
The bounded plan existence problem for SAS$^+$-1 (deciding whether a plan of a given maximum length M exists) is **NP***-complete, even if all domain transition graphs are strongly connected.*

Proof: Membership in **NP** follows from the previous result. For hardness, we reduce from the **NP**-complete X3C (exact cover by 3-sets) problem (Garey & Johnson 1979, problem SP2): Given a collection C of three-element subsets of the set $\{1, \ldots, 3q\}$, does C contain an *exact cover*, i. e. a subset C' of cardinality q with $\bigcup C' = \{1, \ldots, 3q\}$?

The corresponding SAS$^+$-1 task has a high-level variable v_{H} with domain $\{0, \ldots, 3q\}$, initial value 0 and goal value $3q$. For each set $S \in C$, there is a low-level variable v_S with domain $\{\bot, \top\}$, initially set to $\bot$. The low-level variables can change their value between $\bot$ and $\top$ arbitrarily; the high-level variable can change its value from i to $i+1$ if some low-level variable v_C with $i+1 \in C$ is currently set to $\top$. If S contains multiple sets containing $i+1$, there is one high-level transition for each. Finally, we add an unconditional transition from $3q$ to 0 to ensure strong connectivity. The maximum plan length M is set to $4q$.

To see that this is a polynomial reduction, observe that low-level transitions correspond to selecting elements for C' and high-level transitions correspond to checking that C' indeed covers all of $\{1, \ldots, 3q\}$. The length bound ensures that only q elements of C are selected. ∎

It is interesting to compare the previous result to LOGISTICS planning with a single vehicle (Helmert 2001), where domain transition graphs are also strongly connected and causal graphs have *one* low-level variable and *many* high-level variables. For these tasks, plan generation is also a polynomial problem, and bounded plan existence is also **NP**-complete.

A Planning Algorithm for SAS$^+$-1

In this section, we describe a polynomial algorithm for solving SAS$^+$-1 tasks. Because plan existence for SAS$^+$-1 is an **NP**-hard problem, the algorithm is not complete: While all generated plans are valid, there are solvable tasks for which no solution is found.

In the following, let $\langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be the given SAS$^+$-1 task. Let v_{H} be the high-level variable, with domain $\mathcal{D}_{\mathrm{H}}$.

The algorithm is based on Dijkstra's algorithm for the single-source shortest path problem. Thus, it tries to find a plan for achieving each value in $\mathcal{D}_{\mathrm{H}}$, not just the desired value $s_\star(v_{\mathrm{H}})$. Like Dijkstra's algorithm, it first finds the shortest plan for reaching any high-level value different to $s_0(v_{\mathrm{H}})$, then tries to reach the second-nearest value etc., until no further values are reachable or a plan to the goal is found. It is not complete because it does not reinvestigate the plans used for achieving a certain high-level value, although it might turn out that these lead to dead ends caused by unfortunate modifications to the low-level variables.

It helps to view the low-level variables as "resources" which are used to achieve a certain goal on the high-level variable. The algorithm tries to reach each high-level value as quickly as possible, without trying to preserve resources. Because it commits to a certain plan for achieving each high-level value, the complete world state is known during planning. The algorithm works as follows[6]:

1. Initialization:
   $$\mathit{plan}(d_{\mathrm{H}}) = \begin{cases} \langle\rangle & \text{if } d_{\mathrm{H}} = s_0(v_{\mathrm{H}}) \\ \text{undefined} & \text{otherwise} \end{cases}$$
   $\mathit{Queue} = \mathcal{D}_{\mathrm{H}}$
2. Let d_{H} be an element of *Queue* minimizing $\|\mathit{plan}(d_{\mathrm{H}})\|$ (undefined plans have cost ∞). Remove d_{H} from *Queue*. Let $\pi = \mathit{plan}(d_{\mathrm{H}})$ and let s be the state that results from applying π in the initial state.

   For all high-level transitions *op* originating from d_{H} with target d'_{H} and condition *pre*:

   (a) Check if it is possible to satisfy all conditions in *pre* starting from state s. If not, stop and consider the next transition. If yes, let π_{L} be a minimal cost plan achieving the preconditions. This can be computed by Dijkstra searches in the domain transition graphs of the low-level variables. Define π' as the concatenation of π, π_{L} and $\langle \mathit{op} \rangle$.

[5] The related (and more general) planning problem for STRIPS tasks with a polytree causal graph with bounded indegree is also known to be tractable (Domshlak & Brafman 2002).

[6] The notation $\|\pi\|$ denotes the *cost* of a plan. Usually, this is the same as its length (number of operators), but we will later need to consider cases in which operator costs are not uniform.

(b) If $\|plan(d'_{\mathrm{H}})\| > \|\pi'\|$, then set $plan(d'_{\mathrm{H}}) = \pi'$.

3. Repeat the previous step until *Queue* no longer contains any values for which *plan* is defined (in which case the algorithm fails), or until $s_\star(v_{\mathrm{H}})$ is removed from *Queue*, in which case $plan(s_\star(v_{\mathrm{H}}))$ is the solution.

We note that the algorithm is complete if the domain transition graphs of all low-level variables are strongly connected: In this case it is always possible to satisfy the preconditions of a high-level transition.

As an example, consider one of the three components of the simple transportation task (Fig. 2). The cargo item, represented by the high-level state variable v_{H}, must be moved from location D to location A with the help of the truck, represented by the (only) low-level state variable v_{L}. The domain of v_{H} is the set $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{t}\}$, where $v_{\mathrm{H}} = \mathrm{t}$ means that the cargo item is being carried by the truck. The domain of v_{L} is the set $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$. Legal operators are pick-up, drop, and truck movement actions, defined in the obvious way.

Fig. 8 shows the order in which the different values for the cargo variable are removed from the queue. It also denotes the high-level transitions (pick-ups and drops) that are considered for each value removed from the queue and shows the subplans generated during the planning process.

Initially, the queue contains the elements $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{t}\}$; we have $plan(\mathrm{D}) = \langle\rangle$ and no further plans are defined.

- Remove D from the queue $\rightsquigarrow s = \langle v_{\mathrm{H}} \mapsto \mathrm{D}, v_{\mathrm{L}} \mapsto \mathrm{A}\rangle$.
  - Consider $drop_{\mathrm{A}}$:
    Set $plan(\mathrm{A}) = \langle \mathrm{move}_{\mathrm{A,B}}, \mathrm{move}_{\mathrm{B,C}}, \mathrm{move}_{\mathrm{C,D}}, \mathrm{pickup}_{\mathrm{D}}, \mathrm{move}_{\mathrm{D,C}}, \mathrm{move}_{\mathrm{C,B}}, \mathrm{move}_{\mathrm{B,A}}, \mathrm{drop}_{\mathrm{A}}\rangle$.
  - Consider $drop_{\mathrm{B}}$:
    Set $plan(\mathrm{B}) = \langle \mathrm{move}_{\mathrm{A,B}}, \mathrm{move}_{\mathrm{B,C}}, \mathrm{move}_{\mathrm{C,D}}, \mathrm{pickup}_{\mathrm{D}}, \mathrm{move}_{\mathrm{D,C}}, \mathrm{move}_{\mathrm{C,B}}, \mathrm{drop}_{\mathrm{B}}\rangle$.
  - Consider $drop_{\mathrm{C}}$:
    Set $plan(\mathrm{C}) = \langle \mathrm{move}_{\mathrm{A,B}}, \mathrm{move}_{\mathrm{B,C}}, \mathrm{move}_{\mathrm{C,D}}, \mathrm{pickup}_{\mathrm{D}}, \mathrm{move}_{\mathrm{D,C}}, \mathrm{drop}_{\mathrm{C}}\rangle$.
  - Consider $drop_{\mathrm{D}}$: No improvement to $plan(\mathrm{D})$.
- Remove C from the queue $\rightsquigarrow s = \langle v_{\mathrm{H}} \mapsto \mathrm{C}, v_{\mathrm{L}} \mapsto \mathrm{C}\rangle$.
  - Consider $pickup_{\mathrm{C}}$: No improvement to $plan(\mathrm{t})$.
- Remove B from the queue $\rightsquigarrow s = \langle v_{\mathrm{H}} \mapsto \mathrm{B}, v_{\mathrm{L}} \mapsto \mathrm{B}\rangle$.
  - Consider $pickup_{\mathrm{B}}$: No improvement to $plan(\mathrm{t})$.
- Remove A from the queue $\rightsquigarrow$ return $plan(\mathrm{A})$.

Figure 8: Computing a plan for a subtask of the simple transportation task (Fig. 2).

Dead-End Detection for SAS$^+$-1

In this section, we investigate the complementary problem of proving SAS$^+$-1 tasks unsolvable. Again, the algorithm is sound but not complete: None of the tasks shown as unsolvable admit a solution, but some tasks without solution pass unnoticed.

Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star\rangle$ be the given SAS$^+$-1 task, with high-level variable v_{H} and low-level variables $\mathcal{V}_{\mathrm{L}}$. We assume that the domains of all variables are disjoint, which can be achieved by renaming. The domain of v_{H} is written as $\mathcal{D}_{\mathrm{H}}$, the union of the domains of $\mathcal{V}_{\mathrm{L}}$ as $\mathcal{D}_{\mathrm{L}}$.

The solution algorithm from the previous section is overly strict because it identifies a single world state with each value of the high-level variable. Complementary to this, we now associate high-level variables with many world states, some of which might not actually be feasible. More precisely, we keep track whether or not each pair $(d_{\mathrm{H}}, d_{\mathrm{L}}) \in \mathcal{D}_{\mathrm{H}} \times \mathcal{D}_{\mathrm{L}}$ is reachable and assume that a state s is reachable whenever for each low-level variable v_{L}, the pair $(s(v_{\mathrm{H}}), s(v_{\mathrm{L}}))$ is reachable. In words, if the high-level value/low-level value pairs can be achieved individually, we assume that they can be achieved simultaneously.

Using this restriction, we obtain the following polynomial algorithm for computing the set R of reachable pairs.

1. Initialization: $R := \{\ (s_0(v_{\mathrm{H}}), s_0(v_{\mathrm{L}})) \mid v_{\mathrm{L}} \in \mathcal{V}_{\mathrm{L}}\ \}$.
2. For each low-level transition from d_{L} to d'_{L}:
   $R := R \cup \{\ (d_{\mathrm{H}}, d'_{\mathrm{L}}) \mid (d_{\mathrm{H}}, d_{\mathrm{L}}) \in R\ \}$.
3. For each high-level transition from d_{H} to d'_{H} with condition $\{v_1 \mapsto d_1, \ldots, v_k \mapsto d_k\}$ such that $(d_{\mathrm{H}}, d_i) \in R$ for all $i = 1, \ldots, k$:
   $R := R \cup \{\ (d'_{\mathrm{H}}, d_1), \ldots, (d'_{\mathrm{H}}, d_k)\ \}$
   $\quad\cup \{\ (d'_{\mathrm{H}}, d_{\mathrm{L}}) \mid (d_{\mathrm{H}}, d_{\mathrm{L}}) \in R \wedge d_{\mathrm{L}} \notin \mathcal{D}_{pre}\ \}$,
   where $\mathcal{D}_{pre}$ is the union of the domains of $\{v_1, \ldots, v_k\}$.
4. Fixpoint iteration: Repeat the two previous steps until R no longer changes. The task is considered solvable iff $(s_\star(v_{\mathrm{H}}), d_{\mathrm{L}}) \in R$ for any $d_{\mathrm{L}} \in \mathcal{D}_{\mathrm{L}}$.

The third step is the only one which needs some explanation. First, we check whether a transition from d_{H} to d'_{H} is applicable. This is the case whenever all values in the condition may occur together with d_{H}. If the transition is applicable, then we can apply it to change the value of the high-level variable to d'_{H}. In the resulting state, the low-level variables mentioned in the condition will have the values dictated by the condition (first line of the update of R), while all other low-level variables retain their values (second line of the update of R).

Of course, in practice the algorithm should not be implemented using the naïve update method we described. One possible implementation encodes the update rules in Horn Logic, using a propositional variable $R_{(d_{\mathrm{H}}, d_{\mathrm{L}})}$ for each possible element of R. The resulting formula has $\mathrm{O}(|\mathcal{O}| \cdot |\mathcal{D}_{\mathrm{H}}|)$ clauses of fixed length for the update rules in the second step and $\mathrm{O}(|\mathcal{O}| \cdot |\mathcal{D}_{\mathrm{L}}|)$ clauses of length bounded by the maximal number of preconditions per operator for the update rules in the third step, for a total bound of $O(\|\Pi\|^2)$. The relevant reasoning problem in Horn Logic can be solved in linear time in the formula size, yielding a quadratic algorithm.

Fig. 9 shows that the algorithm can prove the example task of Fig. 1 unsolvable. In this case, a stable value for the set R is reached after the third iteration. The result set does

not contain the pair $(\mathrm{B}, d_\mathrm{L})$ for any low-level value d_L, so the cargo item cannot be transported to location B.

$$
\begin{aligned}
R_1 &= \{(\mathrm{E}, \mathrm{C}_1), (\mathrm{E}, \mathrm{F}_2)\} \\
R_2 &= R_1 \cup \{(\mathrm{E}, \mathrm{A}_1), (\mathrm{E}, \mathrm{B}_1), (\mathrm{E}, \mathrm{D}_1), (\mathrm{E}, \mathrm{D}_2), (\mathrm{E}, \mathrm{E}_2)\} \\
R_3 &= R_2 \cup \{(\mathrm{t}_2, \mathrm{E}_2), (\mathrm{t}_2, \mathrm{A}_1), (\mathrm{t}_2, \mathrm{B}_1), (\mathrm{t}_2, \mathrm{C}_1), (\mathrm{t}_2, \mathrm{D}_1)\} \\
R'_2 &= R_3 \cup \{(\mathrm{t}_2, \mathrm{D}_2), (\mathrm{t}_2, \mathrm{F}_2))\} \\
R'_3 &= R'_2 \cup \{(\mathrm{D}, \mathrm{D}_2), (\mathrm{D}, \mathrm{A}_1), (\mathrm{D}, \mathrm{B}_1), (\mathrm{D}, \mathrm{C}_1), (\mathrm{D}, \mathrm{D}_1)\} \\
&\qquad \cup \{(\mathrm{F}, \mathrm{F}_2), (\mathrm{F}, \mathrm{A}_1), (\mathrm{F}, \mathrm{B}_1), (\mathrm{F}, \mathrm{C}_1), (\mathrm{F}, \mathrm{D}_1)\} \\
R''_2 &= R'_3 \cup \{(\mathrm{F}, \mathrm{D}_2), (\mathrm{F}, \mathrm{E}_2))\} \\
R''_3 &= R''_2 \cup \{(\mathrm{t}_1, \mathrm{D}_1), (\mathrm{t}_1, \mathrm{D}_2)\} \\
R'''_2 &= R''_3 \\
R'''_3 &= R''_3 \rightsquigarrow \text{fixpoint}
\end{aligned}
$$

Figure 9: Proving the running example (Fig. 1) unsolvable. To disambiguate the location values for the two trucks, indices 1 and 2 are used. The indices for R correspond to the numbers of the algorithm steps 1., 2., and 3.

Putting the Pieces Together

We did not discuss SAS$^+$-1 planning for the reason that it is interesting in its own right (although we think it is), but rather because we want to use the algorithms derived in the previous sections to find plans for general SAS$^+$ tasks.

When considering planning tasks with a complex causal structure, it is most important to capture the interactions between state variables which are connected by an arc in the causal graph. Interactions between state variables where one is a remote ancestor of the other are less critical because they are mediated by other state variables. State variables where neither is an ancestor of the other cannot interact at all except via a common ancestor. Thus, instead of solving the planning task as a whole, we only consider small local portions at a time, namely the subtasks induced by one state variable and its immediate predecessors in the causal graph. Each of these subtasks is a SAS$^+$-1 task.

By splitting the planning task into parts like this, we no longer obtain a solution algorithm, because typically the solutions to the parts cannot be fitted together to form a whole. However, we can combine the lengths of the solutions of the parts to form a *distance heuristic* for the overall task. To this end, we compute an estimate for the *cost* of changing the value of state variable v from a given value d to another value d', written as $\mathit{cost}_v(d, d')$, where $d, d' \in \mathcal{D}_v$.

Naturally, the cost of changing the value of v depends on the cost of changing the values of its causal predecessors. The following algorithm for calculating $\mathit{cost}_v(d, d')$ reflects this. The planning task is again given as $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$.

1. If v has no causal predecessors, $\mathit{cost}_v(d, d')$ is the length of the shortest path from d to d' in the domain transition graph G_v, or ∞ if no such path exists. Otherwise, continue.
2. Let $\mathcal{V}_v$ be the set consisting of v and all predecessors of v in the causal graph of Π. Let Π_v be the planning task induced by $\mathcal{V}_v$ except that the initial value of v is set to d and the goal value of v is set to d'.
3. $\mathit{cost}_v(d, d') = \|\pi\|$, where π is the plan for Π_v found by the SAS$^+$-1 planning algorithm. Here, all high-level transitions have cost 1 and a low-level transition of variable v_L from d_L to d'_L has cost $\mathit{cost}_{v_\mathrm{L}}(d_\mathrm{L}, d'_\mathrm{L})$.

Having computed the individual *cost* values, the total cost for achieving the goal from a state s is defined as the sum of $\mathit{cost}_v(s(v), s_\star(v))$ over all variables v for which the goal $s_\star$ is defined.

As a simple example of how the decomposition into SAS$^+$-1 tasks works, consider Fig. 2 for the last time. Goals are defined for all cargo items, so for each cargo item variable, we need to consider the subtask induced by that variable and its sole causal predecessor, the variable of the truck that is located in the same component of the roadmap.[7] In Fig. 8, we have shown that the SAS$^+$-1 planner optimally solves tasks of this kind. Adding the costs for the individual subtasks together, we get an optimal heuristic for the overall planning task in this case.

The observant reader will notice that the above algorithm can only compute the transition costs in the case where the causal graph is acyclic. Otherwise, the costs for variable v can depend on the costs of variable v', which can in turn depend on the costs for variable v. There are several ways of solving this problem. One is initializing all costs to ∞ and then updating the costs iteratively until a fixpoint is reached.

A second, more radical approach is to ignore some preconditions during the heuristic computation so that cycles in the causal graph are broken. In our experiments, we used the following strategy: When variables v, v' are part of a cycle in the causal graph and there is a transition for variable v' with a condition on variable v, then this condition is ignored if v is considered *higher-level* than v'. We consider v higher-level than v' iff v is a precondition of fewer operators than v'. Thus, as many preconditions as possible are respected.

Somewhat surprisingly, in our implementation the more radical approach leads to the better planner. Although typically more state evaluations are needed, faster computation of the heuristic value results in better overall performance.

Because of space restriction, we do not discuss the generalization of the dead end detection algorithm to arbitrary SAS$^+$ tasks in detail. It is very similar to the generalization of the planning algorithm.

Experimental Results

We implemented the heuristic described in the previous section in a forward planner using best-first search, which we called CG for "causal graph". We tested CG on all STRIPS benchmarks from the 1998, 2000, and 2002 planning competitions, totaling 550 planning tasks. When the heuristic

[7] Depending on the way the task has been specified, the other truck variables could also be causal predecessors. Whether or not this is the case does not affect the heuristic estimate, since these trucks can never pick up this cargo item and thus cannot interfere.

evaluator classified a state as unsolvable, this was verified using the dead end detection algorithm. The heuristic may consider a state unsolvable while the dead end detection does not; in this case, the state was not explored further by CG, but a failure of the search was not counted as a proof that no solution exists. Thus, we have to consider four possible outcomes of the planning process:

- A plan is found. This is counted as a solved task.
- The search space is explored until all frontier states are assigned a heuristic value of ∞ and verified to be dead ends. This is counted as a task proven unsolvable.
- The search space is explored until all frontier states are assigned a heuristic value of ∞, but some of them are not verified to be dead ends. This is counted as a failure.
- The planner exhausts the time or memory bound. This is counted as a failure.

The third outcome was never observed in the experiments. The benchmark suite contains eleven unsolvable tasks (all in the MYSTERY domain), for all of which CG could prove that the initial state is already a dead end. All other tasks are solvable.

We conducted two experiments. Both were run on a 3 GHz Linux machine where planners were allowed to use up to 1 GB RAM and 5 minutes CPU time. In the first experiment, we compared the quality of the heuristic function used by CG to other heuristics which have proved successful in other planners. Towards this end, we reimplemented the FF heuristic and HSP's h_{add} heuristic in our best-first-search planner, resulting in two planners we call CG$^{\text{FF}}$ and CG$^{\text{HSP}}$. Note that we *only* changed the heuristic estimator.

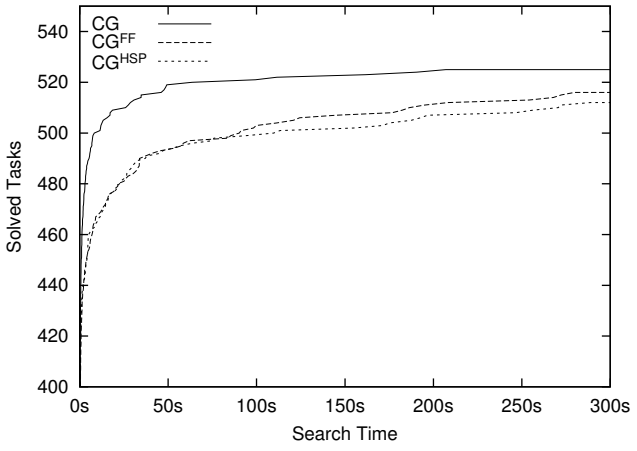

Figure 10: Comparison of CG, FF and HSP heuristics.

All three planners used the same best-first search technique, and all applied the dead end detection routine to the initial state of each task, identifying the unsolvable MYSTERY instances.[8] Because the three planners only differ in their search component, we only measured pure search time (rather than total running time) in the first experiment.

The results are depicted in Fig. 10, where the number of tasks solved is drawn as a function of time. For example, the graph for CG$^{\text{FF}}$ passes through the point $(100\text{s}, 503)$ because 503 (of 550) benchmarks needed at most 100 seconds of search time to solve using the FF heuristic.

Within the five minute deadline, CG failed to solve 25 tasks, CG$^{\text{FF}}$ failed to solve 34 tasks, and CG$^{\text{HSP}}$ failed to solve 38 tasks. Fig. 11 shows how many tasks from which planning domain each planner could not solve (the two rightmost columns of the figure are explained further below).

Domain	CG	CG$^{\text{FF}}$	CG$^{\text{HSP}}$	FF	LPG
BLOCKSWORLD (35)	0	0	0	4	0
DEPOT (22)	14	10	11	3	0
DRIVERLOG (20)	3	2	3	5	0
FREECELL (80)	2	9	11	5	74
GRID (5)	1	1	2	0	1
LOGISTICS (63)	0	0	0	0	4
MPRIME (35)	1	6	6	3	7
MYSTERY (30)	1	1	0	12	15
ROVERS (20)	3	3	5	0	0
SATELLITE (20)	0	2	0	0	0
Total	**25**	**34**	**38**	**32**	**101**

Figure 11: Unsolved tasks by planner and domain. Numbers in parentheses denote total number of tasks in a domain.

All heuristics were computed at comparable speeds, so the differences in performance were mainly due to differences in quality. To measure this, we compared the number of state expansions for all tasks solved by all three heuristics. It turned out that for the overwhelming majority of tasks, very few search nodes were necessary to find a solution using *any* heuristic, making comparisons based on node expansions difficult. For about two thirds of the solved tasks, less than 100 nodes were expanded, and for about 90% of them, less than 1000 nodes were expanded (independent of the heuristic). Comparing the most successful planners CG and CG$^{\text{FF}}$ on a domain-by-domain basis, CG expanded many nodes less in the SATELLITE and ZENOTRAVEL domains and many nodes more for GRIPPER tasks.

Differences in plan quality were very slight: Among the tasks solved by all three heuristics, the average plan lengths were 44.5 for CG, 43.6 for CG$^{\text{FF}}$, and 45.8 for CG$^{\text{HSP}}$.

In our second experiment, we compared the performance of CG to other domain-independent planning systems from the literature. The experiment showed that two planners, FF and LPG, could solve considerably more tasks than the other systems we tested, so only these two planners are shown in the experimental results (Fig. 12, this time listing total running time for CG, not search time).[9]

The figure shows that CG finds more plans within seconds than LPG finds at all, and that it finds more plans in one

[8] This slightly favors the FF and HSP heuristics, which would only identify two of these tasks as unsolvable and fail on the other nine without this help. The CG heuristic assigns an infinite heuristic estimate to the initial states of all these tasks.

[9] LPG was run using the "fast" competition settings. Because of the large random variation in LPG's performance, we solved each task five times and kept the result with median completion time.

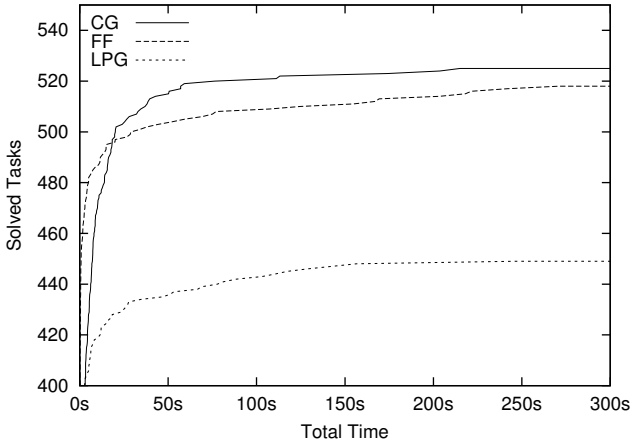

Figure 12: Comparison of the CG, FF and LPG planners.

Domain	easy	median	difficult
BLOCKSWORLD	0.07s	0.17s	0.51s
DEPOT	0.13s	1.21s	1.61s
DRIVERLOG	0.10s	0.20s	0.53s
FREECELL	3.31s	6.84s	16.05s
GRID	1.72s	4.16s	14.14s
GRIPPER	0.11s	0.54s	2.01s
LOGISTICS-1998	0.49s	2.04s	7.24s
LOGISTICS-2000	0.06s	0.09s	0.13s
MICONIC	0.08s	0.26s	0.59s
MOVIE	0.04s	0.04s	0.05s
MPRIME	0.77s	2.32s	9.59s
MYSTERY	0.19s	0.91s	2.76s
ROVERS	0.06s	0.18s	0.53s
SATELLITE	0.16s	0.64s	2.25s
ZENOTRAVEL	0.12s	0.32s	2.67s

Figure 13: CG solution times.

minute than FF does in five minutes. However, the comparatively low numbers for LPG are mainly due to the FREECELL domain and should not be overinterpreted. Fig. 11, which shows the number of unsolved tasks by planner and domain, provides a clearer picture of the different strengths and weaknesses of the planners we tested.

The average plan length was 50.3 for CG, 42.1 for FF, and 54.6 for LPG (only considering tasks solved by all three planners). Thus, CG produced shorter plans than LPG but longer plans than FF. In detail, CG produced much longer plans than FF in the domains BLOCKSWORLD, DEPOT, GRIPPER, LOGISTICS and MICONIC. Since CG and CG^{FF} computed plans of similar quality, we attribute this difference in plan quality to the search strategy, not the heuristic.

We close our presentation of results with Fig. 13, which gives an impression of the scaling behavior of CG in the individual domains. For each domain, the table lists the solution time for "easy", "median" and "difficult" tasks. These values were obtained by sorting the solved tasks by solution time and taking the values at 25% (easy), 50% (median) and 75% (difficult) of the spectrum. For example, if 40 tasks were solved, the 10th, 20th and 30th position are reported.

We conclude that CG compares favorably with the state of the art in STRIPS planning. We believe that the performance of the planner can be enhanced further by adapting some of the search enhancements popularized by FF, namely helpful action pruning and goal agenda management. More than half of the tasks that CG cannot solve belong to the DEPOT domain. The fact that CG^{FF} is also weak in the DEPOT domain while FF is much better suggests that FF's better search control is very useful for these tasks.

Discussion

We have presented a novel approach to heuristic planning that is based on decomposing the causal graph of a translated planning task into a sequence of easier SAS^+-1 tasks. We provided an in-depth analysis of SAS^+-1 planning and showed that a heuristic planning system based on SAS^+-1 tasks is competitive with the state of the art.

Interestingly, Fig. 11 shows that CG has distinctly other weaknesses than FF or LPG. To us, this signifies that alternative ways of relaxing planning tasks have not yet been explored in sufficient depth.

We see three major options for building on the results presented in this paper:

- The search control and speed of the planner could be improved. For example, FF's *helpful actions* can be adapted into this context. Considering the difficulties with the DEPOT domain, goal ordering techniques should be tried. The speed of the planner could be improved by reusing heuristic estimates for subproblems that are encountered repeatedly during search.
- The planner could be integrated with an FF-like approach to combine the strengths of the CG and FF heuristics in an orthogonal way.
- The planner could be extended to more general planning formalisms, such as ADL, planning with domain axioms, or planning with (limited) numerical state variables.

We have made some encouraging first steps in the first two directions. Experiments with helpful actions have led to an improved planner which solves all but eleven of the benchmark tasks, all unsolved instances being from the DEPOT and FREECELL domains.

In another experiment, we have implemented a planner which computes both the CG and FF estimates of each state and alternates between expanding the open state with lowest CG value and the open state with lowest FF value. This planner was able to solve many of the tasks on which CG failed. Interestingly, it could also solve some tasks where both CG and FF failed, indicating that the two heuristics can sometimes be combined orthogonally.

References

Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence* 71(1):43–100.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In Kuipers, B. J., and Webber, B., eds., *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 714–719. AAAI Press.

Domshlak, C., and Brafman, R. I. 2002. Structure and complexity in planning with unary operators. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 34–43. AAAI Press.

Domshlak, C., and Dinitz, Y. 2001a. Multi-agent off-line coordination: Structure and complexity. In Cesta, A., and Borrajo, D., eds., *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*, 277–288. Toledo, Spain: Morgan Kaufmann.

Domshlak, C., and Dinitz, Y. 2001b. Multi-agent off-line coordination: Structure and complexity. Technical Report CS-01-04, Ben-Gurion University of the Negev.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In Fox, M., and Biundo, S., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 135–147. New York: Springer-Verlag.

Edelkamp, S., and Helmert, M. 2000. On the implementation of MIPS. Paper presented at the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Workshop on Model-Theoretic Approaches to Planning. Breckenridge, Colorado, 14 April.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:367–421.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman.

Gerevini, A., and Schubert, L. 1998. Inferring state constraints for domain-independent planning. In Rich, C., and Mostow, J., eds., *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 905–912. AAAI Press.

Helmert, M. 2001. On the complexity of planning in transportation domains. In Cesta, A., and Borrajo, D., eds., *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*, 349–360. Toledo, Spain: Morgan Kaufmann.

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

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 92–100. AAAI Press.

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1–2):125–176.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Refanidis, I., and Vlahavas, I. 1999. GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In Fox, M., and Biundo, S., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 347–359. New York: Springer-Verlag.

Williams, B. C., and Nayak, P. P. 1997. A reactive planner for a model-based executive. In Pollack, M. E., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1178–1195. Morgan Kaufmann.