In ICAPS-15 Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)

A Heuristic Estimator based on Cost Interaction

Yolanda E-Martín[1,2] and **María D. R-Moreno**[1] and **David E. Smith**[3]
[1] Departamento de Automática. Universidad de Alcalá. Ctra Madrid-Barcelona, Km. 33,6 28871 Alcalá de Henares (Madrid), Spain.
{yolanda,mdolores}@aut.uah.es
[2] Universities Space Research Association. 615 National Ave, Suite 220, Mountain View, CA 94043
[3] Intelligent Systems Division. NASA Ames Research Center. Moffett Field, CA 94035-1000
david.smith@nasa.gov

Abstract

In planning with action costs, heuristic estimators are commonly used to guide the search towards lower cost plans. Some admissible cost-based heuristics are weak (non-informative), while others are expensive to compute. In contrast, non-admissible cost-based heuristics are in general more informative, but may overestimate the computed cost, yielding non-optimal plans. This paper introduces a domain-independent non-admissible heuristic for planning with action costs that computes more accurate cost estimates. This heuristic is based on cost propagation in a plan graph, but uses *interaction* to compute information about the relationships between pairs of propositions and pairs of actions to calculate more accurate cost estimates. We show the trade-off between the solution quality and the planning performance when applying this heuristic in classical planning. This heuristic is expensive for planning, but we demonstrate that it is quite useful for goal recognition.

Introduction

In planning with action costs, heuristic estimators are commonly used to guide the search towards lower cost plans. Some admissible cost-based heuristics, like the max heuristic h^{max} (Bonet and Geffner, 2001), are weak (non-informative), while others, like the higher-order heuristics h^m (Haslum and Geffner, 2000), are expensive to compute. In contrast, non-admissible cost-based heuristics, like the additive heuristic h^{add} (Bonet and Geffner, 2001) and the set-additive heuristic h^s_a (Keyder and Geffner, 2007), are more informative and have shown good performance in time and solution quality. However, they may overestimate the computed cost in some cases, yielding non-optimal plans.

In this paper, we introduce a domain-independent non-admissible heuristic for planning with action costs that computes more accurate cost estimates. This heuristic is based on cost propagation in a plan graph, but we use *interaction* (Bryce and Smith, 2006) to compute information about the relationships between pairs of propositions and pairs of actions, yielding more accurate cost estimates. We show the trade-off between the solution quality and the planning performance when applying this heuristic in classical planning. This heuristic is expensive for planning, but we demonstrate it is quite worthwhile for goal recognition.

This paper begins with a review of cost propagation in a plan graph. Then, we introduce *interaction* and its use in cost propagation in a plan graph. Then we describe two cost-based heuristic estimators and evaluate their accuracy. We present an empirical study of the heuristics in classical planning and goal recognition.

Cost propagation in a plan graph

Plan graphs (Blum and Furst, 1997) provide an optimistic method to estimate the set of achievable propositions, and the set of feasible actions, for a particular starting state and goal state. They have been commonly used to compute heuristic distance estimates between states and goals and, recently, to compute estimates of the cost that propositions can be achieved, and an action can be performed. Propagation of cost estimates in a plan graph is a technique that has been used in a number of planning systems (Bonet, Loerincs, and Geffner, 1997; Nguyen, Kambhampati, and Nigenda, 2002; Do and Kambhampati, 2002). The computation of cost estimates starts from the initial conditions and works progressively forward through each successive layer of the plan graph. For level 0 it is assumed that the cost of the propositions is zero. With this assumption, the propagation starts by computing the cost of the actions at level zero. In general, the cost of performing an action a at level l with a set of preconditions $\mathcal{P}_a$ is equal to the cost of achieving its preconditions. This may be computed in two different ways: (1) Maximization: the cost of an action is equal to the cost of reaching its costliest precondition, i.e., $cost(a) = \max_{x_i \in \mathcal{P}_a} cost(x_i)$, (2) Summation: the cost of an action is equal to the cost of reaching all its preconditions, i.e., $cost(a) = \sum_{x_i \in \mathcal{P}_a} cost(x_i)$. The first method is admissible since it underestimates the cost of an action. This allows for dependence among the preconditions of an action. In contrast, the second method is inadmissible since it assumes independence among all preconditions of an action. As a consequence, the cost may be underestimated, if some of the preconditions interfere with each other, or overestimated, if some of the preconditions are achieved by a common action.

The cost of achieving a proposition x at level l, achieved by actions $\mathcal{A}_x$ at the preceding level, is the minimum cost among all $a \in \mathcal{A}_x$. It is defined as $cost(x) = \min_{a \in \mathcal{A}_x} [cost(a) + Cost_a]$, where $Cost_a$ is the cost of ap-

plying the action a.

Taking the above calculations into consideration, a cost-plan graph is built in the same way that an ordinary planning graph is created. The construction process finishes when two consecutive proposition layers are identical and there is quiescence in cost for every proposition and action in the plan graph. On completion, each possible goal proposition has an estimated cost.

Our work focuses on computing more accurate estimates of cost by introducing the concept of interaction in the cost propagation in a plan graph. This primarily impacts the calculation of cost for sets of preconditions, although the consequences propagate through to propositions. We describe the calculation and use of interaction in the next section.

Interaction

Interaction is a value that represents how more or less costly it is that two propositions or actions are established together instead of independently. This concept is a generalization of the mutual exclusion concept used in classical planning graphs. Formally, the optimal Interaction, I^*, considers n-ary interaction relationships among propositions and among actions (p_0 to p_n) in the plan graph, and it is defined as:

$$I^*(p_0, ..., p_n) = cost^*(p_0 \wedge ... \wedge p_n) - (cost^*(p_0) + ... + cost^*(p_n)) \quad (1)$$

where the term $cost^*(p_0 \wedge ... \wedge p_n)$ is the minimum cost among all the possible plans that achieve all the members in the set. Computing I^* would be computationally prohibitive. As a result, we limit the calculation of these values to pairs of propositions and pairs of actions in each level of a plan graph – in other words, binary interaction:

$$I^*(p, q) = cost^*(p \wedge q) - (cost^*(p) + cost^*(q)) \quad (2)$$

It has the following features:

$$I^*(p,q) \ \ is \ \begin{cases} < 0 & \text{if } p \text{ and } q \text{ are } \textit{synergistic} \\ = 0 & \text{if } p \text{ and } q \text{ are } \textit{independent} \\ > 0 & \text{if } p \text{ and } q \textit{ interfere} \end{cases}$$

In other words, I provides information about the degree of interference or synergy between pairs of propositions and pairs of actions in a plan graph. When $0 < I(p, q) < \infty$ it means that there is some interference between the best plans for achieving p and q so it is harder (more costly) to achieve them both than to achieve them independently. In the extreme case, $I = \infty$, the propositions or actions are mutually exclusive. Similarly, $I(p, q) < 0$ (synergy) means that the cost of establishing both p and q is less than the sum of the costs for establishing the two independently. However, this cost cannot be less than the cost of establishing the most difficult of p and q. As a result $I(p, q)$ is bounded below by $-\min[cost(p), cost(q)]$.

Instead of computing mutex information, the new cost propagation technique computes interaction information between all pairs of propositions and all pairs of actions at each level. Interaction is taken into account in the cost propagation, which establishes a better estimation of the cost for two propositions or two actions performed at the same time.

The computation of cost and interaction information begins at level zero of the plan graph and sequentially proceeds to higher levels. For level zero it is assumed that (1) the cost of each proposition at this level is 0, and (2) the interaction between each pair of propositions at this level is 0, which means independence between propositions. Neither of these assumptions are essential, but they are adopted here for simplicity.

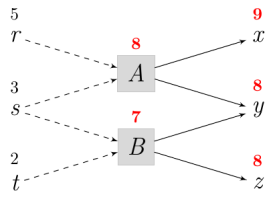


Figure 1: A cost plan graph level with cost and interaction calculation and propagation.

Computing action cost and interaction

The cost and interaction information of a proposition layer at a given level of the plan graph is used to compute the cost and the interaction information for the subsequent action layer. Considering an action a at level l with a set of preconditions $\mathcal{P}_a$, the cost of an action is approximated as the cost that all the preconditions are achieved plus the interaction between all pairs of preconditions:

$$cost^*(a) = cost^*(\mathcal{P}_a) \approx \sum_{x_i \in \mathcal{P}_a} cost(x_i) + \sum_{\substack{(x_i, x_j) \in \mathcal{P}_a \\ j > i}} I(x_i, x_j) \quad (3)$$

As an example, consider one level of the cost-plan graph shown in Figure 1. There are three propositions r, s, and t with costs 5, 3, and 2 respectively, and interaction values $I(r, s) = 0$, $I(r, t) = -1$, and $I(s, t) = 2$. There are two actions A and B, which have cost 1. The number above the propositions and actions are the costs computed for each one during the cost propagation process (those highlighted are the cost that we will compute in this section). For this example, the costs of actions A and B are:

$$\begin{aligned} cost(A) &\approx cost(r) + cost(s) + I(r, s) = 5 + 3 = 8 \\ cost(B) &\approx cost(s) + cost(t) + I(s, t) = 3 + 2 + 2 = 7 \end{aligned}$$

The next step is to compute the interaction between actions, which is defined as follows:

$$I^*(a, b) = \begin{cases} \infty & \text{if } a \text{ and } b \text{ are mutex by inconsistent effects or interference} \\ cost^*(a \wedge b) - cost^*(a) - cost^*(b) & \text{otherwise} \end{cases}$$

where $cost^*(a \wedge b)$ is defined to be $cost(\mathcal{P}_a \cup \mathcal{P}_b)$. This is approximated as in (3) by:

$$cost(\mathcal{P}_a \cup \mathcal{P}_b) \approx \sum_{x_i \in \mathcal{P}_a \cup \mathcal{P}_b} cost(x_i) + \sum_{\substack{(x_i, x_j) \in \mathcal{P}_a \cup \mathcal{P}_b \\ j > i}} I(x_i, x_j)$$

If the actions are mutex by inconsistent effects, or interference, then the interaction is ∞. Otherwise, the interaction

above simplifies to:

$$I(a,b) \approx \sum_{\substack{x_i \in \mathcal{P}_a - \mathcal{P}_b \\ x_j \in \mathcal{P}_b - \mathcal{P}_a}} I(x_i,x_j) - \Big[\sum_{x_i \in \mathcal{P}_a \cap \mathcal{P}_b} cost(x_i) + \sum_{\substack{(x_i,x_j) \in \mathcal{P}_a \cap \mathcal{P}_b \\ j>i}} I(x_i,x_j) \Big]$$

For the example in Figure 1, the interaction between actions A and B reduces to:

$$I(A,B) \approx I(r,t) - cost(s) = -1 - 3 = -4$$

The fact that $I(A,B) = -4$ means that there is some degree of synergy between both actions. This synergy comes from the fact that they have a common precondition s.

Computing proposition cost and interaction

The next step consists of calculating the cost of the propositions at the next level. In this calculation, all the possible actions at the previous level that achieve each proposition are considered. We make the usual optimistic approximation that the least expensive action can be used. Therefore, the cost of a proposition is the minimum over the costs of all the actions that can achieve the proposition. More formally, for a proposition x at level l, achieved by actions $\mathcal{A}_x$ at the preceding level, the cost is calculated as:

$$cost^*(x) = \min_{a \in \mathcal{A}_x} [cost(a) + Cost_a] \quad (4)$$

In our example, the cost of proposition y of the graph is:

$$\begin{aligned} cost(y) &= \min[cost(A) + Cost_A, cost(B) + Cost_B] \\ &= \min[8+1, 7+1] = 8 \end{aligned}$$

Finally, the interaction between a pair of propositions x and y is computed. In order to compute the interaction between two propositions at a level l, all possible ways of achieving those propositions at the previous level are considered. In other words, all the actions that achieve the pair of propositions are considered. Suppose that $\mathcal{A}_x$ and $\mathcal{A}_y$ are the sets of actions that achieve propositions x and y respectively at level l. The interaction between x and y is then:

$$\begin{aligned} I^*(x,y) &= cost^*(x \wedge y) - cost^*(x) - cost^*(y) \\ &= \min_{\substack{a \in \mathcal{A}_x \\ b \in \mathcal{A}_y}} \Big\{ cost(a \wedge b) \Big\} - cost^*(x) - cost^*(y) \\ &\approx \min \left\{ \begin{array}{l} \min\limits_{a \in \mathcal{A}_x \cap \mathcal{A}_y} cost(a) + Cost_a \\ \\ \min\limits_{\substack{a \in \mathcal{A}_x - \mathcal{A}_y \\ b \in \mathcal{A}_y - \mathcal{A}_x}} \left[\begin{array}{l} cost(a) + Cost_a + \\ cost(b) + Cost_b + \\ I(a,b) \end{array} \right] \end{array} \right\} \\ & \quad - cost(x) - cost(y) \end{aligned}$$

In our simple example, consider the calculation of the interaction between x and y where the possible ways to achieve both are performing A or A and B. Only the first of these possibilities is considered, since In this case, $\mathcal{A}_x - \mathcal{A}_y$ is empty. The interaction is therefore simply:

$$\begin{aligned} I(x,y) &\approx [cost(A) + Cost_A] - cost(x) - cost(y) \\ &= [8+1] - 9 - 8 = -8 \end{aligned}$$

Heuristic estimator based on interactions

The cost-plan graph described in the previous section includes, for each proposition and action in the plan graph, an approximate cost of achievement. These cost estimates may be used as heuristic estimators. The next subsections describe two different methods that make use of this information to compute heuristic estimates. The first method computes the estimated cost using the information given in the plan graph, while the second one builds a relaxed plan where the propagated cost information is taken into account in the relaxed-plan construction.

The h^I heuristic

The h^I heuristic is based directly on the cost and interaction information computed in the cost-plan graph. It defines the estimated cost of achieving a (conjunctives) goal $G = \{g_1, ..., g_n\}$ as:

$$h^I = cost(G) \approx \sum_{g_i \in G} \left[cost(g_i) + \sum_{\substack{(g_i,g_j) \in G \\ j<i}} I(g_i,g_j) \right] \quad (5)$$

The interaction information helps to compute more accurate estimates of cost when subgoals interfere with each other. However, the fact that the interaction computation is binary makes the heuristic h^I non-admissible. Therefore, the estimated cost is an approximation of the optimal cost. Essentially, when all the preconditions of each action are independent of each other, the heuristic h^I reduces to the heuristic h^{add}. Otherwise, h^I will be greater or less than h^{add} depending on whether the interaction is negative or positive.

The h^I_{rp} heuristic

The h^I_{rp} heuristic is based on computing a relaxed plan with the use of cost information in the plan graph. The more sophisticated strategy is to make use of cost and interaction information in the plan graph when selecting actions for the relaxed plan. In particular, to achieve a particular subgoal at a level l, the relaxed plan construction chooses the action that minimizes the cost of achieving the goal at level l. The sum of the costs of the actions in the relaxed plan π provides an estimate of cost for the goal. It is defined as:

$$h^I_{rp} = cost(\pi) = \sum_{a_i \in \pi} Cost_{a_i} \quad (6)$$

Like the heuristic h^I, the heuristic h^I_{rp} is non-admissible.

Accuracy Evaluation

The previous section describes two inadmissible heuristics based on a cost-plan graph with interactions. To evaluate the accuracy of these heuristics, we compare *estimated cost* against the *optimal cost* for a suite of problems. The optimal cost of each problem is computed using the optimal planner HSP^*_f (Haslum, 2008), which was allowed to run for an unlimited amount of time. In addition to the h^I and h^I_{rp} heuristics, the evaluation is done for h^{add} and h^{add}_{rp}, which are the versions without interaction. In these cases, the cost-plan graph is built using traditional cost propagation. Likewise, the evaluation is done for the set-additive heuristic h^s_a. In addition, we did the evaluation using the satisficing planners LPG (Gerevini, Saetti, and Serina, 2003), SGPlan_6 (Chen, W., and Chih-Wei, 2006), and MetricFF (Hoffmann, 2003). LPG is run using LPG-speed (LPG_s) that computes the first solution, and LPG-quality (LPG_q) that computes the best solution. MetricFF is run under its three different approaches that perform cost minimization using weighted A^* (MetricFF_3), A^*_ε (MetricFF_4), and enforced hill-climbing then A^*_ε (MetricFF_5). The experiments were conducted on an Intel Xeon CPU E5-1650 processor running at 3.20GHz with 32 GB of RAM in a time limit of 1800s.

Table 1 shows the results of this evaluation on 8 planning domains with 15 problems each. For each approach in each domain, each column shows the following measures:

- *r* is the ratio of the *estimated cost* to the *optimal cost* per problem.
- *M* is the mean of the ratio among the *solved* problems.
- σ is the standard deviation of the ratio among the *solved* problems.

The symbol "-" means the approach does not solve the problem within the time limit. Bold values symbolized the closer and lower variance cost estimate. In the Blocksword and Elevator domains, h^I computes cost estimates that are considerably closer to the optimal and more consistent (lower variance) than the actual costs computed by the satisficing planners LPG_s, LPG_q, SGPlan_6, and MetricFF. In the Logistics domain, h^I computes cost estimates as accurate as MetricFF_3, which computes the closest and most consistent solution among the evaluated planners. In the rest of the domains, h^I generates better estimates than h^{add} and h^{add}_{rp}, but not as good as h^s_a and the better satisficing planners. For the h^I_{rp} heuristic, in the Intrusion and Kitchen domains, it computes cost estimates equal to the optimal cost. In the Blocks, Campus, Floortile, and Logistics domains, h^I_{rp} computes cost estimates that are better than the costs computed by some of the satisficing planners. In the Elevator domain, it computes poor cost estimates, but in Pegsol it does slightly better than h^I.

Overall, h^I computes cost estimates with lower variance and closer to the optimal cost than cost estimates computed by h^{add} in all the domains except Logistics. h^{add}_{rp} and h^I_{rp} have similar behaviors. Our hypothesis is that while constructing the cost-relaxed plan, the algorithm only considers the actions that minimize the cost, but not the interactions between/among them. Those selected actions might be the same as the ones where the interaction during cost propagation is not considered. As a result, using interaction information in relaxed plan extraction might give a more accurate estimate of cost.

h^I Heuristics in Planning

As a result of the increased accuracy and stability of the above-described h^I heuristics, it is natural to try to use them for planning purposes. The MetricFF planner (Hoffmann, 2003) was modified to incorporate these heuristics. The search strategy remains the same. For purposes of this test, the A^*_ε strategy was chosen. The only difference is the successors evaluation of the current state, which is based on the cost estimate computed by the h^I or h^I_{rp} heuristic. The best successor is the one with the lowest cost. Four variations of the MetricFF planner are compared:

- MetricFF^I: h^I as heuristic function (equation (5)), and cost propagation through the plan graph considering interaction information.
- MetricFF^{add}: h^{add} as heuristic function, and cost propagation through the plan graph not considering interaction information.
- MetricFF^I_{rp}: h^I_{rp} as heuristic function (equation (6)), and cost propagation through the plan graph considering interaction information.
- $\text{MetricFF}^{add}_{rp}$: h^I_{rp} as heuristic function, and cost propagation through the plan graph not considering interaction information.

We have tested an additional strategy, namely MetricFF^I_k, which uses the h^I heuristic function and cost propagation through the plan graph considering interaction information in only the first k levels of the search process, and then h^{add} heuristic for the rest of the search. Because the h^I heuristic is expensive to compute, and heuristics tend to be less accurate earlier in the search, we wanted to find out if the h^I heuristic could benefit the search process, but limit computation by only using it for a limited number of levels of the search.

Tables 2 and 3 show the results of an accuracy evaluation on the same 8 planning domains used in the previous section. For each approach, each row shows an average of M and σ among all the domains. The MetricFF^I_k planner has been tested with $k = 1$ up to 4. In general, the 8 planner variations reach solutions very close to the optimal. As we expected, MetricFF^I and MetricFF^I_{rp} solution qualities are slightly better than the ones generated by MetricFF^{add} and $\text{MetricFF}^{add}_{rp}$. For MetricFF^I_k, when $k = 2$ or $k = 4$ the technique generates closer and more consistent cost estimates than for any other k value tested and any MetricFF variation. However, for $k = 2$ the technique is faster than for $k = 4$. For computational time, Figure 2 shows a scatter plot, where each dot in the plot represents the relationship between $\text{MetricFF}^{add}_{rp}$ and MetricFF^I, and between $\text{MetricFF}^{add}_{rp}$ and MetricFF^I_2 for each tested problem. (We chose to compare these three planner since they show better performance in terms of accuracy.) In general, MetricFF^I and MetricFF^I_2 are

Table 1: Accuracy evaluation among different planning and heuristic techniques.

Domain	Approach	r															M	σ
		p01	p02	p03	p04	p05	p06	p07	p08	p09	p10	p11	p12	p13	p14	p15		
Blocks	LPG_s	1.33	1.18	1	1.21	1.08	1	1	1.1	1.15	1	1	1	1.16	1.34	1.53	1.14	0.155
	LPG_q	1	1	1	1.21	1.25	1	1	1	1	1	1	1	1	1.13	1	1.039	0.081
	SGPlan	1	1	1	1	1	1	1	1	1	1	1	1	2.33	1	2.68	1.201	0.517
	MetricFF_3	1	1	1	1	1.25	1	1	1	1	1	1	1	1	1.26	1.18	1.046	0.094
	MetricFF_4	1.08	1	1	1.21	1.37	1	1	1	1.15	1.8	1	1	1.16	1.39	3.5	1.311	0.623
	MetricFF_5	1.41	1.75	1.35	1.21	1.45	1.8	1.75	1.1	1.15	1.8	1.37	1.37	1.16	1.47	1.34	1.435	0.231
	h^s_a	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
	h^I	1	1.12	1	1.1	1.2	1	1	0.75	1	1	1	1	1	1.04	1.15	**1.025**	**0.099**
	h^{add}	1.16	1.37	1.35	1.21	1.2	1	1	1.7	0.7	1	1.37	1.37	0.66	1.08	1.15	1.158	0.259
	h^I_{rp}	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
	h^{add}_{rp}	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
Campus	LPG_s	1.35	1.1	1.43	1.12	1	1.35	1.18	1.07	1.28	1.25	1.07	1.03	1.21	1.14	1.25	1.193	0.124
	LPG_q	1	1.07	1.12	1	1.12	1.07	1.12	1.07	1.28	1	1.18	1.03	1	1.03	1.31	1.096	0.096
	SGPlan	1	1	1	1	1	1.14	1	1	1	1	1	1	1	1	1	1.009	0.035
	MetricFF_3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	MetricFF_4	1.85	1.25	1.37	1.37	1.37	1.71	1.56	1.33	1.85	1.43	1.33	1.07	1.14	1.14	1.37	1.413	0.233
	MetricFF_5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	h^s_a	0.85	0.92	0.87	0.81	0.87	0.85	0.75	1	0.85	0.87	1	0.96	0.92	0.92	0.81	0.888	0.068
	h^I	1	0.89	1	1	1	1	1.12	0.92	1	1	0.92	0.89	0.96	0.96	1	0.979	0.055
	h^{add}	2.5	2.67	2.93	2.81	2.68	2.64	2.68	2.81	2.5	2.93	2.81	2.71	2.53	2.53	2.81	2.707	0.141
	h^I_{rp}	1	0.92	1	1	0.93	1	0.87	1	1	1	1	1.1	0.92	0.92	1	0.98	0.051
	h^{add}_{rp}	0.85	0.92	0.87	0.81	1.06	0.85	0.75	1	0.85	0.87	1	1.07	0.92	0.92	0.81	0.907	0.09
Elevator	LPG_s	1.6	1.4	2.11	1.28	1.09	1.33	-	1.53	1	1.26	2.33	2.638	1.94	2.52	2.88	1.781	0.595
	LPG_q	1.9	1.8	2.11	1.28	1	1.08	-	1.06	1	1.36	1.8	1.42	1.94	2.19	2.76	1.624	0.514
	SGPlan	1.2	1	1.33	1.28	1.09	1.08	-	1.06	1.2	1.21	1.46	1.47	1.15	2.04	2.05	1.333	0.323
	MetricFF_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	MetricFF_4	3.2	3.6	2	2.71	1.9	3.75	-	4.2	1.1	4.21	-	-	-	-	-	2.964	1.04
	MetricFF_5	2	1.8	1.66	1.28	1.27	1.08	-	1.73	1.2	1.31	2.33	1.94	1.73	2.52	1.7	1.686	0.411
	h^s_a	0.8	0.8	1.11	0.85	0.9	0.83	-	1	1	0.78	1.06	0.84	0.94	0.95	1	0.922	0.1
	h^I	0.7	1.8	1	1.85	0.72	0.75	-	1	1.3	0.73	0.66	0.57	0.94	0.85	0.94	**0.990**	**0.384**
	h^{add}	2.2	2.8	2.55	3.14	2.45	2.16	0	2.13	2	2.26	5.2	3.52	4	4.71	4.35	3.107	1.031
	h^I_{rp}	1.7	2.2	1.77	1.71	1.72	1.5	-	1.46	1.6	1.42	1.73	1.68	1.68	1.57	1.76	1.681	0.18
	h^{add}_{rp}	1.7	2.4	1.77	1.14	1.72	1.5	0	1.46	1.6	1.42	1.73	1.68	1.68	1.57	1.7	1.651	0.263
Floortile	LPG_s	1	1.36	1.32	1.53	1.76	4.94	2	4.55	5.46	7.65	7.79	5.2	4.74	4.32	4.16	3.855	2.177
	LPG_q	1	1	2.84	1	1.58	1.5	1.16	1.4	1.55	1.44	1.18	1.64	-	1.42	1.13	1.421	0.45
	SGPlan	1	1.36	1.8	2.06	1.52	1.944	1.54	1.5	1.72	1.65	1.37	1.49	1.43	1.78	1.47	1.58	0.252
	MetricFF_3	1	1	1	1.4	1	1	1.29	-	1.09	-	-	-	-	-	-	1.097	0.148
	MetricFF_4	1	1	3.16	1.93	3.08	-	-	-	-	-	-	-	-	-	-	2.036	0.951
	MetricFF_5	1	1.72	2.4	1.53	1.64	-	-	-	-	-	-	-	-	-	-	1.661	0.448
	h^s_a	1.2	1.09	0.92	1	0.88	0.86	0.75	1.01	0.8	1.03	0.86	0.83	1.15	0.91	0.94	**0.951**	**0.125**
	h^I	1	1	1.12	0.86	1	1	0.83	0.33	0.66	0.16	0.29	0.09	0.12	0.48	0.25	0.614	0.366
	h^{add}	1.2	1.27	1.04	1.2	1.02	1.02	1	1.28	1.04	1.66	1.18	1.41	1.85	1.19	1.27	1.246	0.233
	h^I_{rp}	1	1.36	1.16	1.13	1.05	0.88	0.83	1.03	0.86	0.86	0.76	0.75	0.81	0.92	0.88	**0.954**	**0.164**
	h^{add}_{rp}	1	1.36	1.16	1.13	1.05	0.88	0.83	1.03	0.86	0.83	0.76	0.75	0.81	0.92	0.88	**0.954**	**0.164**
Intrusion	others	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	h^I	1	1	1	1	1	1	1	1	0.9	1	1	1	1	1	1	0.993	0.024
	h^{add}	1	1.33	1.25	1.33	1.25	1.22	1.33	1.25	1.5	1.28	1.25	1.25	1.25	1.25	1.28	1.269	0.097
Kitchen	others	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	LPG_s	1	1	1	1	1	1	1	1	1.3	1.3	1	1	1	1	1.3	1.06	0.12
	LPG_q	1.06	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.004	0.016
	SGPlan	1.06	1.06	1.06	1.06	1.13	1.13	1.13	1.13	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.192	0.103
	MetricFF_5	1.04	1.04	1.04	1.04	1	1	1	1	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.104	0.09
	h^I	0.97	0.97	0.97	0.97	1	1	1	1	1	1	1	1	1	1	1	0.994	0.009
	h^I_{rp}	1.06	1.06	1.06	1.06	1	1	1	1	1	1	1	1	1	1	1	1.017	0.028
Logistic	LPG_s	1.14	1.25	1.41	1.58	1.28	1.36	1.08	1.2	1	1.16	2.25	1.77	1	1.4	1.07	1.333	0.321
	LPG_q	1.21	1.08	1.08	1	1	1.27	1.41	1	1.08	1.16	1.5	1.22	1	1	1.3	1.156	0.156
	SGPlan	1	1	1.16	1.16	1	1.18	1.16	1.2	1	1	1	1.22	1.16	1	1	1.084	0.091
	MetricFF_3	1.07	1	1	1.16	1.07	1	1	1	1	1	1	1	1	1	1	**1.02**	**0.045**
	MetricFF_4	1.35	1.08	2.91	1.83	1.35	1.63	1.08	1	1.75	1.08	1.08	1	1.08	1.3	2.23	1.453	0.525
	MetricFF_5	1.14	1.08	1.08	1.08	1.07	1.18	1	1	1.08	1.08	1.08	1.44	1.08	1.6	1.23	1.15	0.158
	h^s_a	0.71	0.75	1	1	0.71	0.81	1	1	1	1	0.75	1	1	0.7	0.84	0.886	0.126
	h^I	0.92	1.08	1	1.08	0.92	0.72	1	1	1	1	1.08	0.88	1	1.2	0.76	**0.979**	**0.116**
	h^{add}	0.71	1.16	1	1	0.71	0.81	1	1	1	1	1	1	1	1	0.84	0.95	0.118
	h^I_{rp}	1	1.16	1	1.16	1	1	1	1	1	1	1.16	1	1	1.2	0.84	1.036	0.092
	h^{add}_{rp}	0.71	0.75	1	1	0.71	0.81	1	1	1	1	0.75	1	1	0.7	0.84	0.886	0.126
Pegsol	LPG_s	1	1	1	1.5	1.25	2	1.66	-	1.26	2.05	-	1.29	-	-	1.79	1.438	0.374
	LPG_q	1	1	1	1	1	1.25	1	1.05	1.26	1.55	-	1.04	-	-	-	**1.106**	**0.171**
	SGPlan	1	1	1	1	1.25	1.5	1.33	-	1.2	-	1.14	-	-	-	1.5	1.192	0.19
	MetricFF_3	1	1	1	1	1.25	1.75	1	1.83	1.4	1.83	1.28	1	1.33	1.57	1.5	1.317	0.309
	MetricFF_4	1	1	1	1.5	1.25	1.75	1.33	1.83	1.6	1.66	1.28	1.25	1.33	1.42	1.37	1.373	0.254
	MetricFF_5	1	1	1	1.5	1.25	1.75	2	1.83	1.6	1.66	1.28	1.25	1.33	1.42	1.37	1.418	0.297
	h^s_a	1	0.8	1.25	1.25	1	2	1.33	1.5	1.8	1.16	1.28	1.12	1	1.28	1.25	1.269	0.299
	h^I	1.5	0.53	0.33	4.41	0.33	0.41	6.44	1.66	0.46	0.22	0.76	0.29	2.81	1.09	0.29	1.439	1.747
	h^{add}	7.5	1.2	1.75	2.25	2.75	4	7.33	1.83	3	2.5	2	2.37	1.66	2.14	2.25	2.97	1.853
	h^I_{rp}	2	0.86	1.33	1.25	1.25	1.25	2.33	1.16	1.86	1.22	1.33	0.87	0.88	1.33	1.45	1.361	0.401
	h^{add}_{rp}	2.5	1	1.25	1.25	1.75	1.75	3	1.16	2	1.33	1.14	0.87	0.88	1.42	1.37	1.514	0.58

slower than $\text{MetricFF}^{add}_{rp}$. However, there are several problems where $\text{MetricFF}^{add}_{rp}$ does not find the solution within the given time, unlike MetricFF^I and MetricFF^I_2. It is also noticeable that the use of the h^I heuristic in only the first two

Table 2: Accuracy evaluation among different MetricFF variations based on h^I, h^{add}, h^I_{rp}, and h^{add}_{rp} heuristics.

Approach	MetricFF^I	MetricFF^{add}	MetricFF$^I_{rp}$	MetricFF$^{add}_{rp}$
M	1.006	1.013	1.088	1.015
σ	0.019	0.034	0.145	0.02

levels of the search process benefits the performance. It generates equal quality results and is a bit faster than MetricFF^I.

Table 3: Accuracy evaluation among different k values.

Approach	MetricFF$^I_1$	MetricFF$^I_2$	MetricFF$^I_3$	MetricFF$^I_4$
M	1.015	1.005	1.006	1.005
σ	0.026	0.017	0.019	0.016

In general, HSP$^*_f$ takes less time to solve a problem than any MetricFF variation we evaluated. One of the reasons might be the difference in the search algorithm, IDA* versus A$^*_\epsilon$. We have not done comparisons between heuristics in HSP$^*_f$ because the code for HSP$^*_f$ is not well documented and we have not been successful at incorporating different heuristics.

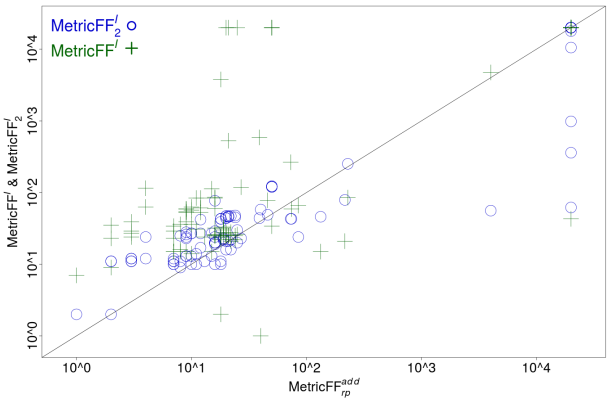


Figure 2: Time comparison among MetricFF$^{add}_{rp}$, MetricFF^I, and MetricFF$^I_2$.

While the computational overhead of full h^I is high and does not usually pay off during the actual search process, this heuristic is extremely useful for goal recognition, which we discuss in the next section.

h^I Heuristics in Goal Recognition

Ramírez (2010; 2012), defines a goal recognition problem to be a tuple $T = \langle P, \mathcal{G}, O, Pr \rangle$ where P is a planning domain and initial conditions, $\mathcal{G}$ is a set of possible goals or hypotheses, O is the observed action sequence $O = o_1, ..., o_n$, and Pr is the prior probability distribution over the goals in $\mathcal{G}$. The solution to a plan recognition problem is a probability distribution over the set of goals $G \in \mathcal{G}$ giving the relative likelihood of each goal. These posterior goal probabilities $P(G|O)$ can be characterized using Bayes Rule as:

$$Pr(G|O) = \alpha \, Pr(O|G) \, Pr(G) \tag{7}$$

where α is a normalizing constant, $Pr(G)$ is the prior distribution over $G \in \mathcal{G}$, and $Pr(O|G)$ is the likelihood of observing O when the goal is G. Ramírez goes on to characterize the likelihood $Pr(O|G)$ in terms of cost differences for achieving G under two conditions: complying with the observations O, and not complying with the observations O. More precisely, Ramírez characterizes the likelihood, $Pr(O|G)$, in terms of a Boltzman distribution:

$$Pr(O|G) = \frac{e^{[-\beta \, \Delta(G,O)]}}{1 + e^{[-\beta \, \Delta(G,O)]}} \tag{8}$$

where β is a positive constant and $\Delta(G, O)$ is the cost difference between achieving the goal with and without the observations:

$$\Delta(G, O) = Cost(G|O) - Cost(G|\overline{O}) \tag{9}$$

Putting equations (7) and (8) together yields:

$$Pr(G|O) = \alpha \frac{e^{[-\beta \, \Delta(G,O)]}}{1 + e^{[-\beta \, \Delta(G,O)]}} \, Pr(G) \tag{10}$$

By computing $\Delta(G, O)$ for each possible goal, equation 10 can be used to compute a probability distribution over those goals. The two costs necessary to compute Δ can be found by optimally solving the two planning problems $G|O$ and $G|\overline{O}$. Ramírez shows how the constraints O and $\overline{O}$ can be compiled into the goals, conditions and effects of the planning problem so that a standard planner can be used to find plans for $G|O$ and $G|\overline{O}$.

A significant drawback to the Ramírez approach is the computational cost of calling a planner twice for each possible goal. This makes the approach impractical for real-time goal recognition, such as for a robot observing a human, and trying to assist or avoid conflicts. Using h^I we show how to quickly infer a probability distribution over the possible goals using the framework of Ramirez. To compute $Cost(G|O)$, the cost-plan graph is pruned considering the sequence of observed actions. Consequently, cost estimates for goals with and without the observations are quickly computed, and a probability distribution over those goals is inferred.

Incremental Plan Recognition

Jigui and Minghao (2007) developed a framework for plan recognition that narrows the set of possible goals by incrementally pruning a plan graph as actions are observed. The approach consists of building a plan graph to determine which actions and which propositions are true (1), false (-1), or unknown (0) given the observations. For level zero, it is assumed the initial state is true: each proposition has value 1. In addition, when an action is observed at a level it gets value 1. The process incrementally builds a plan graph

and updates it level by level. The values of propositions and actions are updated according to the following rules:

1. An action in the plan graph gets value -1 when any of its preconditions or any of its effects is -1.
2. An action in the plan graph gets value 1 when it is the sole producer of an effect that has value 1, `noop` included.
3. A proposition in the plan graph gets value -1 when all of its consumers or all of its producers are -1, `noop` included.
4. A proposition in the plan graph gets value 1 when any of its consumers or any of its producers is 1, `noop` included.

The process results in a plan graph where each proposition and each action is labeled as 1, -1, or 0. Those propositions and actions identified as -1 can be ignored for plan recognition purposes, meaning that these are pruned from the resulting plan graph.

This technique assumes we know the time at which each action has been observed. To relax this assumption, we developed a modified version of this pruning technique. Like Ramírez and Geffner (2010), we assume that the sequence of actions is sequential. Initially, an *earliest time step* (*ets*) i is assigned to each action o in the observed sequence. The *ets* is given by the order of each action in the observed sequence. That is, given $(o_0, o_1, ..., o_i)$, the *ets* for each action is: *ets*(o_0)=0, *ets*(o_1)=1, *ets*(o_2)=2, etc. When the pruning process starts, we establish that an observed action o may be observed at the assigned level i if all its preconditions are true (value 1) and/or unknown (value 0), and they are not mutually exclusive at level $i-1$. Otherwise, the action cannot be executed at that level, which results in an update of the *ets* of each remaining action in the observed sequence. The result of this updating is that each observed action is assumed to occur at the earliest possible time consistent with both the observation sequence and the constraints found in constructing the plan graph, using the interaction information. To illustrate this propagation and pruning technique, consider a simple problem with three operators:

$$\begin{aligned} A &: y \rightarrow z \\ B &: y \rightarrow, \neg y, t \\ C &: t \rightarrow k, \neg t \end{aligned} \tag{11}$$

Suppose the sequence of observed actions is A and C, with initial *ets* 0 and 1 respectively. As a result of the propagation, z must be true (have value 1) at level 1 because A was observed. Since A and B are mutex, B and its effects t and $\neg y$ are false (have value -1) at level 0. C is initially assumed to be at level 1, but this cannot be the case because its precondition t is false at level 0. Therefore, the *ets* for C is updated to 2. At level 2, k and $\neg t$ must be true because C was observed. (This results in t being true at level 1.) Since A and C, and B and C are mutex, A, B, $\neg y$, and t are not possible (have value -1) at level 2. B is unknown (has value 0) at level 1 since there is not enough information to determine whether it is true or false. The proposition y is true at level 0 since the initial state is assumed to be true, and is unknown at level 1 because there is not enough information to determine whether it is true or false. However, it is false at level 2 due to the mutex relation between C and noop-y. Proposition z is true at each level since there are no operators in the domain that delete it.

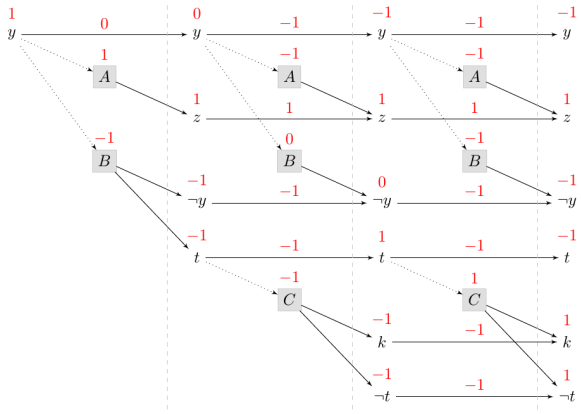


Figure 3: A plan graph with status values of propositions and actions

Fast Goal Recognition

The union of the plan graph cost estimation and the observation pruning techniques results in a heuristic approach that allows fast estimation of cost differences $\Delta(G, O)$, giving us probability estimates for the possible goals $G \in \mathcal{G}$. The steps are:

1. Build a plan graph for the problem P (domain plus initial conditions) and propagate cost and interaction information through this plan graph according to the technique early described.
2. For each (possibly conjunctive) goal $G \in \mathcal{G}$ estimate the $Cost(G)$ from the plan graph using equation (5).
3. Prune the plan graph, based on the observed actions O, using the technique early described.
4. Compute new cost and interaction estimates for this pruned plan graph, considering only those propositions and actions labeled 0, or 1.
5. For each (possibly conjunctive) goal $G \in \mathcal{G}$ estimate the $Cost(G|O)$ from the cost and interaction estimates in the pruned plan graph, again using equation (5). The pruned cost-plan graph may discard propositions or/and actions in the cost-plan graph necessary to reach the goal. This constraint provides a way to discriminate possible goals. However, it may imply that 1) the real goal is discarded, 2) the calculated costs are less accurate. Therefore, computation of $Cost(G|O)$ has been developed under two strategies:
   (a) $Cost(G|O)$ is computed using the pruned cost-plan graph.
   (b) $Cost(G|O)$ is computed after the pruned cost-plan graph is expanded to quiescence again. This will reintroduce any pruned goals that are still possible given the observations.
6. For each goal $G \in \mathcal{G}$, compute $\Delta(G, O)$, and using equation (10) compute the probability $Pr(G|O)$ for the goal given the observations.

To illustrate this computation, consider again actions A, B, and C from equation (11), and the plan graph shown in Figure 3. Suppose that A, B, and C have costs 2, 1, and 3 respectively, and that the possible goals are $g_1 = \{z, k\}$ and $g_2 = \{z, t\}$. Propagating cost and interaction information through the plan graph, we get $Cost(t) = 1$, $Cost(z) = 2$, $Cost(k) = 4$, and interaction values $I(k, t) = \infty$ and $I(k, z) = 0$ at level 3. Now consider the hypothesis $g_1 = \{z, k\}$; in order to compute $Cost(k \wedge z)$, we use the cost and interaction information propagated through the plan graph. In order to compute $Cost(k \wedge z|O)$, the cost and interaction information is propagated again only in those actions with status 1 and 0. In our example, these costs are:

$$Cost(k \wedge z) \approx Cost(z) + Cost(k) + I(k, z) = 2 + 4 + 0 = 6$$

$$Cost(k \wedge z|O) \approx Cost(k) + Cost(z) + I(k, z) = 4 + 2 - 1 = 5$$

Thus, the cost difference is:

$$\Delta(g_1, O) = Cost(g_1|O) - Cost(g_1) = 5 - 6 = -1$$

As a result:

$$Pr(O|g_1) = \frac{e^{-(-1)}}{1 + e^{-(-1)}} = 0.73$$

For the hypothesis $g_2 = \{z, t\}$, the plan graph dismisses this hypothesis as a solution because once the plan graph is pruned, propositions *t* and *z* are labeled as -1. Therefore:

$$Cost(k \wedge t|O) \approx Cost(k) + Cost(t) + I(k, t) = \infty$$

So:

$$Pr(O|g_2) = \frac{e^{-\infty}}{1 + e^{-\infty}} = \frac{0}{1} = 0$$

If we expand the pruned cost-plan graph until quiescence again, the solution is still the same because A and B are permanently mutually exclusive.

Assuming uniform priors, $Pr(G)$, after normalizing the probabilities, we get that $Pr(g_1|O) = 1$ and $Pr(g_2|O) = 0$, so the goal g_1 is certain in this simple example, given the observations of actions A and C.

Experimental Results

We conducted an experimental evaluation on planning domains used by Ramírez: BlocksWord, Intrusion, Kitchen, and Logistics. Each domain has 15 problems. The hypotheses set and actual goal for each problem were chosen at random with the priors on the goal sets assumed to be uniform. For each problem in each of the domains, we ran the LAMA planner (Richter and Westphal, 2010) to solve the problem for the actual goal. The set of observed actions for each recognition problem was taken to be a subset of this plan solution, ranging from 100% of the actions, down to 10% of the actions.

Ramírez evaluates his technique using an optimal planner HSP$^*_f$, and LAMA, a satisficing planner that is used in two modes: as a greedy planner that stops when it finds the first plan (LAMA$_G$), and as a planner that returns the best plan found in a given time limit (LAMA). For purposes of this test, Ramírez technique is also evaluated using the heuristic h^s_a, which was used in (Ramírez and Geffner, 2009). Like our technique, this requires no search since the cost is given by a heuristic function. We compare our goal recognition technique, GR, against Ramírez's technique for those three planners and h^s_a, on the aforementioned domains, using a range of time limits from 5 seconds up to 1800 seconds. We present three variations of our technique, with and without extension of the plan graph after pruning:

- GR$_I$: cost propagation in a plan graph considers interaction information.
- GR$_I$E: same as above, but the pruned cost-plan graph is expanded until quiescence.
- GR$_{add}$: traditional cost propagation in a plan graph.

Table 4 summarizes the results. For each planner, each column shows average performance over the 15 problems in each domain. The first row in the table represents the optimal solution where HSP$^*_f$ (HSP$^*_f$u) was allowed to run for an unlimited amount of time. The other rows represent different measures of quality and performance:

- T shows the average time in seconds taken for solving the problems.
- Q shows the fraction of times the actual goal was among the goals found to be the most likely.
- S shows the *spread*, that is, the average number of goals in $\mathcal{G}$ that were found to be the most likely.
- Q_{20} and Q_{50} show the fraction of times the actual goal is in the top 20% and top 50% of the ranked goals. Although Q might be less than 1 for some problem, Q_{20} or Q_{50} might be 1, indicating that the actual goal was *close* to the top.
- d is the mean distance between the probability scores produced for all the goal candidates, and the probability scores produced by gHSP$^*_f$u. More precisely, if the set of possible goals is $\{G_1, ..., G_n\}$, a method produces probabilities $\{e_1, ..., e_n\}$ for those goals, and gHSP$^*_f$u produces $\{p_1, ..., p_n\}$, d is defined as:

$$d = 1/n \sum_{i=1}^{n} |e_i - p_i| \tag{12}$$

The use of an optimal planner like HSP$^*_f$u is generally impractical for real-time goal recognition on any non-trivial domain. Surprisingly, LAMA does not perform any better on the harder domains, and the solution quality is uneven. Greedy LAMA is much faster, but still no faster than HSP$^*_f$u in the blocks world domain. The h^s_a heuristic (Ramírez and Geffner, 2009) for approximating costs is quite fast, but the cost estimates are not very accurate, leading to poor quality results for goal recognition. The GR$_I$ heuristic is also quite fast, but yields much better results. On the harder domains, it is two orders of magnitude faster than HSP$^*_f$u, LAMA, or Greedy LAMA, and yields results of comparable quality to both LAMA and Greedy LAMA for the higher observation percentages.

Conclusions and Future Work

This paper presents a heuristic estimator h^I based on a cost-plan graph and interaction information to compute more accurate cost estimates. This heuristic provides cost estimates

Table 4: Goal recognition with random observations

Domain		Blocks					Intrusion					Kitchen					Logistics				
Approach	%O	100	70	50	30	10	100	70	50	30	10	100	70	50	30	10	100	70	50	30	10
$HSP^*_{\hat{f}}u$	T	558.08	419.45	379.81	357.94	357.94	447.41	281.12	151.37	3.58	3.55	480.51	171.08	49.62	37.93	37.92	36.26	32.46	14.08	7.04	7.04
	Q	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.66	0.8	0.8
	S	1.06	1.13	4.06	11.46	11.46	1	1	1.06	4.46	4.6	1	1	1.33	1.4	1.4	1	1.13	2.26	3.6	3.6
LAMA	T	1603.24	1522.96	1260.6	1077.62	1082.15	92.85	89.01	64.97	17.57	17.48	26.33	26.2	20.45	20.3	20.3	45.17	41.71	26.53	11.38	11.39
	Q	0.33	0.8	0.8	0.93	1	0.93	1	1	1	1	0.73	1	1	1	1	1	0.93	0.66	0.8	0.8
	S	1	1.13	3.86	10.4	10.93	0.93	1	1.06	4.46	4.6	0.73	1	1.33	1.4	1.4	1	1.13	2.26	3.6	3.6
	Q_{20}	1	1	1	1	1	0.93	1	1	1	1	0.73	1	1	1	1	1	1	0.93	1	1
	Q_{50}	1	1	1	1	1	0.93	1	1	1	1	0.73	1	1	1	1	1	1	1	1	1
	d	0.24	0.316	0.192	0.048	0.068	1.739	1.12	0.095	7×10^{-6}	7×10^{-6}	0.358	3×10^{-3}	0.016	0.012	0.012	0.019	0.673	1.051	0.956	0.956
$LAMA_G$	T	849.08	840.76	814.95	803.04	809.01	3.32	2.63	2.21	2.08	2.08	0.42	0.36	0.33	0.32	0.32	5.47	4.95	4.5	4.33	4.36
	Q	0.93	0.8	0.73	0.66	0.46	0	0.4	1	1	1	0.73	1	1	1	1	1	0.8	0.4	0.46	0.46
	S	1	1.2	3	6.2	4.2	0	0.4	1.13	4.46	4.6	0.73	1	1.33	1.4	1.4	1	1.2	1.8	2.93	2.93
	Q_{20}	0.93	1	1	1	1	0	0.4	1	1	1	0.73	1	1	1	1	1	1	0.66	0.6	0.6
	Q_{50}	0.93	1	1	1	1	0	0.4	1	1	1	0.73	1	1	1	1	1	1	0.93	0.86	0.86
	d	0.068	0.34	0.404	0.322	0.341	1.86	1.12	0.108	7×10^{-6}	7×10^{-6}	0.358	3×10^{-3}	0.016	0.012	0.012	0.019	0.675	1.179	1.063	1.063
h^*_a	T	1.04	0.89	0.81	0.81	0.8	0.7	0.46	0.39	0.35	0.36	0.066	0.049	0.044	0.045	0.046	0.61	0.55	0.51	0.51	0.51
	Q	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.13	0.06	0.13	0.06	0.06
	S	20.26	20.26	20.26	20.26	20.26	16.66	16.66	16.66	16.66	16.66	3	3	3	3	3	2.8	1.86	1.93	1	1
	Q_{20}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.93	0.86	0.86	0.86
	Q_{50}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.93	0.86	0.86	0.86
	d	0.092	0.087	0.062	0.035	0.035	0.123	0.124	0.119	0.075	0.073	0.443	0.436	0.284	0.226	0.226	0.123	0.117	0.099	0.074	0.074
GR_I	T	1.007	1.029	1.265	1.452	1.452	0.885	0.493	0.212	0.209	0.21	0.261	0.195	0.140	0.135	0.135	0.886	1.015	1.19	1.266	1.271
	Q	1	0.66	0.4	0.13	0.13	1	1	0.93	0.93	0.93	1	1	1	1	1	1	0.86	0.53	0.6	0.6
	S	1.06	0.8	1.06	1.73	1.73	1	1	1	4.4	4.53	1	1	1.2	1.26	1.26	1	1.26	1.6	2.46	2.46
	Q_{20}	1	0.66	0.53	0.46	0.46	1	1	1	0.93	0.93	1	1	1	1	1	1	0.93	0.66	0.73	0.73
	Q_{50}	1	0.73	0.73	0.8	0.8	1	1	1	1	1	1	1	1	1	1	1	0.93	0.8	0.86	0.86
	d	0.149	0.962	0.751	0.336	0.336	3.2×10^{-4}	0.055	0.872	0.241	0.241	3.98×10^{-4}	0.036	0.107	0.192	0.192	0.264	0.786	1.163	0.718	0.718
$GR_I E$	T	9.936	8.211	4.542	3.696	3.687	1.293	1.191	0.996	0.743	0.738	0.287	0.266	0.230	0.193	0.193	7.535	4.24	3.016	2.842	2.834
	Q	0.46	0.53	0.46	0.13	0.13	1	1	0.93	0.93	0.93	1	1	1	1	1	0.86	0.66	0.66	0.6	0.6
	S	1.26	1.13	1.86	1.73	1.73	1	1	1	4.4	4.53	1	1	1.2	1.26	1.26	1.13	1.4	1.8	2.8	2.8
	Q_{20}	0.46	0.73	0.66	0.4	0.4	1	1	1	0.93	0.93	1	1	1	1	1	0.86	0.8	0.8	0.73	0.73
	Q_{50}	0.46	0.8	0.86	0.8	0.8	1	1	1	1	1	1	1	1	1	1	0.93	1	0.86	0.86	0.86
	d	1.094	1.025	0.742	0.358	0.358	3.2×10^{-4}	0.055	0.872	0.241	0.241	3.98×10^{-4}	0.036	0.107	0.192	0.192	0.387	0.943	1.107	0.771	0.771
GR_{add}	T	0.761	0.609	0.643	0.782	0.783	0.886	0.491	0.196	0.192	0.193	0.258	0.191	0.136	0.128	0.129	0.806	0.405	0.448	0.508	0.51
	Q	1	0.46	0.46	0.46	0.46	1	1	1	1	0.93	1	1	1	1	1	1	0.4	0.46	0.6	0.6
	S	1	0.53	1.2	2.06	2.06	1	1.13	1.13	4.06	3.93	1	1	1.33	1.4	1.4	1	1	2.13	2.93	2.93
	Q_{20}	1	0.46	0.6	0.6	0.6	1	1	1	1	0.93	1	1	1	1	1	1	0.46	0.6	0.66	0.66
	Q_{50}	1	0.46	0.6	0.73	0.73	1	1	1	1	1	1	1	1	1	1	1	0.53	0.73	0.8	0.8
	d	0.088	1.084	1.036	1.004	1.004	0.151	0.764	1.25	0.311	0.285	2×10^{-6}	0.038	5×10^{-6}	0.022	0.022	0.011	1.196	1.29	0.899	0.899

that are substantially closer to the optimal and more consistent. Regardless of the quality estimates generated by h^I, its use in classical planning does not pay off because of the computational overhead. However, it is very useful for goal recognition to infer probability estimates for the possible goals.

Acknowledgments

This work was funded by the JCCM project PEII-2014-015A, USRA, and NASA Ames Research Center.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *AI* 90:281—300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *JAIR* 129:5—33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI'97*.

Bryce, D., and Smith, D. E. 2006. Using interaction to compute better probability estimates in plan graphs. In *ICAPS'06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems*.

Chen, Y.; W., W. B.; and Chih-Wei, H. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *JAIR* 26:323—369.

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *AIPS'02*. Toulouse, France.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *JAIR* 20:239—290.

Haslum, P., and Geffner, H. 2000. Admissible heuristic for optimal planning. In *AIPS'00*.

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. In *IPC-08*.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291—341.

Jigui, S., and Minghao, Y. 2007. Recognizing the agent's goals incrementally: planning graph as a basis. *In Frontiers of Computer Science in China* 1(1):26—36.

Keyder, E., and Geffner, H. 2007. Heuristics for planning with action costs. In *CAEPIA'07*.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. S. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. *AI* 135(1-2):73—123.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI'09*.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI'10*.

Ramírez, M. 2012. *Plan Recognition as Planning*. Ph.D. Dissertation, Universitat Pompeu Fabra, Barcelona, Spain.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *JAIR* 39:127—177.

Red-Black Planning: A New Tractability Analysis and Heuristic Function

Daniel Gnad and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de

Abstract

Red-black planning is a recent approach to partial delete relaxation, where red variables take the relaxed semantics (accumulating their values), while black variables take the regular semantics. Practical heuristic functions can be generated from tractable sub-classes of red-black planning. Prior work has identified such sub-classes based on the black causal graph, i. e., the projection of the causal graph onto the black variables. Here, we consider cross-dependencies between black and red variables instead. We show that, if no red variable relies on black preconditions, then red-black plan generation is tractable in the size of the black state space, i. e., the product of the black variables. We employ this insight to devise a new red-black plan heuristic in which variables are painted black starting from the causal graph leaves. We evaluate this heuristic on the planning competition benchmarks. Compared to a standard delete relaxation heuristic, while the increased runtime overhead often is detrimental, in some cases the search space reduction is strong enough to result in improved performance overall.

Introduction

In classical AI planning, we have a set of finite-domain state variables, an initial state, a goal, and actions described in terms of preconditions and effects over the state variables. We need to find a sequence of actions leading from the initial state to a goal state. One prominent way of addressing this is heuristic forward state space search, and one major question in doing so is how to generate the heuristic function automatically, i. e., just from the problem description without any further human user input. We are concerned with that question here, in satisficing planning where no guarantee on plan quality needs to be provided. The most prominent class of heuristic functions for satisficing planning are *relaxed plan* heuristics (e. g. (McDermott 1999; Bonet and Geffner 2001; Hoffmann and Nebel 2001; Gerevini, Saetti, and Serina 2003; Richter and Westphal 2010)).

Relaxed plan heuristics are based on the *delete* (or *monotonic*) *relaxation*, which assumes that state variables accumulate their values, rather than switching between them. Optimal delete-relaxed planning still is **NP**-hard, but satisficing delete-relaxed planning is polynomial-time (Bylander 1994). Given a search state s, relaxed plan heuristics generate a (not necessarily optimal) delete-relaxed plan for s, resulting in an inadmissible heuristic function which tends to be very informative on many planning benchmarks (an explicit analysis has been conducted by Hoffmann (2005)).

Yet, like any heuristic, relaxed plan heuristics also have significant pitfalls. A striking example (see, e. g., (Coles et al. 2008; Nakhost, Hoffmann, and Müller 2012; Coles et al. 2013)) is "resource persistence", that is, the inability to account for the consumption of non-replenishable resources. As variables never lose any "old" values, the relaxation pretends that resources are never actually consumed. For this and related reasons, the design of heuristics that take *some* deletes into account has been an active research area from the outset (e. g. (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; van den Briel et al. 2007; Helmert and Geffner 2008; Coles et al. 2008; Keyder and Geffner 2008; Baier and Botea 2009; Keyder, Hoffmann, and Haslum 2012; Coles et al. 2013; Keyder, Hoffmann, and Haslum 2014). We herein continue the most recent approach along these lines, *red-black planning* as introduced by Katz et al. (2013b).

Red-black planning delete-relaxes only a subset of the state variables, called "red", which accumulate their values; the remaining variables, called "black", retain the regular value-switching semantics. The idea is to obtain an inadmissible yet informative heuristic in a manner similar to relaxed plan heuristics, i. e. by generating some (not necessarily optimal) red-black plan for any given search state s. For this to make sense, such red-black plan generation must be sufficiently fast. Therefore, after introducing the red-black planning framework, Katz et al. embarked on a line of work generating red-black plan heuristics based on tractable fragments. These are characterized by properties of the projection of the causal graph – a standard structure capturing state variable dependencies – onto the black variables (Katz, Hoffmann, and Domshlak 2013a; Katz and Hoffmann 2013; Domshlak, Hoffmann, and Katz 2015). *Cross-dependencies between black and red variables were not considered at all yet.* We fill that gap, approaching "from the other side" in that we analyze *only* such cross-dependencies. We ignore the structure inside the black part, assuming that there is a single black variable only; in practice, that "single variable" will correspond to the cross-product of the black variables.

Distinguishing between (i) black-precondition-to-red-effect, (ii) red-precondition-to-black-effect, and (iii) mixed-red-black-effect dependencies, and assuming there is a sin-

gle black variable, we establish that (i) alone governs the borderline between **P** and **NP**: If we allow type (i) dependencies, deciding red-black plan existence is **NP**-complete, and if we disallow them, red-black plan generation is polynomial-time. Katz et al. also considered the single-black-variable case. Our hardness result strengthens theirs in that it shows only type (i) dependencies are needed. Our tractability result is a major step forward in that *it allows to scale the size of the black variable*, in contrast to Katz et al.'s algorithm whose runtime is exponential in that parameter. Hence, in contrast to Katz et al.'s algorithm, ours is practical. It leads us to a new red-black plan heuristic, whose painting strategy draws a "horizontal line" through the causal graph viewed as a DAG of strongly connected components (SCC), with the roots at the top and the leaves at the bottom. The part above the line gets painted red, the part below the line gets painted black, so type (i) dependencies are avoided.

Note that, by design, the black variables must be "close to the causal graph leaves". This is in contrast with Katz et al.'s red-black plan heuristics, which attempt to paint black the variables "close to the causal graph root", to account for the to-and-fro of these variables when servicing other variables (e. g., a truck moving around to service packages). Indeed, if the black variables are causal graph leaves, then provably no information is gained over a standard relaxed plan heuristic (Katz, Hoffmann, and Domshlak 2013b). However, in our new heuristic we paint black *leaf SCCs*, as opposed to *leaf variables*. As we point out using an illustrative example, this can result in better heuristic estimates than a standard relaxed plan, and even than a red-black plan when painting the causal graph roots black. That said, in the International Planning Competition (IPC) benchmarks, this kind of structure seems to be rare. Our new heuristic often does not yield a search space reduction so its runtime overhead ends up being detrimental. Katz et al.'s heuristic almost universally performs better. In some cases though, our heuristic does reduce the search space dramatically relative to standard relaxed plans, resulting in improved performance.

Preliminaries

Our approach is placed in the *finite-domain representation (FDR)* framework. To save space, we introduce FDR and its delete relaxation as special cases of red-black planning. A *red-black (RB)* planning task is a tuple $\Pi = \langle V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G\rangle$. V^{B} is a set of *black state variables* and V^{R} is a set of *red state variables*, where $V^{\mathsf{B}} \cap V^{\mathsf{R}} = \emptyset$ and each $v \in V := V^{\mathsf{B}} \cup V^{\mathsf{R}}$ is associated with a finite domain $\mathcal{D}(v)$. The *initial state* I is a complete assignment to V, the *goal* G is a partial assignment to V. Each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ of partial assignments to V called *precondition* and *effect*. We often refer to (partial) assignments as sets of *facts*, i. e., variable-value pairs $v = d$. For a partial assignment p, $\mathcal{V}(p)$ denotes the subset of V instantiated by p. For $V' \subseteq \mathcal{V}(p)$, $p[V']$ denotes the value of V' in p.

A state s assigns each $v \in V$ a non-empty subset $s[v] \subseteq \mathcal{D}(v)$, where $|s[v]| = 1$ for all $v \in V^{\mathsf{B}}$. An action a is applicable in state s if $\mathsf{pre}(a)[v] \in s[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of $v \in \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}}$ to $\{\mathsf{eff}(a)[v]\}$, and changes the value of $v \in \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}}$ to $s[v] \cup \{\mathsf{eff}(a)[v]\}$. The resulting state is denoted $s[\![a]\!]$. By $s[\![\langle a_1, \ldots, a_k\rangle]\!]$ we denote the state obtained from sequential application of $a_1, \ldots, a_k$. An action sequence $\langle a_1, \ldots, a_k\rangle$ is a *plan* if $G[v] \in I[\![\langle a_1, \ldots, a_k\rangle]\!][v]$ for all $v \in \mathcal{V}(G)$.

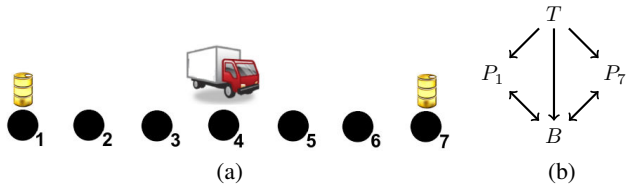
Π is a *finite-domain representation (FDR)* planning task if $V = V^{\mathsf{B}}$, and is a *monotonic finite-domain representation (MFDR)* planning task if $V = V^{\mathsf{R}}$. Optimal planning for MFDR tasks is **NP**-complete, but satisficing planning is polynomial-time. The latter can be exploited for deriving (inadmissible) *relaxed plan* heuristics, denoted h^{FF} here. Generalizing this to red-black planning, the *red-black relaxation* of an FDR task Π relative to a *variable painting*, i. e. a subset V^{R} to be painted red, is the RB task $\Pi^{\mathsf{RB}}_{V^{\mathsf{R}}} = \langle V \setminus V^{\mathsf{R}}, V^{\mathsf{R}}, A, I, G\rangle$. A plan for $\Pi^{\mathsf{RB}}_{V^{\mathsf{R}}}$ is a *red-black plan* for Π. Generating optimal red-black plans is **NP**-hard regardless of the painting simply because we always generalize MFDR. The idea is to generate satisficing red-black plans and thus obtain a *red-black plan heuristic* h^{RB} similarly as for h^{FF}. That approach is practical if the variable painting is chosen so that satisficing red-black plan generation is tractable (or sufficiently fast, anyway).

A standard means to identify structure, and therewith tractable fragments, in planning is to capture dependencies between state variables in terms of the *causal graph*. This is a digraph with vertices V. An arc (v, v') is in CG_Π if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$.

Prior work on tractability in red-black planning (Katz, Hoffmann, and Domshlak 2013b; 2013a; Domshlak, Hoffmann, and Katz 2015) considered (a) the "black causal graph" i. e. the sub-graph induced by the black variables only, and (b) the case of a single black variable. Of these, only (a) was employed for the design of heuristic functions. Herein, we improve upon (b). Method (a) is not of immediate relevance to our technical contribution, but we compare to it empirically, specifically to the most competitive heuristic h^{Mercury} as used in the Mercury system that participated in IPC'14 (Katz and Hoffmann 2014). That heuristic exploits the tractable fragment of red-black planning where the black causal graph is acyclic and every black variable is "invertible" in a particular sense. The painting strategy is geared at painting black the "most influential" variables, close to the causal graph roots.

Example 1 *As an illustrative example, we use a simplified version of the IPC benchmark TPP. Consider Figure 1. There is a truck moving along a line $l_1, \ldots, l_7$ of locations. The truck starts in the middle; the goal is to buy two units of a product, depicted in Figure 1 (a) by the barrels, where one unit is on sale at each extreme end of the road map.*

Concretely, say the encoding in FDR is as follows. The state variables are T with domain $\{l_1, \ldots, l_7\}$ for the truck position; B with domain $\{0, 1, 2\}$ for the amount of product bought already; P_1 with domain $\{0, 1\}$ for the amount of product still on sale at l_1; and P_7 with domain $\{0, 1\}$ for the amount of product still on sale at l_7. The initial state is as shown in the figure, i. e., $T = l_3$, $B = 0$, $P_1 = 1$, $P_7 = 1$. The goal is $B = 2$. The actions are:



(a) (b)

Figure 1: Our running example (a), and its causal graph (b).

- *move*(x, y)*: precondition* $\{T = l_x\}$ *and effect* $\{T = l_y\}$*, where* $x, y \in \{1, \ldots, 7\}$ *such that* $|x - y| = 1$*.*
- *buy*(x, y, z)*: precondition* $\{T = l_x, P_x = 1, B = y\}$ *and effect* $\{P_x = 0, B = z\}$*, where* $x \in \{1, 7\}$ *and* $y, z \in \{0, 1, 2\}$ *such that* $z = y + 1$*.*

The causal graph is shown in Figure 1 (b). Note that the variables pertaining to the product, i. e. B, P_1, P_7*, form a strongly connected component because of the "buy" actions.*

Consider first the painting where all variables are red, i. e., a full delete relaxation. A relaxed plan then ignores that, after buying the product at one of the two locations l_1 *or* l_7*, the product is no longer available so we have to move to the other end of the line. Instead, we can buy the product again at the same location, ending up with a relaxed plan of length* 5 *instead of the* 11 *steps needed in a real plan.*

Exactly the same problem arises in h^{Mercury}*: The only "invertible" variable here is* T*. But if we paint only* T *black, then the red-black plan still is the same as the fully delete-relaxed plan (the truck does not have to move back and forth anyhow), and we still get the same goal distance estimate* 5*.*

Now say that we paint T *red, and paint all other variables black. This is the painting our new heuristic function will use. We can no longer cheat when buying the product, i. e., we do need to buy at each of* l_1 *and* l_7*. Variable* T *is relaxed so we require* 6 *moves to reach both these locations, resulting in a red-black plan of length* 8*.*

Tractability Analysis

We focus on the case of a single black variable. This has been previously investigated by Katz et al. (2013b), but scantly only. We will discuss details below; our contribution regards a kind of dependency hitherto ignored, namely *cross-dependencies* between red and black variables:

Definition 1 *Let* $\Pi = \langle V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G \rangle$ *be a RB planning task. We say that* $v, v' \in V^{\mathsf{B}} \cup V^{\mathsf{R}}$ have different colors *if either* $v \in V^{\mathsf{B}}$ *and* $v' \in V^{\mathsf{R}}$ *or vice versa. The* red-black causal graph CG^{RB}_{Π} *of* Π *is the digraph with vertices* V *and those arcs* (v, v') *from* CG_{Π} *where* v *and* v' *have different colors. We say that* (v, v') is of type*:*

(i) BtoR *if* $v \in V^{\mathsf{B}}$*,* $v' \in V^{\mathsf{R}}$*, and there exists an action* $a \in A$ *such that* $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$*.*

(ii) RtoB *if* $v \in V^{\mathsf{R}}$*,* $v' \in V^{\mathsf{B}}$*, and there exists an action* $a \in A$ *such that* $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$*.*

(iii) EFF *else.*

We investigate the complexity of satisficing red-black planning as a function of allowing vs. disallowing each of the types (i) – (iii) of cross-dependencies individually. We completely disregard the inner structure of the black part of Π, i. e., the subset V^{B} of black variables may be arbitrary. The underlying assumption is that these variables will be pre-composed into a single black variable. Such "pre-composition" essentially means to build the cross-product of the respective variable domains (Seipp and Helmert 2011). We will refer to that cross-product as the *black state space*, and state our complexity results relative to the assumption that $|V^{\mathsf{B}}| = 1$, denoting the single black variable with v^{B}. In other words, our complexity analysis is relative to the size $|\mathcal{D}(v^{\mathsf{B}})|$ of the black state space, as opposed to the size of the input task. From a practical perspective, which we elaborate on in the next section, this makes sense provided the variable painting is chosen so that the black state space is "small".

Katz et al. (2013b) show in their Theorem 1, henceforth called "KatzP", that satisficing red-black plan generation is polynomial-time in case $|\mathcal{D}(v^{\mathsf{B}})|$ is fixed, via an algorithm that is exponential only in that parameter. They show in their Theorem 2, henceforth called "KatzNP", that deciding red-black plan existence is **NP**-complete if $|\mathcal{D}(v^{\mathsf{B}})|$ is allowed to scale. They do not investigate any structural criteria distinguishing sub-classes of the single black variable case. We close that gap here, considering the dependency types (i) – (iii) of Definition 1. The major benefit of doing so will be a polynomial-time algorithm for scaling $|\mathcal{D}(v^{\mathsf{B}})|$.

Switching each of (i) – (iii) on or off individually yields a lattice of eight sub-classes of red-black planning. It turns out that, as far as the complexity of satisficing red-black plan generation is concerned, this lattice collapses into just two classes, characterized by the presence or absence of dependencies (i): If arcs of type BtoR are allowed, then the problem is **NP**-complete even if arcs of types RtoB and EFF are disallowed. If arcs of type BtoR are disallowed, then the problem is polynomial-time even if arcs of types RtoB and EFF are allowed. We start with the negative result:

Theorem 1 *Deciding red-black plan existence for RB planning tasks with a single black variable, and without* CG^{RB}_{Π} *arcs of types RtoB and EFF, is* ***NP****-complete.*

Proof: Membership follows from KatzNP. (Plan length with a single black variable is polynomially bounded, so this holds by guess-and-check.)

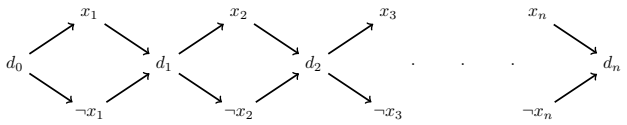


Figure 2: Illustration of the black variable v^{B} in the SAT reduction in the proof of Theorem 1.

We prove hardness by a reduction from SAT. Consider a CNF formula ϕ with propositional variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Our RB planning task has m Boolean red variables v^{R}_i, and the single black variable v^{B} has domain $\{d_0, \ldots, d_n\} \cup \{x_i, \neg x_i \mid 1 \leq i \leq n\}$. In the initial state, all v^{R}_j are set to false and v^{B} has value d_0. The goal is for all v^{R}_j to be set to true. The actions moving v^{B} have

Algorithm NoBtoR-Planning:
$R := I[V^{\mathsf{R}}] \cup$ RedFixedPoint(A^{R})
if $G[V^{\mathsf{R}}] \subseteq R$ and BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ **then**
  return "solvable" /* case (a) */
endif
$R := I[V^{\mathsf{R}}] \cup$ RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$
if $G[V^{\mathsf{R}}] \subseteq R$ **then**
  for $a \in A^{\mathsf{RB}}$ s.t. $\mathsf{pre}(a) \subseteq R$ **do**
    if BlackReachable$(R, \mathsf{eff}(a)[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ **then**
      return "solvable" /* case (b) */
    endif
  endfor
endif
return "unsolvable"

Figure 3: Algorithm used in the proof of Theorem 2.

preconditions and effects only on v^{B}, and are such that we can move as shown in Figure 2, i. e., for $1 \leq i \leq n$: from d_{i-1} to x_i; from d_{i-1} to $\neg x_i$; from x_i to d_i; and from $\neg x_i$ to d_i. For each literal $l \in c_j$ there is an action allowing to set v^{R}_j to true provided v^{B} has the correct value, i. e., for $l = x_i$ the precondition is $v^{\mathsf{B}} = x_i$, and for $l = \neg x_i$ the precondition is $v^{\mathsf{B}} = \neg x_i$. This construction does not incur any RtoB or EFF dependencies. The paths v^{B} can take correspond exactly to all possible truth value assignments. We can achieve the red goal iff one of these paths visits at least one literal from every clause, which is the case iff ϕ is satisfiable. □

The hardness part of KatzNP relies on EFF dependencies. Theorem 1 strengthens this in showing that these dependencies are not actually required for hardness.

Theorem 2 *Satisficing plan generation for RB planning tasks with a single black variable, and without CG^{RB}_Π arcs of type BtoR, is polynomial-time.*

Proof: Let $\Pi = \langle \{v^{\mathsf{B}}\}, V^{\mathsf{R}}, A, I, G \rangle$ as specified. We can partition A into the following subsets:

- $A^{\mathsf{B}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \{v^{\mathsf{B}}\}, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} = \emptyset\}$ are the actions affecting only the black variable. These actions may have red preconditions.
- $A^{\mathsf{R}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \emptyset, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} \neq \emptyset\}$ are the actions affecting only red variables. As there are no CG^{RB}_Π arcs of type BtoR, the actions in A^{R} have no black preconditions.
- $A^{\mathsf{RB}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \{v^{\mathsf{B}}\}, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} \neq \emptyset\}$ are the actions affecting both red variables and the black variable. As there are no CG^{RB}_Π arcs of type BtoR, the actions in A^{RB} have no black preconditions.

Consider Figure 3. By *RedFixedPoint*(A') for a subset $A' \subseteq A$ of actions without black preconditions, we mean all red facts reachable using only A', ignoring any black effects. This can be computed by building a relaxed planning graph over A'. By *BlackReachable*(R, d, d') we mean the question whether there exists an A^{B} path moving v^{B} from d to d', using only red preconditions from R.

Clearly, NoBtoR-Planning runs in polynomial time. If it returns "solvable", we can construct a plan π^{RB} for Π as follows. In case (a), we obtain π^{RB} by any sequence of A^{R} actions establishing RedFixedPoint(A^{R}) (there are neither black preconditions nor black effects), and attaching a sequence of A^{B} actions leading from $I[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$. In case (b), we obtain π^{RB} by: any sequence of A^{R} actions establishing RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ (there are no black preconditions); attaching the A^{RB} action a successful in the for-loop (which is applicable due to $\mathsf{pre}(a) \subseteq R$ and $\mathcal{V}(\mathsf{pre}(a)) \cap V^{\mathsf{B}} = \emptyset$); and attaching a sequence of A^{B} actions leading from $\mathsf{eff}(a)[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$. Note that, after RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$, only a single A^{RB} action a is necessary, enabling the black value $\mathsf{eff}(a)[v^{\mathsf{B}}]$ from which the black goal is A^{B}-reachable.

If there is a plan π^{RB} for Π, then NoBtoR-Planning returns "solvable". First, if π^{RB} does not use any A^{RB} action, i. e. π^{RB} consists entirely of A^{R} and A^{B} actions, then case (a) will apply because RedFixedPoint(A^{R}) contains all we can do with the former, and BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ examines all we can do with the latter. Second, say π^{RB} does use at least one A^{RB} action. RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ contains all red facts that can be achieved in Π, so in particular *(*) RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ contains all red facts true along π^{RB}*. Let a be the last A^{RB} action applied in π^{RB}. Then π^{RB} contains a path from $\mathsf{eff}(a)[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$ behind a. With (*), $\mathsf{pre}(a) \subseteq R$ and BlackReachable$(R, \mathsf{eff}(a)[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ succeeds, so case (b) will apply. □

In other words, if (a) no A^{RB} action is needed to solve Π, then we simply execute a relaxed planning fixed point prior to moving v^{B}. If (b) such an action is needed, then we mix A^{RB} with the fully-red ones in the relaxed planning fixed point, which works because, having no black preconditions, once an A^{RB} action has become applicable, it remains applicable. Note that the case distinction (a) vs. (b) is needed: When making use of the "large" fixed point RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$, there is no guarantee we can get v^{B} back into its initial value afterwards.

Example 2 *Consider again our illustrative example (cf. Figure 1), painting T red and painting all other variables black. Then v^{B} corresponds to the cross-product of variables B, P_1, and P_7; A^{B} contains the "buy" actions, A^{R} contains the "move" actions, and A^{RB} is empty.*

The call to RedFixedPoint(A^{R}) in Figure 3 results in R containing all truck positions, $R = \{T = l_1, \ldots, T = l_7\}$. The call to BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ then succeeds as, given we have both truck preconditions $T = l_1$ and $T = l_7$ required for the "buy" actions, indeed the black goal $B = 2$ is reachable. The red-black plan extracted will contain a sequence of moves reaching all of $\{T = l_1, \ldots, T = l_7\}$, followed by a sequence of two "buy" actions leading from $I[v^{\mathsf{B}}] = \{B = 0, P_1 = 1, P_2 = 1\}$ to $G[v^{\mathsf{B}}] = \{B = 2\}$.

Theorem 2 is a substantial improvement over KatzP in terms of the scaling behavior in $|\mathcal{D}(v^{\mathsf{B}})|$. KatzP is based on an algorithm with runtime exponential in $|\mathcal{D}(v^{\mathsf{B}})|$. Our NoBtoR-Planning has low-order polynomial runtime in that

parameter, in fact all we need to do is find paths in a graph of size $|\mathcal{D}(v^{\mathsf{B}})|$. This dramatic complexity reduction is obtained at the price of disallowing BtoR dependencies.

Heuristic Function

Assume an input FDR planning task Π. As indicated, we will choose a painting (a subset V^{R} of red variables) so that BtoR dependencies do not exist, and for each search state s generate a heuristic value by running NoBtoR-Planning with s as the initial state. We describe our painting strategy in the next section. Some words are in order regarding the heuristic function itself, which diverges from our previous theoretical discussion – Figure 3 and Theorem 2 – in several important aspects.

While the previous section assumed that the entire black state space is pre-composed into a single black variable v^{B}, that assumption was only made for convenience. In practice there is no need for such pre-composition. We instead run NoBtoR-Planning with the BlackReachable(R, d, d') calls implemented as a forward state space search within the projection onto the black variables, using only those black-affecting actions whose red preconditions are contained in the current set of red facts R. This is straightforward, and avoids having to generate the entire black state space up front – instead, we will only generate those parts actually needed during red-black plan generation as requested by the surrounding search. Still, of course for this to be feasible we need to keep the size of the black state space at bay.

That said, actually what we need to keep at bay is not the black state space itself, but its *weakly connected components*. As the red variables are taken out of this part of the problem, chances are that the remaining part will contain separate components.

Example 3 *In our running example, say there are several different kinds of products, i. e. the truck needs to buy a goal amount of several products. (This is indeed the case in the TPP benchmark suite as used in the IPC.) The state variables for each product then form an SCC like the variables B, P_1, P_7 in Figure 1 (b), mutually separated from each other by taking out (painting red) the central variable T.*

We can *decompose* the black state space, handling each connected component of variables $V_c^{\mathsf{B}} \subseteq V^{\mathsf{B}}$ separately. When calling BlackReachable(R, d, d'), we do not call a single state space search within the projection onto V^{B}, but call one state space search within the projection onto V_c^{B}, for every component V_c^{B}. The overall search is successful if all its components are, and in that case the overall solution path results from simple concatenation.

We finally employ several simple optimizations: *black state space results caching*, *stop search*, and *optimized red-black plan extraction*. The first of these is important as the heuristic function will be called on the same black state space many times during search, and within each call there may be several questions about paths from d to d' through that state space. The same pairs d and d' may reappear many times in the calls to BlackReachable(R, d, d'), so we can avoid duplicate effort simply by caching these results. Precisely, our cache consists of pairs (d, d') along with a black path $\pi(d, d')$ from d to d'. (In preliminary experiments, caching the actual triples (R, d, d') led to high memory consumption.) Whenever a call to BlackReachable(R, d, d') is made, we check whether (d, d') is in the cache, and if so check whether $\pi(d, d')$ works given R, i. e., contains only actions whose red preconditions are contained in R. If that is not so, or if (d, d') is not in the cache at all yet, we run the (decomposed) state space search, and in case of success add its result to the cache.

Stop search is the same as already used in (and found to be important in) Katz et al.'s previous work on red-black plan heuristics. If the red-black plan π^{RB} generated for a search state s is actually executable in the original FDR input planning task, then we terminate search immediately and output the path to s, followed by π^{RB}, as the solution.

Finally, the red-black plans π^{RB} described in the proof of Theorem 2 are of course highly redundant in that they execute the entire red fixed points, as opposed to establishing only those red facts $R^g \subseteq R$ required by the red goal $G[V^{\mathsf{R}}]$, and required as red preconditions on the solution black path found by BlackReachable(R, d, d'). We address this straightforwardly following the usual relaxed planning approach. The forward red fixed point phase is followed by a backward red plan extraction phase, in which we select supporters for R^g and the red subgoals it generates.

Painting Strategy

Given an input FDR planning task Π, we need to choose our painting V^{R} such that the red-black causal graph CG_Π^{RB} has no BtoR dependencies. A convenient view for doing so is to perceive the causal graph CG_Π as a DAG D of SCCs in which the root SCCs are at the top and the leaf SCCs at the bottom: Our task is then equivalent to drawing a "horizontal line" anywhere through D, painting the top part red, and painting the bottom part black. We say that such a painting is *non-trivial* if the bottom part is non-empty.

Example 4 *In our running example, the only non-trivial painting is the one illustrated in Figure 4.*

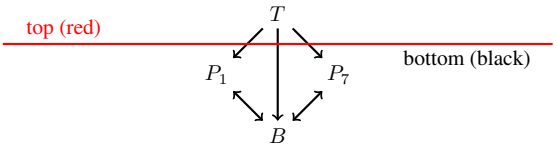


Figure 4: The painting in our running example.

If there are several different kinds of products as described in Example 3, then the state variables for each product form a separate component in the bottom part. If there are several trucks, then the "horizontal line" may put any non-empty subset of trucks into the top part.

We implemented a simple painting strategy accommodating the above. The strategy has an input parameter N imposing an upper bound on the (conservatively) estimated size of the decomposed black state space. Starting with the DAG D

		without preferred operators								with preferred operators							
				$N =$								$N =$					
domain	#	h^{Mercury}	h^{FF}	0	$1k$	$10k$	$100k$	$1m$	$10m$	h^{Mercury}	h^{FF}	0	$1k$	$10k$	$100k$	$1m$	$10m$
Logistics00	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28
Logistics98	35	35	26	23	23	23	23	23	23	35	35	32	32	32	32	32	32
Miconic	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
ParcPrinter08	30	30	26	30	30	30	30	30	30	30	26	30	30	30	30	30	30
ParcPrinter11	20	20	12	20	20	20	20	20	20	20	12	20	20	20	20	20	20
Pathways	30	11	11	8	10	10	10	10	10	30	20	23	23	23	23	23	23
Rovers	40	27	23	23	23	24	26	25	24	40	40	40	40	40	40	40	40
Satellite	36	36	30	26	26	26	26	25	26	36	36	35	35	35	35	35	35
TPP	30	23	22	18	18	18	18	18	19	30	30	30	30	30	30	30	30
Woodworking08	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Woodworking11	20	20	19	19	19	20	20	20	20	20	20	20	20	20	20	20	20
Zenotravel	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
$\sum$	469	430	397	395	397	399	401	399	400	469	447	458	458	458	458	458	458

Table 1: Coverage results. All heuristics are run with FD's greedy best-first search, single-queue for configurations without preferred operators, double-queue for configurations with preferred operators. The preferred operators are taken from h^{FF} in all cases (see text).

of SCCs over the original causal graph, and with the empty set V^{B} of black variables, iterate the following steps:

1. Set the candidates for inclusion to be all leaf SCCs $V_l \subseteq V$ in D.
2. Select a V_l where $\prod_{v \in V'} |\mathcal{D}(v)|$ is minimal.
3. Set $V' := V^{\mathsf{B}} \cup V_l$ and find the weakly connected components $V_c^{\mathsf{B}} \subseteq V'$.
4. If $\sum_{V_c^{\mathsf{B}}} \prod_{v \in V_c^{\mathsf{B}}} |\mathcal{D}(v)| \leq N$, set $V^{\mathsf{B}} := V'$, remove V_l from D, and iterate; else, terminate.

Example 5 *In our running example, this strategy will result in exactly the painting displayed in Figure 4, provided N is choosen large enough to accommodate the variable subset $\{B, P_1, P_7\}$, but not large enough to accommodate the entire set of variables.*

If there are several different kinds of products, as in the IPC TPP domain, then N does not have to be large to accommodate all products (as each is a separate component), but would have to be huge to accommodate any truck (which would reconnect all these components). Hence, for a broad range of settings of N, we end up painting the products black and the trucks red, as desired.

Note that our painting strategy may terminate with the trivial painting ($V^{\mathsf{B}} = \emptyset$), namely if even the smallest candidate V_l breaks the size bound N. This will happen, in particular, on all input tasks Π whose causal graph is a single SCC, unless N is large enough to accommodate the entire state space. Therefore, in practice, we exclude input tasks whose causal graph is strongly connected.

Experiments

Our techniques are implemented in Fast Downward (FD) (Helmert 2006). For our painting strategy, we experiment with the size bounds $N \in \{1k, 10k, 100k, 1m, 10m\}$ ("m" meaning "million"). We run all IPC STRIPS benchmarks, precisely their satisficing-planning test suites, where we obtain non-trivial paintings. This excludes domains whose causal graphs are strongly connected, and it excludes domains where even the smallest leaf SCCs break our size bounds. It turns out that, given this, only 9 benchmark domains qualify, 3 of which have been used in two IPC editions so that we end up with 12 test suites.

As our contribution consists in a new heuristic function, we fix the search algorithm, namely FD's greedy best-first search with lazy evaluation, and evaluate the heuristic function against its closest relatives. Foremost, we compare to the standard relaxed plan heuristic h^{FF}, which we set out to improve upon. More specifically, we compare to two implementations of h^{FF}: the one from the FD distribution, and our own heuristic with size bound $N = 0$. The former is more "standard", but differs from our heuristic even in the case $N = 0$ because these are separate implementations that do not coincide exactly in terms of tie breaking. As we shall see, this seemingly small difference can significantly affect performance. To obtain a more precise picture of which differences are due to the black variables rather than other details, we use $N = 0$, i. e. "our own" h^{FF} implementation, as the main baseline. We also run h^{Mercury}, as a representative of the state of the art in alternate red-black plan heuristics.

A few words are in order regarding preferred operators. As was previously observed by Katz et al., h^{Mercury} yields best performance when using the standard preferred operators extracted from h^{FF}: The latter is computed as part of computing h^{Mercury} anyhow, and the standard preferred operators tend to work better than variants Katz et al. tried trying to exploit the handling of black variables in h^{Mercury}. For our own heuristics, we made a similar observation, in that we experimented with variants of preferred operators specific to these, but found that using the standard preferred operators from h^{FF} gave better results. This is true despite the fact that our heuristics do *not* compute h^{FF} as part of the process. The preferred operators are obtained by a separate call to FD's standard implementation of h^{FF}, on every search state. Hence, in what follows, all heuristics reported use the exact

same method to generate preferred operators.

Table 1 shows coverage results. Observe first that, in terms of this most basic performance parameter, h^{Mercury} dominates all other heuristics, across all domains and regardless whether or not preferred operators are being used. Recall here that, in contrast to our heuristics which paint black the variables "close to the causal graph leaves", h^{Mercury} uses paintings that paint black the variables "close to the causal graph roots". Although in principle the former kind of painting can be of advantage as illustrated in Example 1, as previously indicated the latter kind of painting tends to work better on the IPC benchmarks. We provide a per-instance comparison of h^{Mercury} against our heuristics, in Rovers and TPP which turn out to be the most interesting domains for these heuristics, further below (Table 3). For now, let's focus on the comparison to the baseline, h^{FF}.

Note first the influence of tie breaking: Without preferred operators, $N = 0$ has a dramatic advantage over h^{FF} in ParcPrinter, and smaller but significant disadvantages in Logistics98, Pathways, Satellite, and TPP. With preferred operators, the coverage differences get smoothed out, because with the pruning the instances become much easier to solve so the performance differences due to the different heuristics do not affect coverage as much anymore. The upshot is that only the advantage in ParcPrinter, but none of the disadvantages, remain. As these differences have nothing to do with our contribution, we will from now on not discuss h^{FF} as implemented in FD, and instead use the baseline $N = 0$.

Considering coverage as a function of N, observe that, with preferred operators, there are no changes whatsoever, again because with the pruning the instances become much easier to solve. Without preferred operators, increasing N and thus the black part of our heuristic function affects coverage in Pathways, Rovers, Satellite, TPP, and Woodworking11. With the single exception of Satellite for $N = 1m$, the coverage change relative to the baseline $N = 0$ is positive. However, the extent of the coverage increase is small in almost all cases. We now examine this more closely, considering more fine-grained performance parameters.

Table 2 considers the number of evaluated states during search, and search runtime, in terms of improvement factors i. e. the factor by which evaluations/search time reduce relative to the baseline $N = 0$. As we can see in the top half of the table, the (geo)mean improvement factors are almost consistently greater than 1 (the most notable exception being Pathways), i. e., there typically is an improvement on average (although: see below). The actual search time, on the other hand, almost consistently gets worse, with a very pronounced tendency for the "improvement factor" to be < 1, and to decrease as a function of N. The exceptions in this regard are Rovers, and especially TPP where, quite contrary to the common trend, the search time improvement factor *grows* as a function of N. This makes sense as Rovers and TPP clearly stand out as the two domains with the highest evaluations improvement factors.

Per-instance data sheds additional light on this. In Logistics, Miconic, ParcPrinter, Pathways, and Zenotravel, almost all search space reductions obtained are on the smallest instances, where N is large enough to accommodate the entire state space and hence, trivially, the number of evaluations is 1. On almost all larger instances of these domains, the search spaces are identical, explaining the bad search time results previously observed. In Satellite and Woodworking, the results are mixed. There are substantial improvements also on some large instances, but the evaluations improvement factor is always smaller than 6, with the single exception of Woodworking08 instance p24 where for $N \geq 10k$ it is 17.23. In contrast, in Rovers the largest evaluations improvement factor is 4612, and in TPP it is 17317.

domain	#	$1k$	$10k$	$100k$	$1m$	$10m$
evaluations						
Logistics00	28	1.00	1.05	1.43	3.71	3.71
Logistics98	23	1.00	0.98	1.01	1.22	1.35
Miconic	150	1.24	1.41	1.86	2.18	3.12
ParcPrinter08	30	1.07	1.38	1.52	1.52	1.71
ParcPrinter11	20	1.00	1.03	1.08	1.08	1.09
Pathways	8	0.71	0.71	0.71	0.88	0.88
Rovers	19	1.60	1.95	5.10	5.84	5.16
Satellite	25	1.04	1.83	1.61	2.18	2.40
TPP	17	1.83	3.76	5.90	20.89	27.54
Woodworking08	30	1.54	2.06	2.06	2.06	2.06
Woodworking11	19	1.08	1.68	1.76	1.76	1.76
Zenotravel	20	1.14	1.14	0.87	1.13	2.27
search time						
Logistics00	28	1.00	1.00	0.94	0.59	0.59
Logistics98	23	1.00	1.00	0.93	0.80	0.58
Miconic	150	0.73	0.73	0.70	0.63	0.44
ParcPrinter08	30	1.00	1.00	1.00	1.00	0.44
ParcPrinter11	20	1.00	0.95	0.96	0.96	0.23
Pathways	8	0.97	0.97	0.96	0.96	0.93
Rovers	19	1.44	1.74	2.01	1.56	0.87
Satellite	25	0.87	1.07	0.90	0.63	0.24
TPP	17	0.98	1.39	1.78	3.75	4.67
Woodworking08	30	0.94	0.86	0.86	0.86	0.86
Woodworking11	19	0.87	0.65	0.67	0.67	0.67
Zenotravel	20	1.00	1.00	0.95	0.78	0.43

Table 2: Improvement factors relative to $N = 0$. Per-domain geometric mean over the set of instances commonly solved for all values of N. Without preferred operators.

Table 3 shows per-instance data on Rovers and TPP, where our techniques are most interesting. We also include h^{Mercury} here for a detailed comparison. $N = 1k$ and $N = 100k$ are left out of the table for lack of space, and as these configurations are always dominated by at least one other value of N here. With respect to the behavior against the baseline $N = 0$, clearly in both domains drastic evaluations and search time improvements can be obtained. It should be said though that there is an unfortunate tendency for our red-black heuristics to have advantages in the smaller instances, rather than the larger ones. This is presumably because, in smaller instances (even disregarding the pathological case where the entire state space fits into the black part of our heuristic) we have a better chance to capture complex variable interactions inside the black part, and hence obtain substantially better heuristic values.

With respect to h^{Mercury}, the conclusion can only be that the previous approach to red-black plan heuristics – painting

	h^{Mercury}		$N=0$		$N=10k$		$N=1m$		$N=10m$	
	E	T	E	T	E	T	E	T	E	T
Rovers										
p01	5	0.1	35	0.1	31	0.1	1	0.1	1	0.1
p02	6	0.1	6	0.1	1	0.1	1	0.1	1	0.1
p03	1	0.1	62	0.1	112	0.1	1	0.1	1	0.1
p04	1	0.1	17	0.1	21	0.1	1	0.1	1	0.1
p05	119	0.1	114	0.1	170	0.1	117	0.1	117	0.1
p06	304	0.1	543	0.1	485	0.1	485	0.1	285	0.6
p07	70	0.1	331	0.1	334	0.1	162	1.5	175	7.1
p08	116	0.1	1742189	46.3	451078	15.4	603	0.1	1929	28.2
p09	358	0.1	2773	0.1	1792	0.1	2120	0.1	2120	0.1
p10	578	0.1	441	0.1	441	0.1	244	0.4	244	0.4
p11	1047	0.1	85832	2.7	85787	3.1	85787	3.1	85787	3.1
p12	6	0.1	606	0.1	958	0.1	301	0.3	698	0.4
p13	25037	2.13	1944	0.1	2578	0.1	2882	0.3	2882	0.3
p14	294	0.1	5161720	208.5	1467	0.1	732	0.2	1119	0.8
p15	1035	0.1	–	–	5024	0.3	3520	1.5	4782	3.3
p16	358	0.1	–	–	11895	0.6	11895	0.6	11895	0.6
p17	1139	0.1	–	–	–	–	3340	0.3	3340	0.3
p18	2156	0.19	93372	6.6	47472	3.6	47472	3.6	48620	6.0
p19	180979	22.08	370650	38.3	452905	47.7	470758	47.7	470758	47.3
p20	–	–	1782100	233.1	–	–	1828671	248.3	1828671	248.0
p22	3674	0.63	2478919	339.2	1707251	237.1	1707251	232.4	1707251	236.0
p23	24347	5.77	–	–	–	–	–	–	–	–
p25	1	0.1	2677	0.3	31890	4.5	31248	4.2	44688	5.8
p26	7338129	1418.16	117583	13.7	83724	11.7	7263721	974.8	–	–
p27	61575	13.73	–	–	6737329	1158.0	6737329	1138.6	6737329	1148.3
p28	6346	1.95	1066434	223.8	15321	3.8	18002	4.3	18002	4.3
p29	8409	2.71	1298473	251.4	–	–	–	–	–	–
p30	–	–	5940371	1134.0	333303	108.0	–	–	–	–
p34	60144	38.59	–	–	–	–	–	–	–	–
TPP										
p01	1	0.1	5	0.1	1	0.1	1	0.1	1	0.1
p02	1	0.1	9	0.1	1	0.1	1	0.1	1	0.1
p03	1	0.1	13	0.1	1	0.1	1	0.1	1	0.1
p04	1	0.1	17	0.1	17	0.1	1	0.1	1	0.1
p05	1	0.1	22	0.1	25	0.1	25	0.1	25	0.1
p06	38	0.1	107	0.1	46	0.1	46	0.1	46	0.1
p07	1672	0.1	1756	0.1	68	0.1	68	0.1	68	0.1
p08	2462	0.1	2534	0.1	71	0.1	71	0.1	71	0.1
p09	6753	0.28	2963	0.1	299	0.1	121	0.1	121	0.1
p10	24370	1.24	10712	0.5	1061	0.1	147	0.1	147	0.1
p11	15519	1.3	1610504	99.8	56090	4.0	93	0.1	93	0.1
p12	54852	4.37	1340734	91.3	699377	55.7	109	0.1	109	0.1
p13	38205	3.5	40291	3.5	40291	3.5	472	0.1	475	0.1
p14	57981	7.37	35089	3.6	35089	3.6	552	0.1	555	0.1
p15	52722	7.32	22842	2.4	22842	2.5	70467	8.1	283	0.1
p16	298618	77.47	247304	49.6	247304	48.4	112610	19.6	253727	38.0
p17	2660716	774.05	–	–	–	–	–	–	1611051	671.4
p18	264855	77.25	–	–	–	–	–	–	–	–
p19	1957381	639.02	1710323	509.3	1710323	508.8	1710323	505.6	–	–
p20	–	–	–	–	–	–	–	–	4161990	1272.4
p21	811226	578.23	–	–	–	–	–	–	–	–
p22	652741	372.74	–	–	–	–	–	–	–	–
p23	1329626	902.09	2432228	1362.0	2432228	1365.3	2432228	1367.6	2432228	1370.7
p24	1253699	801.71	–	–	–	–	–	–	–	–

Table 3: Evaluations and search time in Rovers and TPP. "E" evaluations, "T" search time. Without preferred operators.

variables "close to the root" black, as opposed to painting variables "close to the leaves" black as we do here – works better in practice. There are rare cases where our new heuristics have an advantage, most notably in Rovers p20, p26, p30, and TPP p5–p17, p19, p20. But overall, especially on the largest instances, h^{Mercury} tends to be better. We remark that, with preferred operators switched on, the advantage of h^{Mercury} tends to be even more pronounced because the few cases that are hard for it in Table 3 become easy.

A few words are in order regarding plan quality, by which, since we only consider uniform action costs in the experiments, we mean plan length. Comparing our most informed configuration, $N = 10m$, to our pure delete relaxed baseline, i. e. our heuristic with $N = 0$, it turns out that the value of N hardly influences the quality of the plans found. Without using preferred operators, the average per-domain gain/loss of one configuration over the other is always $< 3\%$. The only domain where solution quality differs more significantly is TPP, where the generated plans for $N = 10m$ are 23.3% shorter on average than those with $N = 0$. This reduces to 10% when preferred operators are switched on. In the other domains, not much changes when enabling preferred operators; the average gain/loss per domain is less than 4.4%.

Comparing our $N = 10m$ configuration to h^{Mercury}, having preferred operators disabled, the plan quality is only slightly different in most domains ($< 3.1\%$ gain/loss on average). Results differ more significantly in Miconic and TPP. In the former, our plans are 25% longer than those found using h^{Mercury}; in the latter, our plans are 25% shorter. Enabling preferred operators does not change much, except in Woodworking, where our plans are on average 19.1% (16.5%) shorter in the IPC'08 (IPC'11) instance suites.

Conclusion

Our investigation has brought new insights into the interaction between red and black variables in red-black planning. The practical heuristic function resulting from this can, in principle, improve over standard relaxed plan heuristics as well as known red-black plan heuristics. In practice – as far as captured by IPC benchmarks – unfortunately such improvements are rare. We believe this is a valuable insight for further research on red-black planning. It remains to be seen whether our tractability analysis can be extended and/or exploited in some other, more practically fruitful, way. The most promising option seems to be to seek tractable special cases of black-to-red (BtoR) dependencies, potentially by restrictions onto the DTG (the variable-value transitions) of the black variable weaker than the "invertibility" criterion imposed by Katz et al.

Acknowledgments. We thank Carmel Domshlak for discussions. We thank the anonymous reviewers, whose comments helped to improve the paper. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, and by the EU FP7 Programme under grant agreement 295261 (MEALS).

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 10–17. AAAI Press.

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

Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds. 2012. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph'lp heuristic for numeric planning domains. In Rintanen et al. (2008), 52–59.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research* 46:343–412.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Fox, M., and Long, D. 2001. Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In Nebel, B., ed., *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 445–450. Seattle, Washington, USA: Morgan Kaufmann.

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

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen et al. (2008), 140–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Koenig, S.; Zilberstein, S.; and Koehler, J., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 161–170. Whistler, Canada: Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 105–113. AAAI Press.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, 43–47.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M., eds., *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*, 489–495. Bellevue, WA, USA: AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 126–134. Rome, Italy: AAAI Press.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In Ghallab, M., ed., *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, 588–592. Patras, Greece: Wiley.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet et al. (2012), 128–136.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

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

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. In Bonet et al. (2012), 181–189.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

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

Seipp, J., and Helmert, M. 2011. Fluent merging for classical planning problems. In *ICAPS 2011 Workshop on Knowledge Engineering for Planning and Scheduling*, 47–53.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessiere, C., ed., *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.

From Fork Decoupling to Star-Topology Decoupling

Daniel Gnad and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de

Carmel Domshlak
Technion
Haifa, Israel
dcarmel@ie.technion.ac.il

Abstract

Fork decoupling is a recent approach to exploiting problem structure in state space search. The problem is assumed to take the form of a fork, where a single (large) *center* component provides preconditions for several (small) *leaf* components. The leaves are then conditionally independent in the sense that, given a fixed center path π^C, the *compliant* leaf moves – those leaf moves enabled by the preconditions supplied along π^C – can be scheduled independently for each leaf. *Fork-decoupled state space search* exploits this through conducting a regular search over center paths, augmented with maintenance of the compliant paths for each leaf individually. We herein show that the same ideas apply to much more general *star-topology* structures, where leaves may supply preconditions for the center, and actions may affect several leaves simultaneously as long as they also affect the center. Our empirical evaluation in planning, super-imposing star topologies by automatically grouping the state variables into suitable components, shows the merits of the approach.

Introduction

In classical AI planning, large deterministic transition systems are described in terms of a set of finite-domain state variables, and actions specified via preconditions and effects over these state variables. The task is to find a sequence of actions leading from a given initial state to a state that satisfies a given goal condition. *Factored planning* is one traditional approach towards doing so effectively. The idea is to decouple the planning task into subsets, *factors*, of state variables. In *localized* factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; 2013; Fabre et al. 2010), two factors interact if they are affected by common actions, and a global plan needs to comply with these *cross-factor interactions*. In *hierarchical* factored planning (e. g. (Knoblock 1994; Kelareva et al. 2007; Wang and Williams 2015)), the factors are used within a hierarchy of increasingly more detailed abstraction levels, search refining abstract plans as it proceeds down the hierarchy, and backtracking if an abstract plan has no refinement.

Both localized and hierarchical factored planning have traditionally been viewed as the resolution of complex interactions between (relatively) small factors. In recent work, Gnad and Hoffmann (2015) (henceforth: GH) proposed to turn this upside down, fixing a *simple* interaction profile the factoring should induce, at the cost of potentially very *large* factors. They baptize this approach *target-profile factoring*, and develop a concrete instance where the target profile is a *fork*: a single (large) *center* factor provides preconditions for several (small) *leaf* factors, and no other cross-factor interactions are present. They introduce a simple and quick *factoring strategy* which, given an arbitrary input planning task, analyzes the state variable dependencies and either outputs a *fork factoring*, or *abstains* if the relevant stucture is not there (the task can then be handed over to other methods).

Say the factoring strategy does not abstain. We then face, not a general planning problem, but a fork planning problem. GH's key observation is that these can be solved via *fork-decoupled state space search*: The leaves are conditionally independent in the sense that, given a fixed center path π^C, the *compliant* leaf moves – those leaf moves enabled by the preconditions supplied along π^C – can be scheduled independently for each leaf. This can be exploited by searching only over center paths, and maintaining the compliant paths separately for each leaf, thus avoiding the enumeration of state combinations across leaves. GH show that this substantially reduces the number of reachable states in (non-abstained) planning benchmarks, almost always by at least $1-2$ orders of magnitude, up to 6 orders of magnitude in one domain (TPP). They show how to connect to classical planning heuristics, and to standard search methods, guaranteeing plan optimality under the same conditions as before.

We herein extend GH's ideas to *star-topology* structures, where the center still supplies preconditions for the leaves, but also the leaves may supply preconditions for the center, and actions may affect several leaves simultaneously as long as they also affect the center. The connection to standard heuristics and search methods remains valid. We run experiments on the planning competition benchmarks.

We place our work in the planning context for the direct connection to GH. However, note that trying to factorize a general input problem, and having to abstain in case the saught structure is not present, really is an artefact of the general-planning context. Star-topology decoupling applies, in principle, to the state space of any system that naturally has a star topology. As star topology is a classical design paradigm in many areas of CS, such systems abound.

Preliminaries

We use a finite-domain state variable formalization of planning (e. g. (Bäckström and Nebel 1995; Helmert 2006)). A *finite-domain representation* planning task, short FDR task, is a quadruple $\Pi = \langle V, A, I, G\rangle$. V is a set of *state variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*. Each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a)\rangle$ where the *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$ is the action's non-negative *cost*.

For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables instantiated by p. For any $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the assignment to V' made by p. An action a is *applicable* in a state s if $\mathsf{pre}(a) \subseteq s$, i. e., if $s[v] = \mathsf{pre}(a)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of each $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere; the outcome state is denoted $s[\![a]\!]$. We also use this notation for partial states p: by $p[\![a]\!]$ we denote the assignment over-writing p with $\mathsf{eff}(a)$ where both p and $\mathsf{eff}(a)$ are defined. The outcome state of applying a sequence of (respectively applicable) actions is denoted $s[\![\langle a_1, \ldots, a_n\rangle]\!]$. A *plan* for Π is an action sequence s.t. $G \subseteq I[\![\langle a_1, \ldots, a_n\rangle]\!]$. The plan is *optimal* if its summed-up cost is minimal among all plans for Π.

The *causal graph* of a planning task captures state variable dependencies (e. g. (Knoblock 1994; Jonsson and Bäckström 1995; Brafman and Domshlak 2003; Helmert 2006)). We use the commonly employed definition in the FDR context, where the causal graph *CG* is a directed graph over vertices V, with an arc from v to v', which we denote $(v \to v')$, if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$. In words, the causal graph captures precondition-effect as well as effect-effect dependencies, as result from the action descriptions. A simple intuition is that, whenever $(v \to v')$ is an arc in *CG*, changing the value of v' may involve changing that of v as well. We assume for simplicity that *CG* is weakly connected (this is wlog: else, the task can be equivalently split into several independent tasks).

We will also need the notion of a *support graph*, *SuppG*, similarly as used e. g. by Hoffmann (2011). *SuppG* is like *CG* except its arcs are only those $(v \to v')$ where there exists an action $a \in A$ such that $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$. In words, the support graph captures only the precondition-effect dependencies, not effect-effect dependencies. This more restricted concept will be needed to conveniently describe our notion of star topologies, for which purpose the effect-effect arcs in *CG* are not suitable.

As an illustrative example, we will consider a simple transportation-like FDR planning task $\Pi = \langle V, A, I, G\rangle$ with one package p and two trucks t_A, t_B, defined as follows. $V = \{p, t_A, t_B\}$ where $\mathcal{D}(p) = \{A, B, l_1, l_2, l_3\}$ and $\mathcal{D}(t_A) = \mathcal{D}(t_B) = \{l_1, l_2, l_3\}$. The initial state is $I = \{p = l_1, t_A = l_1, t_B = l_3\}$, i. e., p and t_A start at l_1, and t_B starts at l_3. The goal is $G = \{p = l_3\}$. The actions (all with cost 1) are truck moves and load/unload:

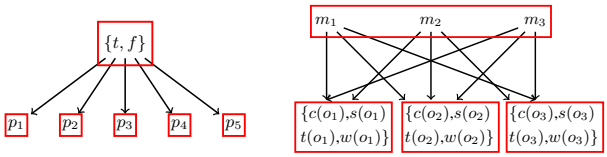


Figure 1: (Gnad and Hoffmann 2015) Possible fork factorings in transportation with fuel consumption (left), and job-planning problems (right).

- move(x, y, z): precondition $\{t_x = y\}$ and effect $\{t_x = z\}$, where $x \in \{A, B\}$ and $\{y, z\} \in \{\{l_1, l_2\}, \{l_2, l_3\}\}$.
- load(x, y): precondition $\{t_x = y, p = y\}$ and effect $\{p = x\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.
- unload(x, y): precondition $\{t_x = y, p = x\}$ and effect $\{p = y\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.

The causal graph and support graph of this task are identical. Their arcs are $(t_A \to p)$ and $(t_B \to p)$.

Fork Decoupling

We give a brief summary of fork decoupling, in a form suitable for describing our extension to star topologies.

Definition 1 (Fork Factoring (GH)) *Let Π be an FDR task with variables V. A* factoring *$\mathcal{F}$ is a partition of V into non-empty subsets F, called* factors. *The* interaction graph *IG($\mathcal{F}$) of $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $(F \to F')$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $(v \to v')$ is an arc in CG.*

$\mathcal{F}$ is a fork factoring *if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. the arcs in IG($\mathcal{F}$) are exactly $\{(F^C \to F^L) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$. F^C is the* center *of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are* leaves. *We also consider the* trivial factoring *where $F^C = V$ and $\mathcal{F}^L = \emptyset$, and the* pathological factoring *where $F^C = \emptyset$ and $\mathcal{F}^L = \{V\}$.*

The only cross-factor interactions in a fork factoring consist in the center factor establishing preconditions for actions moving individual leaf factors. We use the word "center", instead of GH's "root", to align the terminology with star topologies. Regarding the trivial and pathological cases, in the former decoupled search simplifies to standard search, and in the latter the entire problem is pushed into the single "leaf". Note that, when we say that $\mathcal{F}$ is a fork factoring, we explicitly exclude these cases.

In our example, we will consider the fork factoring where $F^C = \{t_A, t_B\}$ and the single leaf is $F^L = \{p\}$.

Given an arbitrary FDR task Π as input, as pointed out by GH, a fork factoring – if one exists, which is the case iff the causal graph has more than one strongly connected component (SCC) – can be found automatically based on a simple causal graph analysis. We describe GH's strategy later, in the experiments, along with our own generalized strategies. For illustration, Figure 1 already shows factorings that GH's strategy may find, on practical problems akin to planning benchmarks. On the left, a truck t with fuel supply f transports packages $p_1, \ldots, p_n$. On the right, objects o_i are independent except for sharing the machines.

We need some terminology, that we will use also for star topologies later on. Assume an FDR task $\Pi = \langle V, A, I, G\rangle$ and a fork factoring $\mathcal{F}$ with center F^C and leaves $F^L \in \mathcal{F}^L$.

We refer to the actions A^C affecting the center as *center actions*, notation convention a^C, and to all other actions as *leaf actions*, notation convention a^L. For the set of actions affecting one particular $F^L \in \mathcal{F}^L$, we write $A^L|_{F^L}$. As $\mathcal{F}$ is a fork factoring, the center actions A^C have preconditions and effects only on F^C. The leaf actions $A^L|_{F^L}$ have preconditions only on $F^C \cup F^L$, and effects only on F^L. The sets A^C and $A^L|_{F^L}$ form a partition of the original action set A.

A *center path* is a sequence of center actions applicable to I; a *leaf path* is a sequence of leaf actions applicable to I when ignoring preconditions on the center. Value assignments to F^C are *center states*, notated s^C, and value assignments to any $F^L \in \mathcal{F}^L$ are *leaf states*, notated s^L. For the leaf states of one particular $F^L \in \mathcal{F}^L$, we write $S^L|_{F^L}$, and for the set of all leaf states we write S^L. A center state s^C is a *goal center state* if $s^C \supseteq G[F^C]$, and a leaf state $s^L \in S^L|_{F^L}$ is a *goal leaf state* if $s^L \supseteq G[F^L]$.

The idea in fork-decoupling is to augment a regular search over center paths with maintenance of cheapest compliant leaf paths for each leaf. A leaf path $\pi^L = \langle a^L_1, \ldots, a^L_n \rangle$ complies with center path π^C if we can schedule the a^L_i at monotonically increasing points alongside π^C so that each a^L_i is enabled in the respective center state. Formally:

Definition 2 (Fork-Compliant Path (GH)) *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C, and π^C a center path traversing center states $\langle s^C_0, \ldots, s^C_n \rangle$. For a leaf path $\pi^L = \langle a^L_1, \ldots, a^L_m \rangle$, an* embedding *into π^C is a monotonically increasing function $t : \{1, \ldots, m\} \mapsto \{0, \ldots, n\}$ so that, for every $i \in \{1, \ldots, m\}$, $\mathsf{pre}(a^L_i)[F^C] \subseteq s^C_{t(i)}$. We say that π^L* fork-complies *with π^C, also π^L is π^C-*fork-compliant*, if an embedding exists.*

Where the center path in question is clear from context, or when discussing compliant paths in general, we will omit "π^C" and simply talk about *compliant* leaf paths.

In our example, $\pi^L = \langle \mathrm{load}(A, l_1) \rangle$ complies with the empty center path, and $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2) \rangle$ complies with the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$. But $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_3) \rangle$ does not comply with π^C as the required precondition $t_A = l_3$ is not established on π^C. And $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2), \mathrm{load}(A, l_2), \mathrm{unload}(A, l_1) \rangle$ does not comply with π^C as the last precondition $t_A = l_1$ does not appear behind $t_A = l_2$ on π^C.

The notion of compliant paths is a reformulation of plans for the original input planning task Π, in the following sense. Say π is a plan for Π, and say π^C is the sub-sequence of center actions in π. Then π^C is a center path. For each leaf $F^L \in \mathcal{F}^L$, say π^L is the sub-sequence of $A^L|_{F^L}$ actions in π. Then π^L is a leaf path, and is π^C-fork-compliant because π schedules π^L along with π^C in a way so that its center preconditions are fulfilled. Vice versa, if a center path π^C reaches a goal center state, and can be augmented with π^C-fork-compliant leaf paths π^L reaching goal leaf states, then the embedding of the π^L into π^C yields a plan for Π. Hence *the plans for Π are in one-to-one correspondence with center paths augmented with compliant leaf paths.*

GH define the *fork-decoupled state space* Θ^ϕ, in which each *fork-decoupled state* s is a pair $\langle \mathsf{center}(s), \mathsf{prices}(s) \rangle$ of a center state $\mathsf{center}(s)$ along with a *pricing function* $\mathsf{prices}(s) : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$. The paths in Θ^ϕ correspond to center paths, i. e., the *fork-decoupled initial state* I^ϕ has $\mathsf{center}(I^\phi) = I[F^C]$, and the transitions over center states are exactly those induced by the center actions. The pricing functions are maintained so that, for every center path π^C ending in fork-decoupled state s, and for every leaf state s^L, $\mathsf{prices}(s)[s^L]$ equals the cost of a cheapest π^C-fork-compliant leaf path π^L ending in s^L. The *fork-decoupled goal states* are those s where $\mathsf{center}(s)$ is a goal center state, and, for every $F^L \in \mathcal{F}^L$, at least one goal leaf state $s^L \in S^L|_{F^L}$ has a finite price $\mathsf{prices}(s)[s^L] < \infty$. Once a fork-decoupled goal state s is reached, a plan for Π can be extracted by augmenting the center path π^C leading to s with cheapest π^C-fork-compliant *goal leaf paths*, i. e., leaf paths ending in goal leaf states. Observe that this plan is optimal subject to fixing π^C, i. e., the cheapest possible plan for Π when comitting to exactly the center moves π^C.

Say $\pi^C = \langle \mathrm{move}(A, l_1, l_2), \mathrm{move}(B, l_3, l_2), \mathrm{move}(B, l_2, l_3) \rangle$ in our example, traversing the fork-decoupled states s_0, s_1, s_2, s_3. Then $\mathsf{prices}(s_0)[p = l_1] = 0$, $\mathsf{prices}(s_0)[p = A] = 1$, $\mathsf{prices}(s_1)[p = l_2] = 2$, $\mathsf{prices}(s_2)[p = B] = 3$, and $\mathsf{prices}(s_3)[p = l_3] = 4$. To extract a plan for Π from the fork-decoupled goal state s_3, we trace back the compliant leaf path supporting $p = l_3$ and embed it into π^C. The resulting plan loads p onto t_A, moves t_A to l_2, unloads p, moves t_B to l_2, loads p onto t_2, moves t_B to l_3, and unloads p.

The core of GH's construction is the maintenance of pricing functions. For forks, this is simple enough to be described in a few lines within the definition of Θ^ϕ. For star topologies, we need to substantially extend this construction, so we hone in on it in more detail here. We reformulate it in terms of *compliant path graphs*, which capture all possible compliant graphs for a leaf F^L given a center path π^C:

Definition 3 (Fork-Compliant Path Graph) *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path traversing center states $\langle s^C_0, \ldots, s^C_n \rangle$. The π^C-*fork-compliant path graph *for a leaf $F^L \in \mathcal{F}^L$, denoted $\mathit{CompG}^\phi(\pi^C, F^L)$, is the arc-labeled weighted directed graph whose vertices are the* time-stamped leaf states $\{s^L_t \mid s^L \in S^L|_{F^L}, 0 \leq t \leq n\}$*, and whose arcs are:*

- *(i)* $s^L_t \xrightarrow{a^L} s'^L_t$ *with weight $c(a^L)$ whenever $s^L, s'^L \in S^L|_{F^L}$ and $a^L \in A^L|_{F^L}$ such that $\mathsf{pre}(a^L)[F^C] \subseteq s^C_t$, $\mathsf{pre}(a^L)[F^L] \subseteq s^L$, and $s^L[\![a^L]\!] = s'^L$.*
- *(ii)* $s^L_t \xrightarrow{0} s^L_{t+1}$ *with weight 0 for all $s^L \in S^L|_{F^L}$ and $0 \leq t < n$.*

In words, the π^C-fork-compliant path graph includes a copy of the leaf states at every time step $0 \leq t \leq n$ along the center path π^C. Within each t, the graph includes all leaf-state transitions enabled in the respective center state. From each t to $t + 1$, the graph has a 0-cost transition for each leaf state. Consider again the example, and the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$. The π^C-fork-compliant path graph for the package is shown in Figure 2.[1]

[1]Note that $\mathit{CompG}^\phi(\pi^C, F^L)$ contains redundant parts, not reachable from the initial leaf state $I[F^L]$, i. e., $(p = l_1)_0$ in the

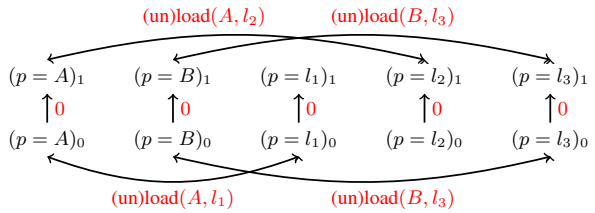


Figure 2: The fork-compliant path graph for $\pi^C = \langle \text{move}(A, l_1, l_2) \rangle$ in our illustrative example.

The π^C-fork-compliant leaf path $\pi^L = \langle \text{load}(A, l_1), \text{unload}(A, l_2) \rangle$ can be embedded by starting at $(p = l_1)_0$, following the arc labeled load(A, l_1) to $(p = A)_0$, following the 0-arc to $(p = A)_1$, and following the arc labeled unload(A, l_2) to $(p = l_2)_1$. The non-compliant leaf path $\pi^L = \langle \text{load}(A, l_1), \text{unload}(A, l_2), \text{load}(A, l_2), \text{load}(A, l_1) \rangle$ cannot be embedded, as (un)load(A, l_2) appears only at $t = 1$, while load(A, l_1) is not available anymore at $t \geq 1$.

Lemma 1 *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path. Let $F^L \in \mathcal{F}^L$, and $s^L \in S^L|_{F^L}$. Then the cost of a cheapest π^C-fork-compliant leaf path π^L ending in s^L equals the cost of a cheapest path from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$.*

Proof: Given a π^C-fork-compliant leaf path $\pi^L = \langle a^L_1, \ldots, a^L_m \rangle$, we can schedule each a^L_i as a (i) arc in CompG$^\phi(\pi^C, F^L)$ at the time step $t(i)$ assigned by the embedding. Connecting the resulting partial paths across time steps using the (ii) arcs, we get a path π from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, whose cost equals that of π^L. Vice versa, given a path π from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, remove the time indices of the vertices on π, and remove the arcs crossing time steps. This yields a π^C-fork-compliant leaf path π^L ending in s^L, whose cost equals that of π. So the π^C-fork-compliant leaf paths ending in s^L are in one-to-one correspondence with the paths from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, showing the claim. □

To prepare our extension in the next section, we now reformulate the fork-decoupled state space Θ^ϕ. Each fork-decoupled state s is a center path $\pi^C(s)$, associated for every leaf $F^L \in \mathcal{F}^L$ with the π^C-fork-compliant path graph CompG$^\phi(\pi^C(s), F^L)$. The fork-decoupled initial state I^ϕ is the empty center path $\pi^C(I^\phi) = \langle\rangle$. The successor states s' of s are exactly the center paths extending $\pi^C(s)$ by one more center action. The fork-decoupled goal states are those s where $\pi^C(s)$ ends in a center goal state, and for every $F^L \in \mathcal{F}^L$ there exists a goal leaf state $s^L \in S^L|_{F^L}$ s.t. $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in CompG$^\phi(\pi^C(s), F^L)$.

This formulation is different from GH's, but is equivalent given Lemma 1: Instead of maintaining the pricing functions prices(s) explicitly listing the costs of cheapest π^C-fork-compliant paths, we maintain the fork-compliant path graphs CompG$^\phi(\pi^C(s), F^L)$, from which these same costs can be obtained in terms of standard graph distance.

figure. This is just to keep the definition simple, in practice one can maintain only the reachable part of CompG$^\phi(\pi^C, F^L)$.

Star-Topology Decoupling

We now extend the concepts of compliant paths, and compliant path graphs, to handle star topologies instead of forks. A star topology is one where the center may interact arbitrarily with each leaf, and even effect-effect dependencies across leaves are allowed so long as they also affect the center:

Definition 4 (Star Factoring) *Let Π be an FDR task, and let $\mathcal{F}$ be a factoring. The* support-interaction graph *SuppIG$(\mathcal{F})$ of $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $(F \to F')$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $(v \to v')$ is an arc in SuppG. $\mathcal{F}$ is a* star factoring *if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. the following two conditions hold:*

(1) The arcs in SuppIG$(\mathcal{F})$ are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.

(2) For every action a, if there exist $F^L_1, F^L_2 \in \mathcal{F} \setminus \{F^C\}$ such that $F^L_1 \neq F^L_2$ and $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_1 \neq \emptyset$ as well as $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_2 \neq \emptyset$, then $\mathcal{V}(\mathsf{eff}(a)) \cap F^C \neq \emptyset$.

F^C is the center *of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are* leaves.

A star factoring $\mathcal{F}$ is strict *if the arcs in IG$(\mathcal{F})$ are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.*

We cannot characterize star factorings in terms of just the causal graph, because the effect-effect arcs in that graph are inserted for all variable pairs in the effect: If there is an arc between two leaves, we cannot distinguish whether or not the same action also affects the center. In contrast, in a strict star factoring every action affects at most one leaf, which *can* be characterized in terms of just the causal graph. Hence strict star factorings are interesting from a practical perspective, allowing different/simpler factoring strategies.

Obviously, Definition 4 generalizes Definition 1. Perhaps less obvious is how far this generalization carries. As pointed out, an FDR task has a fork factoring iff its causal graph has more than one SCC. In contrast, *every* FDR task has a star factoring. In fact, any partition of the variables into two non-empty subsets is a star factoring: Calling one half of the variables the "center", and the other the "leaf", we have a (strict) star factoring, as Definition 4 does not apply any restrictions if there is a single leaf only.[2] That said, it is not clear whether single-leaf factorings are useful in practice. We get back to this when discussing our experiments.

To illustrate, consider what we will refer to as the *no-empty example*, where we forbid "empty truck moves". This is as before, except the precondition of move(x, y, z) is no longer $\{t_x = y\}$, but is $\{t_x = y, p = x\}$: A truck can only move if the package is currently inside it. The causal graph arcs now are not only $(t_A \to p)$ and $(t_B \to p)$ as before, but also $(p \to t_A)$ and $(p \to t_B)$. Hence there exists no fork factoring. But our previous factoring with $F^C = \{t_A, t_B\}$ and the single leaf $F^L = \{p\}$ is now a strict star factoring.

[2] Observe also that Definition 4 (2) can always be enforced wlog, simply by introducing redundant effects on F^C. However, actions affecting F^C are those our search must branch over, so this is just another way of saying "cross-leaf effects can be tackled by centrally branching over the respective actions". Doing so weakens the decoupling obtained, and for now we do not consider it.

To define compliance, we will be using the same terminology as before, i. e. center actions/paths and leaf actions/paths. These concepts are (mostly) defined as before, however their behavior is more complicated now.

The notions of center/leaf states, and of goal center/leaf states, remain the same. The center actions A^C are still all those actions affecting the center, and the leaf actions $A^L|_{F^L}$ for $F^L \in \mathcal{F}^L$ are still all those actions affecting F^L. However, *A^C and $A^L|_{F^L}$ are no longer disjoint*, as the same action may affect both A^C and $A^L|_{F^L}$.

A leaf path still is a sequence of leaf actions applicable to I when ignoring all center preconditions. The notion of center path changes, as now there may be leaf preconditions; we ignore these, i. e., *a center path is now a sequence of center actions applicable to I when ignoring all leaf preconditions*. As center and leaf paths may overlap, we need to clarify where to account for the cost of the shared actions. Our search, as we explain in a moment, views all A^C actions as part of the center, so we account for their costs there. To that end, *the cost of a leaf path π^L now is the summed-up cost of its $(A^L|_{F^L} \setminus A^C)$ actions*. By construction, these actions do not affect any factor other than F^L itself.

We will define star-compliant paths, and star-compliant path graphs $\mathit{CompG}^\sigma(\pi^C(s), F^L)$, below. For an overview before delving into these details, consider first the *star-decoupled state space* Θ^σ. A *star-decoupled state* s is a center path $\pi^C(s)$ associated for every leaf $F^L \in \mathcal{F}^L$ with the π^C-star-compliant path graph $\mathit{CompG}^\sigma(\pi^C(s), F^L)$. The *star-decoupled initial state* I^σ is the empty center path $\pi^C(I^\sigma) = \langle\rangle$. The *star-decoupled goal states* are those s where $\pi^C(s)$ ends in a center goal state, and for every $F^L \in \mathcal{F}^L$ there exists a goal leaf state $s^L \in S^L|_{F^L}$ s.t. $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$.

The definition of successor states s' of a star-decoupled state s changes more substantially. For forks, these simply were all center paths extending $\pi^C(s)$ by one more center action a^C. This worked due to the absence of leaf preconditions: by definition of "center path", $\mathsf{pre}(a^C)$ was satisfied at the end of $\pi^C(s)$. Given a star factoring instead, the successor states s' still result from extending $\pi^C(s)$ by one more center action a^C, but restricted to those a^C whose leaf preconditions can be satisfied at the end of $\pi^C(s)$. Namely, we require that, for every $F^L \in \mathcal{F}^L$, there exists $s^L \in S^L|_{F^L}$ such that $\mathsf{pre}(a^C)[F^L] \subseteq s^L$ and $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$.

In our original example, the star-decoupled initial state has two successors, from $\mathrm{move}(A, l_1, l_2)$ and $\mathrm{move}(B, l_3, l_2)$. In the no-empty example, only $\mathrm{move}(A, l_1, l_2)$ is present: Its precondition $p = A$ is reachable from $I[F^L]_0$ given the empty center path. But that is not so for the precondition $p = B$ of $\mathrm{move}(B, l_3, l_2)$.

Like for forks, we first identify a notion of compliant paths which captures how plans for the input task Π can be understood as center paths augmented with compliant leaf paths; and we then capture compliant paths in terms of compliant path graphs. However, the notion of "compliance" is now quite a bit more complicated. Given center path π^C and leaf path π^L, we require that (1) the sub-sequences of shared actions in π_L and π_C coincide, and (2) in between, we can schedule π^L at monotonically increasing points alongside π^C s.t. (2a) the center precondition of each leaf action holds in the respective center state and (2b) the F^L precondition of each center action holds in the respective leaf state.

Definition 5 (Star-Compliant Path) *Let Π be an FDR task, $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path. Let π^L be a leaf path for $F^L \in \mathcal{F}^L$. We say that π^L* star-complies *with π^C, also π^L is π^C-*star-compliant*, if the following two conditions hold:*

(1) The sub-sequence of A^C actions in π^L coincides with the sub-sequence of $A^L|_{F^L}$ actions in π^C.

(2) Assume, for ease of notation, dummy $A^C \cap A^L|_{F^L}$ actions added to start and end in each of π^C and π^L. For every pair $\langle a, a'\rangle$ of subsequent $A^C \cap A^L|_{F^L}$ actions in π^L and π^C, there exists an embedding at $\langle a, a'\rangle$.

Here, denote the sub-sequence of π^C between a and a' (not including a and a' themselves) by $\langle a^C_1, \dots, a^C_n\rangle$, and the F^C states it traverses by $\langle s^C_0, \dots, s^C_n\rangle$. Denote the sub-sequence of π^L between a and a' by $\langle a^L_1, \dots, a^L_m\rangle$, and the F^L states it traverses by $\langle s^L_0, \dots, s^L_m\rangle$. An embedding at $\langle a, a'\rangle$ *then is a monotonically increasing function $t : \{1, \dots, m\} \mapsto \{0, \dots, n\}$ so that both:*

(a) For every $i \in \{1, \dots, m\}$, $\mathsf{pre}(a^L_i)[F^C] \subseteq s^C_{t(i)}$.

(b) For every $t \in \{1, \dots, n\}$, $\mathsf{pre}(a^C_t)[F^L] \subseteq s^L_{i(t)}$ where $i(t) := \max\{i \mid t(i) < t\}$ (with $\max \emptyset := 0$).

To illustrate this, consider our no-empty example, the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2)\rangle$, and the leaf path $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2)\rangle$. Definition 5 (1) is trivially fulfilled because the sub-sequences it refers to are both empty. For Definition 5 (2), we assume dummy shared actions, $\pi^C = \langle a, \mathrm{move}(A, l_1, l_2), a'\rangle$ and $\pi^L = \langle a, \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2), a'\rangle$. The only pair of subsequent shared actions then is $\langle a, a'\rangle$. We need to find an embedding $t : \{1, 2\} \mapsto \{0, 1\}$ of $\langle a^L_1 = \mathrm{load}(A, l_1), a^L_2 = \mathrm{unload}(A, l_2)\rangle$ traversing F^L states $\langle s^L_0 = \{p = l_1\}, s^L_1 = \{p = A\}, s^L_2 = \{p = l_2\}\rangle$, into $\langle a^C_1 = \mathrm{move}(A, l_1, l_2)\rangle$ traversing F^C states $\langle s^C_0 = \{t_A = l_1\}, s^C_1 = \{t_A = l_2\}\rangle$. Given their center preconditions, we must schedule $\mathrm{load}(A, l_1)$ before $\mathrm{move}(A, l_1, l_2)$ and $\mathrm{unload}(A, l_2)$ behind $\mathrm{move}(A, l_1, l_2)$. So the only possibility is $t(1) := 0, t(2) := 1$. Indeed, that is an embedding: For Definition 5 (2a), $\mathsf{pre}(a^L_1)[F^C] = \{t_A = l_1\} \subseteq s^C_{t(1)} = s^C_0$, and $\mathsf{pre}(a^L_2)[F^C] = \{t_A = l_2\} \subseteq s^C_{t(2)} = s^C_1$. For Definition 5 (2b), $i(1) = \max\{i \mid t(i) < 1\} = 1$ because $t(1) = 0$ i. e. $a^L_1 = \mathrm{load}(A, l_1)$ is scheduled before $a^C_1 = \mathrm{move}(A, l_1, l_2)$. So $\mathsf{pre}(a^C_1)[F^L] = \{p = A\} \subseteq s^L_{i(1)} = s^L_1$ as required.

Despite the much more complex definition, the correspondence of compliant paths to plans for the original input planning task Π is as easily seen as for fork factorings. Say π is a plan for Π. The sub-sequence π^C of center actions in π is a center path. For a leaf $F^L \in \mathcal{F}^L$, the sub-sequence π^L of $A^L|_{F^L}$ actions in π is a leaf path. The sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^L coincides by construction with the sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^C, so we fulfill Definition 5 (1). Furthermore, between any pair of subsequent shared actions, all F^C preconditions of π^L, and all F^L

preconditions of π^C, must be satisfied because π is a plan, so we can read off an embedding fulfilling Definition 5 (2), and π^L is π^C-star-compliant. Vice versa, say center path π^C ends in a goal center state, and can be augmented for every $F^L \in \mathcal{F}^L$ with a π^C-star-compliant leaf path π^L ending in a goal leaf state. Note that, if an action a affects more than one leaf, by the definition of star factorings a must also affect the center, so by Definition 5 (1) the sub-sequences of such actions are synchronized via π^C: They must be identical for every leaf involved, and correspond to the same action occurences in π^C. Hence, sequencing all actions in π^C and every π^L according to the embeddings, we get an executable action sequence π achieving the overall goal in Π. Recall, finally, that we defined the cost of leaf paths to account only for those actions affecting just the leaf in question and nothing else. So, in both directions above, the cost of π equals the summed-up cost of the center path and leaf paths. We get that *the plans for* Π *are in one-to-one correspondence with center paths augmented with compliant leaf paths.*

We finally show how to capture π^C-star-compliant paths in terms of the weighted graphs $\mathit{CompG}^\sigma(\pi^C(s), F^L)$ we maintain alongside search over center paths in Θ^σ:

Definition 6 (Star-Compliant Path Graph) *Let* Π *be an FDR task,* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$*, and* $\pi^C = \langle a_1^C, \dots, a_n^C \rangle$ *a center path traversing center states* $\langle s_0^C, \dots, s_n^C \rangle$*. The* π^C-star-compliant path graph *for a leaf* $F^L \in \mathcal{F}^L$*, denoted* $\mathit{CompG}^\sigma(\pi^C, F^L)$*, is the arc-labeled weigthed directed graph whose vertices are* $\{s_t^L \mid s^L \in S^L|_{F^L}, 0 \leq t \leq n\}$*, and whose arcs are as follows:*

- *(i)* $s_t^L \xrightarrow{a^L} {s'}_t^L$ *with weight* $c(a^L)$ *whenever* $s^L, {s'}^L \in S^L|_{F^L}$ *and* $a^L \in A^L|_{F^L} \setminus A^C$ *s.t.* $\mathsf{pre}(a^L)[F^C] \subseteq s_t^C$*,* $\mathsf{pre}(a^L)[F^L] \subseteq s^L$*, and* $s^L[\![a^L]\!] = {s'}^L$*.*
- *(ii)* $s_t^L \xrightarrow{0} {s'}_{t+1}^L$ *with weight* 0 *whenever* $s^L, {s'}^L \in S^L|_{F^L}$ *s.t.* $\mathsf{pre}(a_t^C)[F^L] \subseteq s^L$ *and* $s^L[\![a_t^C]\!] = {s'}^L$*.*

Item (i) is a benign change of Definition 3. Exactly as before, within each time step t the arcs correspond to those leaf-only actions whose center precondition is enabled at t. The only difference is that we need to explicitly exclude actions a^L affecting also the center (which for fork factorings cannot happen anyway). Item (ii) differs more substantially. Intuitively, whereas for fork factorings the $t \to t+1$ arcs simply stated that whichever leaf state we achieved before will survive the center action a_t^C (which could neither rely on, nor affect, the leaf), these arcs now state that the surviving leaf states are only those which comply with a_t^C's precondition, and will be mapped to possibly different leaf states by a_t^C's effect. Note that, if a_t^C has no precondition on F^L, then all leaf states survive, and if a_t^C has no effect on F^L, then all leaf states remain the same at $t+1$. If both is the case, then we are back to exactly the arcs (ii) in Definition 3.

For our no-empty task and $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$, the π^C-star-compliant path graph is as shown in Figure 3.

Note the (only) difference to Figure 2: From time 0 to time 1, the only (ii) arc we have now is that from $(p = A)_0$ to $(p = A)_1$. This is because $\mathrm{move}(A, l_1, l_2)$ now has precondition $p = A$, so all other values of p do not comply with the center action being applied at this time step.

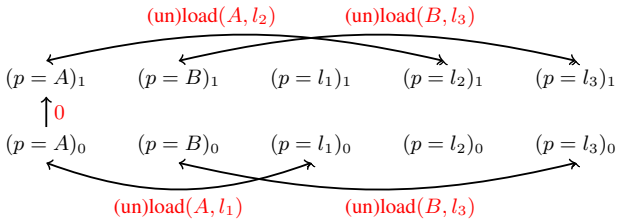


Figure 3: The star-compliant path graph for $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$ in our no-empty example.

Lemma 2 *Let* Π *be an FDR task,* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$*, and* π^C *a center path. Let* $F^L \in \mathcal{F}^L$*, and* $s^L \in S^L|_{F^L}$*. Then the cost of a cheapest* π^C*-star-compliant leaf path* π^L *ending in* s^L *equals the cost of a cheapest path from* $I[F^L]_0$ *to* s_n^L *in* $\mathit{CompG}^\sigma(\pi^C, F^L)$*.*

Proof: Consider first a π^C-star-compliant leaf path $\pi^L = \langle a_1^L, \dots, a_m^L \rangle$ for leaf $F^L \in \mathcal{F}^L$. By Definition 5 (1), the $A^C \cap A^L|_{F^L}$ sub-sequences in π^C and π^L coincide. Scheduling these at the respective time steps $t \to t+1$ in $\mathit{CompG}^\sigma(\pi^C, F^L)$, corresponding (ii) arcs must be present by construction. In between each pair of such actions, by Definition 5 we have embeddings t mapping the respective sub-sequence of π^L to that of π^C. Schedule each π^L action at its time step assigned by t. Then corresponding (i) arcs must be present by Definition 5 (2a). By Definition 5 (2b), if a π^C action here relies on an F^L precondition, then the corresponding leaf state satisfies that precondition so we have the necessary (ii) arc. Overall, we obtain a path π from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$, and clearly the cost of π accounts exactly for the F^L-only actions on π^L, as needed.

Vice versa, consider any path π from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$. Removing the time indices of the vertices on π, and removing those (ii) arcs $s_t^L \xrightarrow{0} {s'}_{t+1}^L$ where $s_t^L = {s'}_{t+1}^L$, clearly we obtain a π^C-star-compliant leaf path π^L ending in s^L, whose cost equals that of π.

So the π^C-star-compliant leaf paths ending in s^L are in one-to-one correspondence with the paths from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$, showing the claim. □

Overall, goal paths in the star-decoupled state space Θ^σ correspond to center goal paths augmented with star-compliant leaf goal paths, which correspond to plans for the original planning task Π, of the same cost. So (optimal) search in Θ^σ is a form of (optimal) planning for Π.

Heuristic Search

GH show how standard classical planning heuristics, and standard search algorithms, can be applied to fork-decoupled search. All these concepts remain intact for star topologies; one issue requires non-trivial attention. (For space reasons, we omit details and give a summary only.)

A *heuristic* for Θ^σ is a function from star-decoupled states into $\mathbb{R}^{0+} \cup \{\infty\}$. The *star-perfect* heuristic, $h^{\sigma *}$, assigns to any s the minimum cost for completing s, i. e., reaching a star-decoupled goal state plus embedding compliant goal leaf paths. A heuristic h is *star-admissible* if $h \leq h^{\sigma *}$.

Given an FDR task Π and a star-decoupled state s, one can construct an FDR task $\Pi^\sigma(s)$ so that computing any admissible heuristic h on $\Pi^\sigma(s)$ delivers a star-admissible heuristic

value for s. $\Pi^\sigma(s)$ is like Π except for the initial state (center state of s, initial state for the leaves), and that new actions are added allowing to achieve each leaf state at its price in s.

Star-decoupled goal states are, as GH put it, *goal states with price tags*: Their path cost accounts only for the center moves, and we still have to pay the price for the goal leaf paths. In particular, $h^{\sigma *}$ is *not* 0 on star-decoupled goal states. We can obtain a standard structure Θ' for search as follows. Introduce a new goal state G. Give every star-decoupled goal state s an outgoing transition to G whose cost equals the summed-up cost of cheapest compliant goal leaf paths in s. Given a heuristic h for Θ^σ, set $h(G) := 0$.

The generalization to star-decoupling incurs one important issue, not present in the special case of fork factorings. If center moves require preconditions on leaves, then we should "buy" these preconditions immediately, putting their price into the path cost g, because otherwise we lose information during the search. For illustration, in our no-empty example, say the goal is $t_A = l_2$ instead of $p = l_3$, and consider the star-decoupled state s after applying move(A, l_1, l_2). Then $t_A = l_2$ is true, $g = 1$, and h^* on the compiled FDR task $\Pi^\sigma(s)$ returns 0 because the goal is already true. But $h^{\sigma *}(s) = 1$ and the actual cost of the plan is 2: We still need to pay the price for the precondition $p = A$ of move(A, l_1, l_2). This is not captured in $\Pi^\sigma(s)$ because it is needed *prior* to s only. The solution is to perceive this "price" as a "cost" already committed to. In our modified structure Θ', when applying a center action a^C to star-decoupled state s, we set the local cost of a^C (its cost specifically at this particular position in Θ') to $\mathsf{cost}(a_t^C) + \sum_{F^L} g(F^L)$. Here, $g(F^L)$ is the minimum over the price in s of those $s^L \in S^L|_{F^L}$ that satisfy a^C's precondition. Intuitively, to apply a^C, we must first buy its leaf preconditions. To reflect that $g(F^L)$ has already been paid, the respective (ii) arcs in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$ are assigned weight $-g(F^L)$. In our example above, the path cost in s is $g = 2$ giving us the correct $g + h = 2$. The "0" arc in Figure 3 is assigned weight -1, so that the overall cost of the compliant path for p will be 0 (as the only action we need to use has already been paid for by the center move).

Any (optimal) standard heuristic search algorithm X on Θ' yields an optimal heuristic search algorithm for Θ^σ, which we denote *Star-Decoupled X (SDX)*.

Experiments

Our implementation is in FD (Helmert 2006), extending that for fork decoupling by GH. We ran all international planning competition (IPC) STRIPS benchmarks ('98–'14), on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

Our experiments are preliminary in that we perform only a very limited exploration of factoring strategies. Factoring strategy design for star topologies is, in contrast to fork topologies, quite challenging. The space of star factorings includes arbitrary two-subset partitions of single-SCC causal graphs, where fork factorings do not exist at all. Even for the simplest possible optimization criterion, maximizing the number of leaves in a strict star factoring, finding an optimal factoring is **NP**-complete (this follows by a straightforward reduction from Maximum Independent Set (Garey and Johnson 1979)). An additional complication is that leaves may be "frozen": As we need to branch over all actions affecting the center, for a leaf F^L to yield a state space size reduction there must be at least one action affecting *only* F^L (not affecting the center). For example, in IPC Visit-All, while the robot position may naturally be viewed as the "center" and each "visited" variable as a leaf, every leaf-moving action also affects the center so nothing is gained.

We shun this complexity here, leaving its comprehensive exploration to future work, and instead design only two simple strict-star factoring strategies by direct extension of GH's fork factoring strategy. That strategy works as follows.

Denote by $\mathcal{F}^{\mathrm{SCC}}$ the factoring whose factors are the SCCs of *CG*. View the interaction graph $\mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$ over these SCCs as a DAG where the root SCCs are at the top and the leaf SCCs at the bottom. Consider the "horizontal lines" $\{T, B\}$ (top, bottom) through that DAG, i. e., the partitions of V where every $F \in \mathcal{F}^{\mathrm{SCC}}$ is fully contained in either of T or B, and where the only arc in $\mathit{IG}(\{T, B\})$ is $(T \to B)$. Let $\mathcal{W}$ be the set of weakly connected components of $\mathcal{F}^{\mathrm{SCC}}$ within B. Then a fork factoring $\mathcal{F}$ is obtained by setting $F^C := T$ and $\mathcal{F}^L := \mathcal{W}$. Any fork factoring can be obtained in this manner, except the *redundant* ones where some $F^L \in \mathcal{F}^L$ contains several weakly connected components.

GH's strategy moves the horizontal line upwards, from leaves to roots in $\mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$, in a greedy fashion, thereby generating a sequence $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of fork factorings. They select the factoring $\mathcal{F}_i$ whose number of leaf factors is maximal, and whose index i is minimal among these factorings. The rationale behind this is to maximize the number of leaves (the amount of conditional independence) while keeping these as small as possible (reducing the runtime overhead). If $k = 0$ (no horizontal line exists i. e. *CG* is a single SCC), or $\mathcal{F}_i$ has a single leaf only, then GH *abstain* from solving the input task. The rationale is that, in GH's experiments, single-leaf factorings hardly ever payed off.

We design two new (non-fork) strategies, *inverted forks* and *X-shape*. The former is exactly GH's strategy but inverting the direction of the arcs in the causal graph. The latter runs GH's fork factoring first, and thereafter runs inverted forks on the fork center component F^C. If an inverted-fork leaf F^L has an outgoing arc into a fork leaf, then F^L is included into F^C. We abstain if no factoring exists or if the selected factoring has a single leaf only. Note that this still abstains on single-SCC causal graphs, and that frozen leaves cannot occur as neither forks nor inverted forks allow leaf-affecting actions to affect the center. The strategies take negligible runtime (rounded to 0.00 in most cases, much faster than FD's pre-processes in the few other cases).

For optimal planning (with LM-cut (Helmert and Domshlak 2009)), while GH reported dramatic gains using fork factoring, our new factoring strategies do not improve much upon these gains. Inverted forks do sometimes help, most notably in Satellite where Star-Decoupled A* (SDA*) with inverted fork factoring solves 3 more instances than each of A* and fork-factoring SDA*, reducing evaluations on commonly solved instances by up to two orders of magnitude.

	(A): Coverage												(B): Evaluations: Improvement factor relative to GBFS												(C): Runtime: Improvement factor relative to GBFS											
	X-shape						inverted fork			fork			X-shape																							
													no preferred operators					with preferred operators							no preferred operators					with preferred operators					
		npo		po				npo	po		npo	po		SDGBFS				SDGBFS				LAMA				SDGBFS				SDGBFS				LAMA		
	#	base	sd	base	sd	LAMA	#	sd	sd	#	sd	sd	#	∑D	GM	max	#	∑D	GM	max	∑D	GM	max	#	∑D	GM	max	#	∑D	GM	max	∑D	GM	max
Childsna	20	0	0	3	+1	-3	20	0	+1	0			0				0							0				0						
Depots	22	14	-1	18	-1	+3	22	-1	-1	0			12	0.5	1.1	14.0	17	1.5	2.1	54.5	3.4	2.3	299	10	0.3	0.3	5.7	14	1.0	0.7	22.5	0.8	1.3	91.0
Driver	20	18	+1	20	-2	0	0			20	+1	-2	17	1.2	2.2	8.3	18	0.1	0.9	4.0	0.7	0.9	21.0	5	0.3	0.4	3.3	12	0.0	0.2	1.0	0.5	0.6	2.1
Elev08	30	30	0	30	0	0	30	0	0	0			30	5.5	5.4	18.1	30	4.2	3.5	12.1	1.0	1.1	3.8	24	2.0	1.5	4.0	20	1.4	1.1	2.0	0.6	0.6	1.4
Elev11	20	18	+2	20	0	0	20	+2	0	0			18	12.6	7.5	46.8	20	11.5	5.6	37.3	1.8	1.1	4.3	18	4.8	2.7	18.5	20	4.7	1.8	13.0	1.3	0.7	2.8
Floor11	20	6	-5	6	-4	0	20	-5	-4	0			1	0.4	0.4	0.4	2	0.7	0.7	0.7	0.8	0.8	1.0	1	0.1	0.1	0.1	2	0.1	0.1	0.1	0.2	0.2	0.3
Floor14	20	2	-2	2	0	0	20	-2	0	0			0				2	0.6	0.6	0.7	0.9	0.9	1.1	0				2	0.1	0.1	0.1	0.2	0.2	0.3
Log00	28	28	0	28	0	0	28	0	0	28	0	0	28	11.8	9.4	18.8	28	5.2	4.7	6.8	0.8	0.9	1.0	0				0						
Log98	35	26	+9	35	0	0	35	+9	0	35	+9	0	26	37.3	12.7	60.0	35	49.4	3.5	177	9.0	1.1	28.1	16	19.0	6.7	34.9	24	19.1	1.5	65.8	4.3	0.8	18.0
Mico	145	145	0	145	0	0	0			145	0	0	145	2.4	2.2	4.4	145	4.3	3.6	8.4	1.0	1.0	1.8	26	1.1	1.1	1.6	45	1.0	1.0	1.2	0.5	0.6	0.9
Mystery	2	0	0	1	0	0	2	0	0	0			0				1	0.9	0.9	0.9	1.4	1.3	1.3	0				1	0.6	0.6	0.6	1.0	1.0	1.0
NoMy	20	9	+10	10	+9	+3	0			20	+10	+9	9	141	34.0	1250	10	43K	33.5	15M	1697	4.5	135K	5	2.6	2.2	5.9	8	1033	3.8	12K	468	2.1	10K
Pathw	30	11	+2	20	0	+4	0			29	+2	0	11	12.7	3.8	26.5	20	1.4	1.5	1.8	0.6	0.7	1.2	7	3.5	1.3	10.7	10	0.4	0.6	1.0	0.3	0.4	1.0
PSR	3	3	0	3	0	0	0			3	0	0	3	1.6	1.7	1.9	3	1.3	1.2	1.6	1.0	1.0	1.0	0				0						
Rovers	40	23	-1	40	0	0	38	+5	-1	40	-1	0	22	1.2	1.7	12.3	40	1.3	1.7	3.1	0.7	0.8	1.8	12	0.4	0.5	1.7	22	0.6	0.7	1.0	0.3	0.4	0.8
Satell	36	30	+3	36	0	0	34	+1	0	36	+3	0	30	2.7	2.0	38.7	36	1.8	1.3	6.8	0.4	0.5	1.5	19	1.1	1.8	21.9	25	1.9	1.0	3.9	0.1	0.3	0.8
TPP	29	22	+1	29	0	0	26	-5	0	27	+1	0	22	3183	454	31K	29	0.1	3.4	41.9	0.8	0.8	1.8	14	95.8	52.7	987	17	0.0	0.2	1.3	0.5	0.6	0.9
Transp08	30	16	+14	28	+2	+2	30	+14	+2	0			16	693	35.7	2226	28	305	21.0	4360	1.0	1.0	3.3	10	135	17.6	262	22	18.0	2.0	156	0.6	0.7	1.7
Transp11	20	0	+20	11	+9	+7	20	+20	+9	0			0				11	500	114	4360	1.0	1.1	8.9	0				11	19.0	5.2	141	0.7	0.7	4.7
Transp14	20	0	+20	6	+14	+9	20	+20	+14	0			0				6	168	130	407	1.1	1.1	6.9	0				6	8.7	7.4	23.9	0.6	0.8	4.4
Wood08	26	26	0	26	0	0	25	0	0	25	0	0	26	0.4	1.2	121	26	1.2	2.3	111	7.8	14.8	145	20	0.1	0.2	2.1	22	0.2	0.3	7.9	2.1	2.4	6.3
Wood11	19	18	0	19	0	0	18	+1	0	18	+1	0	2	0.4	0.5	4.3	19	1.2	1.2	3.0	11.8	15.8	53.8	16	0.0	0.1	0.5	18	0.0	0.1	0.2	2.1	2.1	4.9
Zeno	20	20	0	20	0	0	18	0	0	20	0	0	20	29.3	12.1	103	20	4.6	3.0	6.3	0.8	0.8	1.3	7	4.2	3.8	10.6	7	0.9	0.8	1.1	0.4	0.4	0.7
∑	655	465	+73	556	+28	+25	426	+59	+20	446	+26	+7																							

Table 1: Results in satisficing planning with h^{FF}. Each table section fixes a factoring strategy (2nd row from top), and uses *only the instances which that strategy does not abstain on*; of these, (B) uses the subset of commonly solved ones (with vs. without preferred operators), (C) the same but excluding ones commonly solved in ≤ 0.1 seconds. The respective numbers of underlying instances are shown in columns "#". GBFS: Greedy Best-First Search; npo: no preferred ops; po: with preferred ops (FD's dual-queue search); base: baseline (GBFS on standard state space); sd: star-decoupled base; $\sum$D: factor over the per-domain sums; GM/max: Geometric mean/maximum over the per-instance factors; K: thousand; M: million.

But such cases are rare. It remains future work to explore more sophisticated star-factoring strategies for optimal planning. Here, we focus on satisficing planning where even our current simple factoring strategies yield good results.

Consider Table 1. X-shape factoring abstains much less than each of the "base strategies", forks and inverted forks, on its own. X-shape factoring substantially improves coverage, overall and in several domains, without preferred operators, and does so in NoMystery and Transport with preferred operators. It even beats LAMA (Richter and Westphal 2010) overall, thanks to excelling in Transport. Indeed, with preferred operators, Transport is solved almost instantly, with a maximum (average) of 69 (36.1) state evaluations and 14.7 (2.7) seconds runtime. Without preferred operators, the average is 1236.8 evaluations and 127.1 seconds. Substantial reductions of evaluations, with corresponding smaller but still significant reductions of runtime, are obtained also in Elevators, Logistics, NoMystery, Pathways, TPP, and Zenotravel, plus smaller improvements in various other cases.

The strength of X-shape factoring is inherited from fork factoring in NoMystery and Pathways, where inverted fork factoring abstains. It stems from inverted fork factoring, and is thus entirely thanks to the new techniques introduced herein, in Elevators and Transport where fork factoring abstains. For Logistics, TPP, and Zenotravel, where neither base strategy abstains, our current configuration of X-shape factoring uses exactly the fork factorings, because the fork strategy is run first and thus given a "preference". To illustrate, in Logistics, the fork makes each package a leaf. The inverted fork would make the trucks leaves, but they have outgoing arcs to the packages so are re-included into the center. Symmetrically, if the X-shape ran the inverted fork strategy first, it would end up with exactly the inverted fork factorings. In Logistics98, these have a consistent advantage ($\sum$D evaluations improvement factor over forks is 6.4 with preferred operators). In Logistics00 they have a consistent small disadvantage ($\sum$D factor 0.9). In TPP and Zenotravel, there is significant per-instance variance, up to 14 times worse respectively 537 times better in TPP, and up to 2 times worse respectively 5 times better in Zenotravel. So there may be room for improvement by combining forks and inverted forks in a less simplistic manner.

As we are mainly interested in speed, the above considers uniform action costs. The results for non-uniform action costs are very similar, except in Elevators where both the baseline and X-shape factoring get worse, in Transport where only the baseline gets worse, and in Woodworking where only X-shape factoring without preferred operators gets worse. The overall coverage gain is then 49 without preferred operators, and 35 with preferred operators.

Conclusion

For our current simple factoring strategies, the upshot from our experiments is that the main empirical advantage of star topologies over forks (as far as IPC benchmarks are concerned) stems from the Transport domain, which this technique "kills" completely. The most important research question in planning remains the exploration of factoring strategies. We need methods able to identify sophisticated star topologies, and we need to understand the strengths of different factorings as a function of problem structure.

Beyond planning, an interesting opportunity is to apply star-topology decoupling to star-topology systems. Verification & synthesis appear especially relevant, possibly with adversarial or probabilistic extensions to star-decoupling. Some classical puzzles may also be amenable; e. g., in the Peg Solitaire game, one can take the "center" to be the middle of the board and the "leaves" to be its peripheral parts.

Acknowledgments. We thank the anonymous reviewers, whose comments helped to improve the paper.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In Gottlob, G., ed., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 929–935. Acapulco, Mexico: Morgan Kaufmann.

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

Brafman, R., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research* 18:315–349.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In Gil, Y., and Mooney, R. J., eds., *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, 809–814. Boston, Massachusetts, USA: AAAI Press.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35. AAAI Press.

Brafman, R., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 65–72. AAAI Press.

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

Gnad, D., and Hoffmann, J. 2015. Beating lm-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J. 2011. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research* 41:155–229.

Jonsson, P., and Bäckström, C. 1995. Incremental planning. In *European Workshop on Planning*.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored planning using decomposition trees. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1942–1947. Hyderabad, India: Morgan Kaufmann.

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

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Wang, D., and Williams, B. C. 2015. tburton: A divide and conquer temporal planner. In Bonet, B., and Koenig, S., eds., *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 3409–3417. AAAI Press.

Goal Recognition Design With Non-Observable Actions

Sarah Keren and **Avigdor Gal**
{sarahn@tx,avigal@ie}.technion.ac.il
Technion — Israel Institute of Technology

Erez Karpas
karpase@csail.mit.edu
Massachusetts Institute of Technology

Abstract

Goal recognition design involves the offline analysis of goal recognition models by formulating measures that assess the ability to perform goal recognition within a model and finding efficient ways to compute and optimize them. In this work we extend earlier work in goal recognition design to partially observable settings. Partial observability is relevant to goal recognition applications such as assisted cognition and security that suffer from reduced observability due to sensor malfunction, deliberate sabotage, or lack of sufficient budget. We relax the full observability assumption by offering a new generalized model for goal recognition design with non-observable actions. In particular we redefine the *worst case distinctiveness* (*wcd*) measure to represent the maximal number of steps an agent can take in a system before the observed portion of his trajectory reveals his objective. We present a method for calculating the *wcd* based on novel compilations to classical planning and propose a method to improve the design. Our empirical evaluation shows the proposed solutions to be effective in computing and improving the *wcd*.

Introduction

Goal recognition design (*grd*) (Keren, Gal, and Karpas 2014; 2015) involves the offline analysis of goal recognition models, interchangeably called in the literature plan recognition (Pattison and Long 2011; Kautz and Allen 1986; Cohen, Perrault, and Allen 1981; Lesh and Etzioni 1995; Ramirez and Geffner 2009; Agotnes 2010; Hong 2001), by formulating measures that assess the ability to perform goal recognition within a model and finding efficient ways to compute and optimize them.

Goal recognition design is applicable to any domain for which quickly performing goal recognition is essential and in which the model design can be controlled. In particular goal recognition design is relevant to goal and plan recognition applications such as assisted cognition (Kautz et al. 2003) and security (Jarvis, Lunt, and Myers 2004; Kaluza, Kaminka, and Tambe 2011; Boddy et al. 2005) that suffer from reduced observability due to sensor malfunction, deliberate sabotage, or lack of sufficient budget. In a safe home setting, for example, reduced coverage means less control over access to sensitive areas such as a hot oven. For intrusion detection applications, malfunctioning sensors may result in an unobserved path to sensitive zones.

Earlier works on goal recognition design (Keren, Gal, and Karpas 2014; 2015) fail to provide support for goal recognition with reduced observability due to its reliance on the assumption that the model being analyzed is fully observable. In this work we relax the full observability assumption and offer innovative tools for a goal recognition design analysis that accounts for a model with non-observable actions and a design process that involves sensor placement.

The set of actions in the *partially observable* setting is partitioned into observable and non-observable actions, reflecting for example a partial sensor coverage. The proposed analysis of observations relies on the partial incoming stream of observations. A key feature of this setting is that it supports a scenario where the system has no information regarding the actions of the agents beyond what is observed. Therefore, in the absence of an observation, the system cannot differentiate unobserved actions from idleness of an agent. An example of such a scenario can be found in Real Time Location Systems (RTLS) where the last known location of an agent is taken as its current position.

The *partially observable* setting provides three extensions to the goal recognition design state-of-the-art. First, we support a partially observable model by defining an observation sequence that contains only the observable actions performed by an agent. This means that each such sequence may be generated by more than one execution sequence. Accordingly, a non-distinctive observation sequence is one that *satisfies* (formally defined in this paper) paths to more than one goal. The *worst case distinctiveness*(*wcd*) is then the length of the maximal execution that produces a non-distinctive observation sequence. The *wcd* serves as an upper bound on the number of observations that need to be collected for each agent before guaranteeing his objective is recognized.

As a second extension, we provide a compilation of the *partially observable* goal recognition design problem into a classical planning problem, which allows us to exploit existing tools for calculating the *wcd*. Our empirical analysis shows that the compilation allows efficient computation of the *wcd*.

After calculating the *wcd* of a model, it may be desired to minimize it. The third extension we present therefore in-

volves finding the optimal set of modifications that can be introduced to the model in order to reduce the *wcd*. We introduce a new design-time modification method that involves exposing non-observable actions, *e.g.*, by (re)placing sensors. This modification method is used in addition to removing actions from the model (Keren, Gal, and Karpas 2014) in order to minimize the *wcd* while respecting the specified restrictions on the number of allowed modifications. The empirical analysis reveals that the combination of observation exposure and activity restriction allows bigger improvements than each of the measures separately.

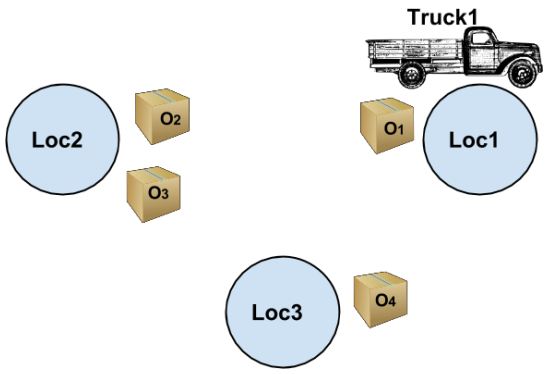


Figure 1: An example of a goal recognition design problem

Example 1 *To illustrate the objective of calculating and optimizing the* wcd *of a goal recognition design model, consider the example depicted in Figure 1, which demonstrates a simplified setting from the logistics domain. There are* 3 *locations, a single truck that is initially located at position* $Loc1$*, and* 4 *objects that are initially placed such that* O_1 *is at location* $Loc1$*,* O_2 *and* O_3 *are at* $Loc2$ *and* O_4 *is at* $Loc3$*. Objects can be moved by loading them onto the truck and unloading them in their destination after the truck reaches it. There are two goals: 1)* O_1 *and* O_2 *at* $Loc3$ *and* O_4 *at* $Loc1$ *(*g_0*) and 2)* O_3 *at* $Loc3$ *(*g_1*). Optimal plans are the only valid plans in this example. In the fully observable setting* wcd $= 0$*, since the goal is revealed by the first action, which can either be* $LoadO_1$ *for* g_0 *or* $Drive\text{-}Loc1\text{-}Loc2$ *for* g_1*. Assume that the non-observable actions are all the load and unload actions, and the observable actions are only the movement of the truck. Here, because the truck needs to travel from* $Loc1$ *to* $Loc2$ *and then to* $Loc3$ *for achieving both goals the goal is revealed only if* $Drive\text{-}Loc3\text{-}Loc1$ *is performed. This means that* g_1 *can be achieved without the system being aware of it. Exposing* $LoadO_1$*, by placing a sensor on the object, changes the situation dramatically.*

The rest of the paper is organized as follows. We start by providing background on classical planning, followed by introducing a model of *grd* problems and the *wcd* value for the *partially observable* setting. We continue by presenting methods for calculating and reducing the *wcd* value of a *grd* problem, respectively. We conclude with an empirical evaluation, a discussion of related work, and concluding remarks.

Background

The basic form of automated planning, referred to as *classical planning*, is a model in which the actions of agents are fully observable and deterministic. A common way to represent classical planning problems is the STRIPS formalism (Fikes and Nilsson 1972): $P = \langle F, I, A, G, C\rangle$ where F is the set of fluents, $I \subseteq F$ is the initial state, $G \subseteq F$ represents the set of goal states, and A is a set of actions. Each action is a triple $a = \langle pre(a), add(a), del(a)\rangle$, that represents the precondition, add, and delete lists respectively, and are all subsets of F. An action a is applicable in state s if $pre(a) \subseteq s$. If action a is applied in state s, it results in a new state $s' = (s \setminus del(a)) \cup add(a)$. $C : A \to \mathbb{R}_0^+$ is a cost function that assigns each action a non-negative cost.

The objective of a planning problem is to find a plan $\pi = \langle a_1, \ldots, a_n\rangle$, a sequence of actions that brings an agent from I to a goal state. The cost $c(\pi)$ of a plan π is $\Sigma_{i=1}^{n}(C(a_i))$. Often, the objective is to find an optimal solution for P, an optimal plan, π^*, that minimizes the cost. We assume the input of the problem includes actions with a uniform cost equal to 1. Therefore, plan cost is equivalent to plan length, and the optimal plans are the shortest ones.

Model

A model for partially observable goal recognition design (*pogrd*) is given as $D = \langle P_D, \mathcal{G}_D, \Pi_{leg}(\mathcal{G}_D)\rangle$ where:

- $P_D = \langle F_D, I_D, A_D\rangle$ is a planning domain where $A = A^o \cup A^{no}$ is a partition of A into observable and non-observable actions, respectively.
- $\mathcal{G}_D$ is a set of possible goals, where each possible goal $g \in \mathcal{G}_D$ is a subset of F_D.
- $\Pi_{leg}(\mathcal{G}_D) = \bigcup_{g\in\mathcal{G}_D} \Pi_{leg}(g)$ is the set of *legal* plans to each of the goals. A plan is an execution of actions that take the agent from I to a goal in $\mathcal{G}_D$. A legal plan is one that is allowed under the assumptions made on the behavior of the agent.

It is worth noting that the model includes an initial state, common to all agents acting in the system. In case there are multiple initial states, there is a simple compilation, which adds a zero cost transition between a dummy common initial state and each initial state, making the model and the methods we propose applicable.

The *pogrd* model divides the system description into three elements, namely *system dynamics*, defined by F_D, I_D, A_D and $\mathcal{G}_D$, *agent strategy* defined by $\Pi_{leg}(\mathcal{G}_D)$, and *observability* defined by the partition of A into A^o and A^{no}. Whenever D is clear from the context we shall use P, $\mathcal{G}$, and $\Pi_{leg}(\mathcal{G})$.

A *path* is a prefix of a legal *plan*. We denote the set of paths in D as $\Pi_{pref}(\mathcal{G}_D)$ and the set of paths to goal $g \in \mathcal{G}_D$ as $\Pi_{pref}(g)$. An *observation sequence* $\vec{o} = \langle a_1, \cdots, a_n\rangle$ is a sequence of actions $a_j \in A^o$. For any two action sequences $\langle a_1, \cdots, a_n\rangle$ and $\langle a_1^{'}, \cdots, a_m^{'}\rangle$ the concatenation of the action sequences is denoted by $\langle a_1, \cdots, a_n\rangle \cdot \langle a_1^{'}, \cdots, a_m^{'}\rangle$.

In the *partially observable* setting the observations sequence that is produced by a path includes only the observable actions that are performed. The relationship between a path and an observation sequence is formally defined next.

Definition 1 *Given a path π, the* observable projection *of π in D, denoted $op_D(\pi)$ ($op(\pi)$ when clear from the context), is recursively defined as follows:*

$$op(\pi) = \begin{cases} \langle\rangle & \text{if } \pi = \langle\rangle \\ \langle a_1\rangle \cdot op(\langle a_2 \cdots a_n\rangle) & \text{if } \pi = \langle a_1, \cdots, a_n\rangle \text{ and } a_1 \in A^o \\ op(\langle a_2, \cdots, a_n\rangle) & \text{otherwise} \end{cases}$$

It is worth noting that the fully observable settings presented by (Keren, Gal, and Karpas 2014; 2015) is a special case of the model presented here, in which the entire action set is observable. In this case, $A^{no} = \emptyset$, $A^o = A$, and the observable projection of any action sequence is equivalent to the action sequence itself.

The relation between an observation sequence and a goal is defined as follows.

Definition 2 *An observation sequence $\vec{o}$ satisfies a goal g if $\exists \pi \in \Pi_{pref}(g)$ s.t. $\vec{o} = op(\pi)$.*

We now define non-distinctive observation sequences and paths, as follows.

Definition 3 *$\vec{o}$ is a* non-distinctive observation sequence *if it satisfies more than one goal. Otherwise, it is* distinctive*.*
π is a non-distinctive path *if its observable projection $\vec{o}$ is non-distinctive. Otherwise, it is* distinctive*.*

The following basic observation sets the relationship between a path and its prefixes.

Lemma 1 *Any prefix of a non-distinctive path is non-distinctive.*

Proof: Let π be a non-distinctive path. According to Definition 3, $\exists g, g' \in \mathcal{G}, \pi \in \Pi_{pref}(g), \pi' \in \Pi_{pref}(g')$ s.t. $g' \neq g$ and $op(\pi) = op(\pi')$. Let π_{pre} be a prefix of π. Then, by Definition 1, $op(\pi_{pre})$ is a prefix of $op(\pi) = op(\pi')$. According to Definition 2, $\pi_{pre} \in \Pi_{pref}(g')$ and therefore, according to Definition 3, π_{pre} is non-distinctive. ∎

Given Lemma 1, we define a maximal non-distinctive prefix of a path π (dubbed $pre_{nd}(\pi)$) to be a prefix of π such that there is no other non-distinctive prefix of π that contains it.

We now examine the effect of moving an action from A^o to A^{no} (dubbed *concealment*) on the number of goals a path satisfies. Relying on Definition 2, we mark the set of goals a path π satisfies in D as $\mathcal{G}_D(\pi)$ and we use $\Pi_{op_D(\pi)}$ to represent the paths π' in D s.t. $op_D(\pi) = op_D(\pi')$.

Theorem 1 *Let D and D' be two* pogrd *models that are identical except that $A^{no}_D \subseteq A^{no}_{D'}$. For any $\pi \in \Pi_{pref}(\mathcal{G}_D)$, $\mathcal{G}_D(\pi) \subseteq \mathcal{G}_{D'}(\pi)$. If, in addition, $\forall a \in A^{no}_{D'} \setminus A^{no}_D$, $\hat{\pi}$ is distinctive in D' for any prefix $\hat{\pi} \cdot \langle a\rangle$ of π, then $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$ (strict equivalence).*

Proof: In general, if $\forall a \in \pi$, $a \notin A^{no}_{D'} \setminus A^{no}_D$ then $op_D(\pi) = op_{D'}(\pi)$ since none of the actions in π changed their observability property. Therefore, $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$.

Otherwise, since $\pi \in \Pi_{pref}(\mathcal{G}_D)$ there exists a goal $g \in \mathcal{G}_D$ and $\pi_g \in \Pi_{leg}(g)$ s.t. $op_D(\pi)$ is a prefix to $op_D(\pi_g)$ (Definition 2). By eliminating any action $a \in A^{no}_{D'} \setminus A^{no}_D$ both in $op_D(\pi)$ and in $op_D(\pi_g)$ we maintain the prefix property. Therefore, $op_{D'}(\pi)$ is a prefix to $op_D(\pi_g)$ and thus $\pi \in \Pi_{pref}(\mathcal{G}_{D'})$.

The only thing left to show now is that if $\hat{\pi}$ is distinctive in D' then $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$. $\hat{\pi}$ is distinctive in D' and therefore $op_{D'}(\hat{\pi})$ satisfies a single goal g (Definition 3). π achieves the goal g and $\hat{\pi}$ is a prefix of π. Therefore, $op_D(\hat{\pi})$ satisfies g as well. $op_{D'}(\hat{\pi} \cdot \langle a\rangle) = op_{D'}(\hat{\pi})$ and therefore also satisfies g.

Assume to the contrary that $\hat{\pi} \cdot \langle a\rangle$ is non-distinctive. Therefore, there must be at least one more goal $g' \neq g$ s.t. $op_{D'}(\hat{\pi} \cdot \langle a\rangle)$ satisfies g'. Therefore, there are two plans π_1 to goal g and π_2 to goal g' such that $op_{D'}(\hat{\pi} \cdot \langle a\rangle)$ is a prefix to both $op_{D'}(\pi_1)$ and $op_{D'}(\pi_2)$. Since $op_{D'}(\hat{\pi} \cdot \langle a\rangle) = op_{D'}(\hat{\pi})$, $op_{D'}(\hat{\pi})$ is also a prefix to both $op_{D'}(\pi_1)$ and $op_{D'}(\pi_2)$, which serves to contradict the assumption that $op_{D'}(\hat{\pi})$ satisfies a single goal and $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$. ∎

Next, we define the worst case distinctiveness (*wcd*) measure of a *pogrd* model. We mark the set of non-distinctive paths of a model D as $\Pi_{nd}(D)$ and define the *wcd* as maximal length of a non-distinctive path in the model.

Definition 4 *The* worst case distinctiveness *of a model D, denoted by* $\text{wcd}(D)$ *is:*

$$wcd(D) = \max_{\pi \in \Pi_{nd}(D)} |\pi|$$

An immediate implication of Theorem 1 is that concealed actions may impact the *wcd* only if they immediately follow a non-distinctive prefix. In particular, if a prefix π is non-distinctive and $\pi \cdot \langle a\rangle$ is distinctive, then by concealing a, $op(\pi \cdot \langle a\rangle)$ may increase the number of goals it satisfies, making it non-distinctive.

Calculating *wcd*

To calculate the *wcd* value of the *pogrd* model we compile it into a classical planning problem, where *wcd* is computed for each pair of goals. Individual results are then combined for computing the *wcd* of the entire model. Note that the following description is restricted to the setting where the set of legal plans is the set of optimal plans. The same approach can be applied to the non-optimal setting present by Keren, Gal, and Karpas (2015) with minor modifications, but is omitted here for lack of space.

The *pogrd* problem with two goals is compiled as a single planning problem involving two agents each aiming at a separate goal. The solution to the problem is a plan for each agent and is divided into two parts by a common *exposure point*. The prefix of a plan up to the exposure point represents a non-distinctive path, one that does not reveal the goal of the agent and may include actions performed by

both agents together in addition to non-observable actions performed by a single agent. After the exposure the goal of the agent is recognized. Since our objective is to reveal the *wcd* of the model we discount the cost of actions that belong to the unexposed prefix of the plan thus encouraging the agents to extend the unexposed prefix as much as possible.

Given a *pogrd* problem D and goals $\mathcal{G} = \{g_0, g_1\}$, the *latest-expose* compilation includes two agents: $agent_0$ aiming at g_0 and $agent_1$ aiming at g_1. The model contains three types of actions: *together-actions* (denoted $A^{0,1}$), actions that both agents perform simultaneously, *non-exposed actions* (A^i_{ne}), actions in A_{no} that are executed by a single agent before the *exposure point*, and *exposed actions* (A^i_e), actions (either observed or non-observed) that are performed by a single agent after the *exposure point*. This setting is achieved by adding to the model the *DoExpose* no-cost operation that represents the exposure point by adding to the current state the *exposed* predicate. Actions in $A^{0,1}$ and A^i_{ne} are applicable only before *DoExpose* occurs while actions in A^i_e can be applied only after.

The use of the exposure point is similar to the use of *split* (Keren, Gal, and Karpas 2014; 2015), where agents are encouraged to act together. However, the addition of non-observable actions to the unexposed extends prefix breaks the symmetry that existed in the fully observable setting. The objective is no longer to find a path that maximizes the number of steps both agents share. Rather, one of the agents seeks a maximum path that keeps the agent unrecognized by combining non-observable actions and actions that are on legal paths to a different goal. To reflect this asymmetry we change the objective to allow only one agent (arbitrarily chosen as $agent_0$) to benefit from performing non-observable actions. We let $\Pi_{nd}(g_i)$ represent non-distinctive paths that are prefixes to plans to g_i and define the *wcd-*g_i to be the maximal *wcd* shared by goal g_i and any other goal.

Definition 5 *The* worst case distinctiveness *of a goal g_i in model D, denoted by* wcd-$g_i(D)$ *is:*

$$\text{wcd-g}_i(D) = \max_{\pi \in \Pi_{nd}(g_i)} |\pi|$$

We now present the *latest-expose* compilation for optimal agents.

Definition 6 *For a* pogrd *problem $D = \langle P, \mathcal{G} = \{g_0, g_1\}, \Pi_{leg}(\mathcal{G})\rangle$ where $P = \langle F, I, A = A_o \cup A_{no}\rangle$ we create a planning problem $P' = \langle F', I', A', G'\rangle$, with action costs C', where:*

- $F' = \{f_0, f_1 \mid f \in F\} \cup \{\textit{exposed}\} \cup \{\textit{done}_0\}$
- $I' = \{f_0, f_1 \mid f \in I\}\}$
- $A' = A^{0,1} \cup A^i_{ne} \cup A^i_e \cup \{\textit{DoExpose}\} \cup \{Done_i\}$
  - $A^{0,1} = \{\langle\{f_0, f_1 \mid f \in pre(a)\} \cup \{\neg\textit{exposed}\}, \{f_0, f_1 \mid f \in add(a)\}, \{f_0, f_1 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - A^i_{ne}= $\{\langle\{f_i \mid f \in pre(a)\} \cup \{\neg\textit{exposed}\}, \{f_i \mid f \in add(a)\}, \{f_i \mid f \in del(a)\}\rangle \mid a \in A_{no}\}$
  - A^0_e= $\{\langle\{f_0 \mid f \in pre(a)\} \cup \{\textit{exposed}\} \cup \{\neg\textit{done}_0\}, \{f_0 \mid f \in add(a)\}, \{f_0 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - A^1_e= $\{\langle\{f_1 \mid f \in pre(a)\} \cup \{\textit{exposed}\} \cup \{\textit{done}_0\}, \{f_1 \mid f \in add(a)\}, \{f_1 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - $\textit{Done}_0 = \langle exposed, \textit{done}_0, \emptyset\rangle$
  - $DoExpose = \langle\emptyset, exposed, \emptyset\rangle$
- $G' = \{f_0 | f \in g_0\} \cup \{f_1 | f \in g_1\}$
- $C'(a) = \begin{cases} 2-\epsilon & \textit{if } a \in A^{0,1} \\ 1-\epsilon & \textit{if } a \in A^i_{ne} \\ 1 & \textit{if } a \in A^i_e \\ 0 & \textit{if } a \in \{\textit{DoExpose}\} \cup \{Done_0\} \end{cases}$

f_i is a copy of F for agent i, $exposed$ is a fluent representing the no-cost action *DoExpose* has occurred, and $done_0$ is a fluent indicating the no-cost $Done_0$ has occurred. The initial state is common to both agents and does not include the *exposed* and $done_0$ fluents. Until a *DoExpose* action is performed, the only actions that can be applied are the actions in $A_{0,1}$ and A^i_{ne}. The *DoExpose* action adds $exposed$ to the current state thus allowing the actions in A^0_e to be applied. After agent 0 accomplishes its goal, $Done_0$ is performed, allowing the application of actions in A^1_e until g_1 is achieved. We enforce agent 1 to wait until agent 0 reaches its goal before starting to act in order to make the search for a solution to P' more efficient by removing symmetries between different interleaving of agent plans after $DoExpose$ occurs.

Having described the compilation we now describe its use. In order to find the *wcd* of the model, we solve a different planning problem for each pair of the goals, each time discounting a single agent. The *wcd-*g_0 value is found by counting the number of actions performed by $agent_0$ before *DoExpose* occurs. The *wcd* value of the model is the maximal *wcd-*g_i over all solved instances.

Given a solution π to $P^{'}$, we mark the projection of π on each agent i as π_i. π_i includes all actions in $A^{0,1}$ A^i_{ne} and A^i_e that appear in π . Accordingly, the projections of the optimal solution π^* to $P^{'}$ on each agent is marked as π^*_i. In addition, $\pi^*_D(g_i)$ represents an optimal solution of g_i in D. Following the idea presented by Keren, Gal, and Karpas (2014), we guarantee that the projection of the optimal solution to P' yields optimal plans for both agents by bounding ϵ, which represents the discount that may be collected for performing actions before *DoExpose* occurs to be lower than the smallest possible diversion from a legal path to any of the agents.

Lemma 2 *π^*_0 and π^*_1 are optimal plans for each agent in P' if*

$$\epsilon < \frac{1}{|\pi^*_D(g_0)|}$$

Proof: In order to guarantee both agents choose optimal paths we require that the maximal discount that may be collected in $P^{'}$ is smaller than the minimal cost of a diversion from a valid path of any of the agents. We therefore bound the difference between the cost of achieving G' in P' and

achieving g_0 and g_1 in D to be smaller than the cost of the minimal diversion from the optimal paths in D. Therefore, assuming the minimal diversion cost is 1 we require that :

$$C'(\pi^*) - \sum_i (|\pi^*_D(g_i)|) < 1$$

The compilation guarantees that action costs in P and $P^{'}$ differ only in the discount that may be accumulated by $agent_0$ in the unexposed prefix of π^*_0, whose maximal length is equal to *wcd*-$g_0(D)$. We therefore need to ensure that

$$\epsilon \cdot \textit{wcd-}g_0(D) < 1$$

and

$$\epsilon < \frac{1}{\textit{wcd-}g_0(D)}$$

In the worst case $agent_0$ can reach g_0 without being exposed. This guarantees an upper bound on *wcd*-$g_0(D)$ s.t. *wcd*-$g_0(D) \leq |\pi^*(g_0)_D|$. Therefore if

$$\epsilon < \frac{1}{|\pi^*_D(g_0)|}$$

both agents act optimally. ∎

Next, we show that the observable projection of the paths prior to the exposure point is non-distinctive. Given a solution π to the $P^{'}$, for any agent i, $unexposed(\pi_i)$ denotes the prefix of π_i prior to the exposure point.

Lemma 3 *$unexposed(\pi_i)$ is non-distinctive.*

Proof: To show that $unexposed(\pi_i)$ is non-distinctive we need to show that $\exists g, g^{'}$ s.t. $g \neq g^{'}$ and $unexposed(\pi_i)$ satisfies both g and $g^{'}$. The compilation guarantees that for any action $a \in unexposed(\pi_i)$, $a \in A^{0,1}$ or A^i_{no}. According to Definition 1, $op(unexposed(\pi_i)) = \{\langle a_1 ... a_n | a \in A^{0,1} \rangle\}$. This means that $op(unexposed(\pi_0)) = op(unexposed(\pi_1))$ and $op(unexposed(\pi_i))$ contains only observable actions the agents perform together and which therefore appear on the plans to both g_0 and g_1. Therefore, $op(unexposed(\pi_i))$ satisfies more then one goal and it is non-distinctive. ∎

Finally, Theorem 2 shows that the optimal solution to P' yields the *wcd*-g_0, thus concluding our proof of correctness.

Theorem 2 *Given a* pogrd *model D with two goals $\langle g_0, g_1 \rangle$ and a model P', created according to Definition 6,* wcd-$g_0(D) = |unexposed(\pi^*_0)|$.

Proof: Lemma 2 guarantees that, apart from the no-cost operation *DoExpose* and Done_0, the solution to $P^{'}$ consists solely of actions that form a pair of optimal paths to each of the goals. Therefore, among the solutions that comply with this condition, π^* is the one that maximizes the accumulated discount. The compilation guarantees that the only way to accumulate discount is by maximizing the number of actions $agent_0$ performs before the exposure point, therefore π^* is the solution to $P^{'}$ that maximizes $|unexposed(\pi_0)|$. Therefore $|unexposed(\pi^*_0)| =$ *wcd*-$g_0(D)$. ∎

In Example 1 the *wcd* is 7 since when calculating *wcd*-g_0 for an agent aiming at g_0, π^*_0 consists of $agent_0$ loading O_1 at Loc_1, driving together with $agent_1$ to Loc_2, loading O_2, driving together with $agent_1$ to Loc_3, unloading both packages and loading 0_4 before the $DoExpose$ occurs. Note that all these actions belong to either $A^{0,1}$ or $A^0_n e$ as opposed to the unobservable action of unloading O_4 which occurs after the exposure point and therefore belongs to A^0_e and is not part of the *wcd* path.

Reducing *wcd*

Having formulated the *wcd* measure, we turn to our second objective of finding ways to optimize the *wcd* by redesigning the model. Optimization can be achieved using two possible modifications, namely *action removal* and *exposure*. The former reduces *wcd* by disallowing actions from being performed. The latter involves exposing an action by moving it from A^{no} to A^o, *e.g.*, by placing a sensor in an area of the model that was previously non-observed.

wcd reduction is performed within a modification budget that represents the constraints to be respected by the reduction method. Given the two possible modifications of a model, we can either provide an integrated budget, B_{total}, or separate budgets $B_{sep} = \langle B_{rem}, B_{sen} \rangle$, where B_{rem} and B_{sen} are the bounds on the number of actions that can be removed and exposed, respectively.

Our objective is to use the budget constraint to minimize the *wcd* value of the model. We mark the modifications by a pair $\langle A_\neg, A_{no \to o} \rangle$, where $A_\neg$ and $A_{no \to o}$ are the disallowed and exposed actions in the transformed model respectively. In our exploration we assume a uniform cost for the removal and exposure of all actions. In addition, we add the requirement that the cost of achieving any of the goals must not increase. Note that these assumptions are made for simplicity and can be easily relaxed without major modification to the reduction algorithms.

Let $D_{\langle A_\neg, A_{no \to o} \rangle}$ denote the transformed version of D where $A = A \setminus A_\neg$, $A_{no} = A_{no} \setminus A_{no \to o}$ and $A_{obs} = A_{obs} \cup A_{no \to o}$. For $B_{sep} = \langle B_{rem}, B_{sen} \rangle$ the objective can be expressed by the following optimization problem.

$$\begin{aligned}
&\underset{A_\neg \cup A_{no \to o}}{\text{minimize}}\ wcd(D_{\langle A_\neg, A_{no \to o} \rangle}) \\
&\text{subject to} \\
&|A_\neg| \leq B_{rem} \text{ and} \\
&|A_{no \to o}| \leq B_{sen} \text{ and} \\
&\forall g \in \mathcal{G}, C^*_D(g) = C^*_{D_{\langle A_\neg, A_{no \to o} \rangle}}(g)
\end{aligned}$$

where $C^*_D(g)$ and $C^*_{D_{\langle A_\neg, A_{no \to o} \rangle}}(g)$ represent the optimal costs of achieving goal g in D and $D_{\langle A_\neg, A_{no \to o} \rangle}$, respectively. When the budget is integrated the first two constraints are replaced with $|A_\neg| + |A_{no \to o}| \leq B_{total}$.

The reduction is performed using a BFS search that iteratively explores all possible modifications to the model. The

initial state is the original model and each successor node introduces a single modification, either exposure or reduction, that was not included in the parent node. A node in the search tree is therefore represented by a pair $\langle A_{\neg}, A_{no\rightarrow o}\rangle$. A node is pruned from the search if any of the constraints have been violated or if there are no more actions to add.

The key question remaining is what are the modifications that should be considered at each stage. A naïve approach would be to consider all possible modifications, which is unpractical and wasteful. Instead, we focus our attention on modifications that have the potential of reducing the *wcd* by either eliminating the *wcd* path (action removal) or by reducing the length of its non-distinctive prefix (exposure). It was already shown that the only actions that need to be considered for elimination are the ones on the current *wcd* path (Keren, Gal, and Karpas 2014). We show that the only non-observable actions that need to be considered for exposure are the ones that appear on the non-distinctive prefix of the current *wcd* path. We refer to the pair of plans that maximize the *wcd* as the wcd *plans* of a model and mark them by $\Pi_{wcd}(D)$.

Theorem 3 *Let $D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}$ be a model and $D_{\langle A_{\neg}, A'_{no\rightarrow o}\rangle}$ be a transformed model s.t. $A_{no\rightarrow o} \subseteq A'_{no\rightarrow o}$. If for all $a \in A'_{no\rightarrow o} \setminus A_{no\rightarrow o}$, $a \notin pre_{nd}(\Pi_{wcd}(D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}))$ then* $\text{wcd}(D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}) = \text{wcd}(D'_{\langle A_{\neg}, A'_{no\rightarrow o}\rangle})$.

The proof is immediate from Theorem 1.

The reduction algorithm creates, for each node, one successor for disallowing each action that appears in $\Pi_{wcd}(D)$ and one successor for exposing each non-observable action in $pre_{nd}(\Pi_{wcd}(D'))$ of the parent node. To avoid redundant computation, we cache computed actions combination.

In Example 1 disallowing actions is not possible without increasing the optimal costs. However, by exposing $LoadO_1$ (i.e. by placing a sensor on the object), the *wcd* is reduced to 0 equivalently to the fully observable setting.

Empirical Evaluation

Our empirical evaluation has several objectives. Having shown that reduced observability may increase the *wcd* value of a model, we first examine empirically the extent of this effect. In addition, we compare the fully observable setting (Keren, Gal, and Karpas 2014) with the *partially observable* setting. Finally, we evaluate the reduction process as well as the effectiveness of action reduction vs. exposure. We describe the datasets and the experiment setup before presenting and discussing the results.

Datasets We use 4 domains of plan recognition (Ramirez and Geffner 2009), namely GRID-NAVIGATION, IPC-GRID$^+$, BLOCK-WORDS, and LOGISTICS. Each problem description contains a domain description, a template for a problem description without the goal, a set of goals and a set of non-observable actions. For each benchmark we generated a separate *grd* problem for each pair of hypotheses and randomly sampled actions to form the non-observable set creating 3 instances with 5%, 10% and 20% randomly chosen non-observable actions. We tested 216 GRID-NAVIGATION instances, 660 IPC-GRID$^+$ instances, 600 BLOCK-WORDS instances, and 300 LOGISTICS instances. In addition, we created a hand crafted benchmark for the LOGISTICS domain dubbed LOGISTICS-Generated, which corresponds to Example 1 where packages load and unload actions are non-observable. This corresponds to real-world settings where satellite imaging can easily track movement of vehicles between locations, but the actual actions performed are obscured from view.

Setup For each problem instance, we calculated the *wcd* value and run-time for the fully observable and *partially observable* settings. For the *wcd* reduction we examined the *partially observable* setting with 3 bound settings: an integrated bound of $B_{total} = 4$ and 2 separate bounds $B_{sep} = \langle 0, 2\rangle$ and $B = \langle 2, 0\rangle$, where the first element of each pair represents B_{rem} and the second B_{sen}. We used the Fast Downward planning system (Helmert 2006) running A^* with the LM-CUT heuristic (Helmert 2006). The experiments were run on Intel(R) Xeon(R) CPU X5690 machines, with a time limit of 30 minutes and memory limit of 2 GB.

Results Table 1 summarizes the impact the ratio of non-observable actions has on the execution time and the *wcd* . The *partially observable* setting is partitioned into the various ratios examined, including a problem with no non-observable activities, which is compared against the values collected for the fully observable setting solved using *latest-split*. For each setting we compare average run time (in seconds) over solved problems. Whenever some of the problems timed-out, we mark in parenthesis the ratio of solved instances. For all domains, *wcd* increases with the increase in the ratio of non-observable actions. As for running time, the *latest-split* outperforms the equivalent *partially observable* setting for all domains except GRID-NAVIGATION, for which performance is similar. However, the overhead for adding non-observable actions is negligible. For the LOGISTICS-Generated domain the increase in *wcd* was more noticeable, with the average *wcd* increasing from 3.77 in the fully observable setting to 4.87 in the *partially observable* setting.

Table 2 summarizes the results for the *wcd* reduction for the *partially observable* setting for each ratio, showing for each budget allocation the average *wcd* reduction achieved within the allocated time (for the LOGISTICS domain results refer only to the problems that were successfully solved in the *wcd* calculation stage). The evaluation shows that for all domains the *wcd* can be decreased by applying at least one of the modification methods separately, but the most substantial reduction is achieved by combining the methods. This hypothesis is supported by the LOGISTICS-Generated domain where the results for the reduction were from 4.87 in the original partially observable setting to 3.34, 3.9 and 3.8 for the $4, \langle 0, 2\rangle, \langle 2, 0\rangle$ bound allocations respectively.

Related Work

Goal recognition design was first introduced by Keren et al. (2014; 2015), offering tools to analyze and solve the *grd* model in fully observable settings. This work relaxes the full observability assumption.

	latest-split		0%		5%		10%		20%	
	Time	*wcd*	Time	*wcd*	Time	*wcd*	Time	*wcd*	Time	*wcd*
GRID-NAVIGATION	0.324	10.36	0.319	10.36	0.356	10.41	0.351	10.46	0.357	11.1
IPC-GRID$^+$	3.53	3.454	8.754	3.454	9.56	3.55	9.64	3.67	9.96	3.84
BLOCKSWORLD	3.03	2.06	29.2	2.06	33.2	2.12	25.89	2.14	31.01	2.82
LOGISTICS	238.497 (0.9)	3.51	165.2 (0.6)	3.71	153.26 (0.31)	3.71	155.48 (0.29)	3.78	191.56 (0.2)	4.1

Table 1: Average running time for *wcd* calculation over solved problems for varying non-observable actions ratio

	5				10				20			
	0	4	2:0	0:2	0	4	2:0	0:2	0	4	2:0	0:2
GRID-NAVIGATION	10.41	9.64	9.71	10.36	10.46	9.34	9.76	10.36	11.1	10.91	11.1	10.91
IPC-GRID$^+$	3.55	2.01	2.01	3.55	3.67	1.75	1.87	2.93	3.84	2.6	2.92	3.35
BLOCKSWORLD	2.12	1.78	1.83	2.12	2.14	1.58	1.64	2.1	2.82	2.15	2.45	2.67
LOGISTICS	3.71	3.37	3.44	3.56	3.78	3.26	3.42	3.51	4.1	3.47	3.8	3.67

Table 2: Average *wcd* after reduction for each ratio and budget allocation achieved within allocated time

The first to establish the connection between the closely related fields of automated planning and goal recognition were Ramirez and Geffner (2009), presenting a compilation of plan recognition problems into classical planning problems that can be solved by any planner. Several works on plan recognition followed this approach (Agotnes 2010; Pattison and Long 2011; Ramirez and Geffner 2010; 2011) by using various automated planning techniques. We follow this approach and introduce a novel compilation of goal recognition design problems with non observable actions into classical planning.

Partial observability in goal recognition has been modeled in various ways (Ramirez and Geffner 2011; Geib and Goldman 2005; Avrahami-Zilberbrand, Kaminka, and Zarosim 2005). In particular, observability can be modeled using a sensor model that includes an observation token for each action (Geffner and Bonet 2013). Note that the *pogrd* model presented for the *partially observable* setting, can be thought of one in which the set of observation tokens O includes an empty observation sequence $o_\emptyset$ and A includes a no-cost action a_{idle} by which an agent remains at his current position.

Conclusions

We presented a model for goal recognition design that accounts for partial observability by partitioning of the set of actions to observable and non-observable actions. We extend the *wcd* measure and proposed ways to calculate and reduce it. By accounting for non-observable actions, we increase the model's relevancy to a wide range of real-world settings.

Our empirical evaluation shows that non-observable actions typically increases the *wcd* value. In addition, we showed that for all of the domains, *wcd* reduction by combining disallowed and exposed actions is preferred over each of the methods separately.

In future work we intend to investigate alternative ways to account for partial observability by creating more elaborated sensor models.

References

Agotnes, T. 2010. Domain independent goal recognition. In *Stairs 2010: Proceedings of the Fifth Starting AI Researchers Symposium*, volume 222, 238. IOS Press, Incorporated.

Avrahami-Zilberbrand, D.; Kaminka, G.; and Zarosim, H. 2005. Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proc. of the AAAI Workshop on Modeling Others from Observations, MOO*.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 12–21.

Cohen, P. R.; Perrault, C. R.; and Allen, J. F. 1981. Beyond question-answering. Technical report, DTIC Document.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(1):1–141.

Geib, C. W., and Goldman, R. P. 2005. Partial observability and probabilistic plan/goal recognition. In *Proceedings of the International workshop on modeling other agents from observations (MOO-05)*.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hong, J. 2001. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research(JAIR 2001)* 15:1–30.

Jarvis, P. A.; Lunt, T. F.; and Myers, K. L. 2004. Identifying terrorist activity with ai plan recognition technology. In *Proceedings of the Sixteenth National Conference on Innovative Applications of Artificial Intelligence (IAAI 2004*, 858–863. AAAI Press.

Kaluza, B.; Kaminka, G. A.; and Tambe, M. 2011. Towards detection of suspicious behavior from multiple observations. In *AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR 2011)*.

Kautz, H., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the Fifth National Conference of the Amercan Association of Artificial Intelligence (AAAI 1986)*, volume 86, 32–37.

Kautz, H.; Etzioni, O.; Fox, D.; Weld, D.; and Shastri, L.

2003. Foundations of assisted cognition systems. *University of Washington, Computer Science Department, Technical Report*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS Conference Proceedings*.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*.

Lesh, N., and Etzioni, O. 1995. A sound and fast goal recognizer. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, volume 95, 1704–1710.

Pattison, D., and Long, D. 2011. Accurately determining intermediate and terminal plan states using bayesian goal recognition. *Proceedings of the First Workshop on Goal, Activity and Plan Recognition(GAPRec 2011)* 32.

Ramirez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*.

Ramirez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2010)*.

Ramirez, M., and Geffner, H. 2011. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence- Volume Three (IJCAI 2011)*, 2009–2014. AAAI Press.