Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning

Pascal Bercher, Thomas Geier, and Susanne Biundo

Institute of Artificial Intelligence,
Ulm University, D-89069 Ulm, Germany,
email: firstName.lastName@uni-ulm.de

Abstract. We present a technique which allows partial-order causal-link (POCL) planning systems to use heuristics known from state-based planning to guide their search.
The technique encodes a given partially ordered partial plan as a new classical planning problem that yields the same set of solutions reachable from the given partial plan. As heuristic estimate of the given partial plan a state-based heuristic can be used estimating the goal distance of the initial state in the encoded problem. This technique also provides the first *admissible* heuristics for POCL planning, simply by using admissible heuristics from state-based planning. To show the potential of our technique, we conducted experiments where we compared two of the currently strongest heuristics from state-based planning with two of the currently best-informed heuristics from POCL planning.

1 Introduction

In most of today's classical planning approaches, problems are solved by informed (heuristic) progression search in the space of states. One reason for the success of this approach is the availability of highly informed heuristics performing a goal-distance estimate for a given state. In contrast, search nodes in plan-based search correspond to partially ordered partial plans; thus, the heuristics known from state-based planning are not directly applicable to plan-based search techniques.

One of the most important representatives of plan-based search is partial-order causal-link (POCL) planning [13, 17]. POCL planning benefits from its least-commitment principle enforcing decisions during planning only if necessary. For instance, POCL planning can be done *lifted* thereby avoiding premature variable bindings. POCL planning has greater flexibility at plan execution time [14] and eases the integration for handling resource or temporal constraints and durative actions [20, 3]. Its knowledge-rich plans furthermore enable the generation of formally sound plan explanations [19].

However, developing well-informed heuristics for POCL planning is a challenging task [21]; thus, heuristics are still rare. To address this shortcoming, we propose a technique which allows to use heuristics already known from state-based search in POCL planning, rather than developing *new* heuristics.

This technique works by transforming a current search node, i.e., a partially ordered partial plan, into a new planning problem, into which the given partial plan is completely encoded, s.t. solutions for the new problem correspond to solutions reachable from the encoded search node. Then, we evaluate the heuristic estimate of the transformed problem's initial state using any heuristic known from state-based search, and use it as heuristic estimate of the search node. As it turns out, not every state-based heuristic works with our technique, but we obtained promising empirical results for some of them.

The remainder of the paper is structured as follows: the next section is devoted to the problem formalization. Section 3 introduces the proposed transformation. In Section 4, we discuss several issues and questions arising when using the technique in practice. In Section 5, we evaluate our approach by comparing our POCL planning system using four different heuristics: two of them are the currently best-informed heuristics known for POCL planning, whereas the other two use state-of-the-art heuristics known from state-based planning in combination with our problem encoding. Finally, Section 6 concludes the paper.

2 POCL Planning

A planning domain is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$, where $\mathcal{V}$ is a finite set of boolean *state variables*, $\mathcal{S} = 2^{\mathcal{V}}$ is the set of *states*, and $\mathcal{A}$ is a finite set of *actions*, each having the form $(\mathit{pre}, \mathit{add}, \mathit{del})$, where $\mathit{pre}, \mathit{add}, \mathit{del} \subseteq \mathcal{V}$. An action is applicable in a state $s \in \mathcal{S}$ if its precondition pre holds in s, i.e., $\mathit{pre} \subseteq s$. Its application generates the state $(s \setminus \mathit{del}) \cup \mathit{add}$. The applicability and application of action sequences is defined as usual. A *planning problem* in STRIPS notation [5] is a tuple $\pi = \langle \mathcal{D}, s_{\mathit{init}}, g \rangle$ with $s_{\mathit{init}} \in \mathcal{S}$ being the *initial state* and $g \subseteq \mathcal{V}$ being the *goal description*. A *solution* to π is an applicable action sequence starting in s_{init} and generating a state $s \supseteq g$ that satisfies the goal condition.

POCL planning is a technique that solves planning problems via search in the space of partial plans. A *partial plan* is a tuple $(\mathit{PS}, \prec, \mathit{CL})$. PS is a set of plan steps, each being a pair $l{:}a$ with an action $a \in \mathcal{A}$ and a unique label $l \in L$ with L being an infinite set of label symbols to differentiate multiple occurrences of the same action within a partial plan. The set $\prec \subset L \times L$ represents ordering constraints and induces a strict partial order on the plan steps in PS. CL is a set of causal links. A causal link $(l, v, l') \in L \times \mathcal{V} \times L$ testifies that the precondition $v \in \mathcal{V}$ of the plan step with label l' (called the *consumer*) is provided by the action with label l (called the *producer*). That is, if $l{:}(\mathit{pre}, \mathit{add}, \mathit{del}) \in \mathit{PS}$, $l'{:}(\mathit{pre}', \mathit{add}', \mathit{del}') \in \mathit{PS}$, and $(l, v, l') \in \mathit{CL}$, then $v \in \mathit{add}$ and $v \in \mathit{pre}'$. Furthermore, we demand $l \prec l'$ if $(l, v, l') \in \mathit{CL}$.

Now, π can be represented as a POCL planning problem $\langle \mathcal{D}, P_{\mathit{init}} \rangle$, where $P_{\mathit{init}} := (\{l_0{:}a_0, l_\infty{:}a_\infty\}, \{(l_0, l_\infty)\}, \emptyset)$ is the *initial partial plan*. The actions a_0 and a_∞ encode the initial state and goal description: a_0 has no precondition and s_{init} as add effect and a_∞ has g as precondition and no effects. A solution to such a problem is a partial plan P with no *flaws*. Flaws represent plan elements violating solution criteria. An *open precondition* flaw is a tuple $(v, l) \in \mathcal{V} \times L$

specifying that the precondition v of the plan step with label l is not yet protected by a causal link. A *causal threat* flaw is a tuple $(l,(l',v,l'')) \in L \times CL$ specifying that the plan step $l{:}(pre, add, del)$ with $v \in del$ may be ordered between the plan steps with label l' and l''. We say, the plan step with label l *threatens* the causal link (l',v,l''), since it might undo its protected condition v.

If a partial plan P has no flaws, every linearization of its plan steps respecting its ordering constraints is a solution to the planning problem in STRIPS notation. Hence, P is called a solution to the corresponding POCL problem.

POCL planning can be regarded as a refinement procedure [12], since it *refines* the initial partial plan P_{init} step-wise until a solution is generated. The algorithm works as follows [22]. First, a most-promising partial plan P is selected based on heuristics estimating the goal-distance or quality of P. Given such a partial plan P, a flaw selection function selects one of its flaws and resolves it. For that end, all *modifications* are applied, which are all possibilities to resolve the given flaw. A causal threat flaw $(l,(l',v,l'')) \in \mathcal{F}_{CausalThreat}$ can only be resolved by *promotion* or *demotion*. These modifications promote the plan step with label l before the one with label l', and demote it behind the one with label l'', respectively. An open precondition flaw $(v,l) \in \mathcal{F}_{OpenPrecondition}$ can only be resolved by inserting a causal link (l',v,l) which protects the open precondition v. This can be done either by using a plan step already present in the current partial plan, or by a new action from $\mathcal{A}$. The two-stage procedure of selecting a partial plan, calculating its flaws, and selecting and resolving a flaw is repeated until a partial plan P without flaws is generated. Hence, P is a solution to the POCL planning problem and returned.

3 Using State-Based Heuristics for POCL Planning

We encode a partially ordered partial plan into a new STRIPS planning problem. A similar encoding was already proposed by Ramírez and Geffner [18]. However, their encoding was used in the context of plan recognition for compiling observations away and it does not feature a partial order, causal links, nor did they state formal properties.

Given a planning problem in STRIPS notation $\pi = \langle\langle \mathcal{V}, \mathcal{A}\rangle, s_{init}, g\rangle$ and a partial plan $P = (PS, \prec, CL)$, let $enc_{plan}(P,\pi) = \langle\langle \mathcal{V}', \mathcal{A}'\rangle, s'_{init}, g'\rangle$ be the *encoding* of P and π with:

$$\begin{aligned}
\mathcal{V}' &:= \mathcal{V} \cup \{l_-, l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}\\
\mathcal{A}' &:= \mathcal{A} \cup \{enc_{planStep}(l{:}a, \prec) \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}, \text{ with}\\
&\quad enc_{planStep}(l{:}(pre, add, del), \prec)\\
&\qquad := (pre \cup \{l_-\} \cup \{l'_+ \mid l' \prec l, l' \neq l_0\}, add \cup \{l_+\}, del \cup \{l_-\}),\\
s'_{init} &:= s_{init} \cup \{l_- \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}\\
g' &:= g \cup \{l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}
\end{aligned}$$

The resulting problem subsumes the original one and extends it in the following way: all plan steps in P become additional actions in $\mathcal{A}'$ (we do not encode the artificial start and end actions, since their purpose is already reflected by the initial state and goal description). For each plan step $l{:}a$, we introduce two indicator variables l_- and l_+ that encode that $l{:}a$ has not or has been executed. Initially, none of the actions representing the encoding of these plan steps are marked as executed and the (additional) goal is to execute all of them. Furthermore, these actions use the indicator variables to ensure that they can only be executed in an order consistent with the partial order of the partial plan.

Although the encoding can be done in linear time [1], the effort for evaluating heuristics might increase as search progresses, since the resulting problem is of larger size than the original one. We will discuss this issue in the next section.

For the sake of simplicity, the formally specified function enc_{plan} ignores causal links. Since causal links induce additional constraints on a partial plan (cf. Section 2), compiling them away, too, captures even more information. The compilation can be done as follows: let (l_1, v, l_2) be a causal link, $l_1{:}a_1$ and $l_2{:}a_2$ the two corresponding plan steps, and a'_1 and a'_2 their encodings within $\mathcal{A}'$. We need to ensure that no action with v as delete effect can be inserted between a'_1 and a'_2. To that end, we introduce a counter variable $count(v)$ which counts how many causal links with the protected condition v are "currently active".[1] To update that counter correctly, any (encoded) action producing a causal link with condition v has the precondition $count(v) = i$ and the effect $count(v) = i + 1$. Analogously, any (encoded) action consuming such a causal link has the precondition $count(v) = i$ and the effect $count(v) = i - 1$. Then, any (encoded and non-encoded) action having v in its delete list has the precondition $count(v) = 0$. Note that the original planning problem does not need to be changed for every partial plan, although we need to add the precondition $count(v) = 0$ to each action for which there is a causal link (l_1, v, l_2) in the current partial plan. We can process the domain only *once* by adding the precondition $count(v) = 0$ to any action for any state variable v in advance. Concerning the overall runtime for compiling away causal links, assume a' being a (compiled) action consuming n and providing m causal links. Then, $|CL|^{(n+m)}$ actions must be created to provide all possible combinations of the *count* variables, where CL is the set of causal links of the partial plan to be encoded. The compilation is therefore exponential in the maximal number of preconditions and effects of all actions. Hence, assuming the planning *domain* is given in advance, our compilation is *polynomial*. In practice, it is also polynomial if the domain is not given in advance, because the maximal number of preconditions and effects is usually bounded by a small constant and does not grow with the domain description.

Before we can state the central property of the transformed problem, we need some further definitions. Let $P = (PS, \prec, CL)$ be a partial plan. Then, $ref(P) := \{\langle PS', \prec', CL' \rangle \mid PS' \supseteq PS, \prec' \supseteq \prec, CL' \supseteq CL\}$ is called the set of all *refinements* of P, i.e., the set of all partial plans which can be derived from P

[1] For the sake of simplicity, we use functions to describe the counter. Since these functions are simple increment and decrement operations, converting them into the STRIPS formalism is possible in linear time w.r.t. their maximum value which is bound by the number of causal links in the given partial plan.

by adding plan elements. Let $sol(\pi)$ be the set of all *solution* plans of π. We call $sol(\pi, P) := sol(\pi) \cap ref(P)$ the set of all solutions of π refining P.

Now, we define mappings to transform partial plans derived from the planning problem $enc_{plan}(P, \pi)$ to partial plans from the original planning problem π.[2] The functions $dec_{planStep}(l{:}(pre, add, del), \mathcal{V}) := l{:}(pre \cap \mathcal{V}, add \cap \mathcal{V}, del \cap \mathcal{V})$ and $dec_{plan}(\langle PS, \prec, CL\rangle, \pi) := \langle\{dec_{planStep}(l{:}a, \mathcal{V}) \mid l{:}a \in PS\}, \prec, \{(l, v, l') \in CL \mid v \in \mathcal{V}\}\rangle$ are called the *decoding* of a plan step and a partial plan, respectively. Thus, given a partial plan P' from the planning problem $enc_{plan}(P, \pi)$, $dec_{plan}(P', \pi)$ eliminates the additional variables and causal links used by the encoding.

The following proposition states that every solution of the original planning problem, which is also a refinement of the given partial plan, does also exist as a solution for the encoded problem and, furthermore, the converse holds as well: every solution of the encoded problem can be decoded into a solution of the original one, which is a refinement of the given partial plan, too. Thus, the set of solutions of the transformed planning problem is identical to the set of solutions of the original problem, reachable from the current partial plan.

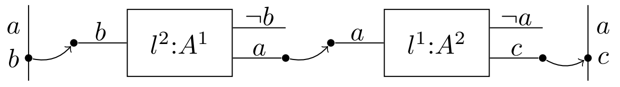
Proposition 1 *Let π be a planning problem and P a partial plan. It holds:*

- *if there is a partial plan P_{sol}, s.t. $P_{sol} \in sol(\pi, P)$, then there exists a partial plan P'_{sol} with $P'_{sol} \in sol(enc_{plan}(P, \pi))$ and $dec_{plan}(P'_{sol}, \pi) = P_{sol}$*
- *if there is a partial plan P'_{sol}, s.t. $P'_{sol} \in sol(enc_{plan}(P, \pi))$, then $dec_{plan}(P'_{sol}, \pi) \in sol(\pi, P)$*

Assuming the plan quality is based on action costs, we can use that proposition to find a heuristic function $h(\pi, P)$ that estimates the goal distance in π from the *partial plan* P by transforming π and P into the planning problem $\pi' = enc_{plan}(P, \pi)$ and setting $h(\pi, P) := \max\{h_{sb}(\pi', s'_{init}) - cost(P), 0\}$, where h_{sb} is any heuristic that takes a *state* as input. We need to subtract the action costs of P, since a heuristic estimate for P *excludes* the actions already present in P, whereas a heuristic estimate for s'_{init} should detect the necessity of inserting them and hence *includes* their cost as well. Taking the maximum of that value and zero is done in case the heuristic is overly positive and returns an estimated cost value lower than those of the actions already present in P.

Since, due to Proposition 1, the set of solutions of π is isomorphic to the solutions of π', even regarding action costs, using an admissible heuristic h_{sb} consequently makes $h(\pi, P)$ admissible, too. This is an interesting property of our approach, since there are no admissible POCL heuristics known to the literature.

Example. Let $\pi = \langle\langle\mathcal{V}, \mathcal{A}\rangle, s_{init}, g\rangle$ be a planning problem with $\mathcal{V} := \{a, b, c\}$, $\mathcal{A} := \{(\{b\}, \{a\}, \{b\}), (\{a\}, \{c\}, \{a\})\}$, $s_{init} := \{a, b\}$, and $g := \{a, c\}$. Let P be a partial plan which was obtained by a POCL algorithm as depicted below:



[2] For the sake of simplicity, our decoding assumes that no causal links were compiled away. Decoding the encoded causal links is straight-forward.

The arrows indicate causal links and A^1 and A^2 the actions of $\mathcal{A}$. P has only one open precondition: (a, l_∞), which encodes the last remaining goal condition.

The transformed problem, without compiling away causal links, is given by $enc_{plan}(P, \pi) = \langle\langle \mathcal{V}', \mathcal{A}'\rangle, s'_{init}, g'\rangle$ with:

$$\begin{aligned}
\mathcal{V}' &:= \{a, b, c, l^1_+, l^1_-, l^2_+, l^2_-\} \\
\mathcal{A}' &:= \{(\{b\}, \{a\}, \{b\}), && // \; A^1 \\
&\qquad (\{b, l^2_-\}, \{a, l^2_+\}, \{b, l^2_-\}), && // \; enc(l^2{:}A^1) \\
&\qquad (\{a\}, \{c\}, \{a\}), && // \; A^2 \\
&\qquad (\{a, l^1_-, l^2_+\}, \{c, l^1_+\}, \{a, l^1_-\})\} && // \; enc(l^1{:}A^2) \\
s'_{init} &:= \{a, b, l^1_-, l^2_-\} \\
g' &:= \{a, c, l^1_+, l^2_+\}
\end{aligned}$$

A heuristic estimate based on the transformed problem may incorporate the negative effects of $l^2{:}A^1$ and $l^1{:}A^2$ and has thus the potential to discover the partial plan/state to be invalid and thus prune the search space.

4 Discussion

Relaxation. Not every state-based heuristic is suited for our proposed approach. In order to determine how informed a chosen heuristic function is when used with our technique, one has to investigate the effect of the (heuristic-dependent) relaxation on the actions in $\mathcal{A}_{new} := \mathcal{A}' \setminus \mathcal{A}$. The actions in $\mathcal{A}_{new}$ (together with the additional goals) encode the planning progress so far, just like the current state does in state-based planning. Thus, relaxing them can have a strong impact on the resulting heuristic values. For instance, in our experiments, we noticed that the FF heuristic [10] always obtains the same estimates for the encoded problems of all search nodes making the heuristic blind.

Preprocessing. Some heuristics, like merge and shrink abstraction (M&S) [4, 9], perform a preprocessing step before the actual search and make up for it when retrieving each single heuristic value. Since we obtain a new planning problem for each single partial plan, a naive approach using this kind of heuristics would also perform that preprocessing in every search node, which is obviously not beneficial (and no *pre*-processing). Thus, given a specific heuristic, one has to investigate whether certain preprocessing steps can be done only *once* and then updated per partial plan if necessary.

Runtime. Although the transformation itself can be done efficiently, the time of evaluating heuristics might increase with the size of the encoded problem. At first glance, this might seem a strange property, since one would expect the heuristic calculation time either to remain constant (as for abstraction heuristics [4, 9]) or to *decrease* (as for the add or FF heuristics [6, 10]), as a partial plan comes closer to a solution. However, since partial plans are complex structures and many interesting decision problems involving them are NP hard w.r.t. their size [15], it is not surprising that evaluating heuristic estimates becomes more expensive as partial plans grow in size.

Ground Planning. The presentation of the proposed transformation in the paper assumes a ground problem representation. However, the approach also works for lifted planning without alterations to the encoding function. In lifted planning [22], the given partial plan is only *partially* ground, i.e., some action parameters are bound to constants, and the remaining ones are either unconstrained, codesignated or non-codesignated. Using the same encoding process but ignoring these designation constraints already works as described, since the initial state of the resulting encoded planning problem is still ground and evaluating its heuristic estimate is thus possible without alterations. Encoding the designation constraints is also possible, but ignoring them is just a problem relaxation as is ignoring causal links.

5 Evaluation

We implemented the described encoding without compiling away causal links in our POCL planning system. We compare the performance of planning using the encoding with two state-of-the-art state-based heuristics against the currently best-informed POCL heuristics. We also show results for a state-based planner.

The used POCL planner is implemented in Java®. As search strategy, we use weighted A* with weight 2. That is, a partial plan p with minimal f value is selected, given by $f(p) := g(p) + 2 * h(p)$ with $g(p)$ being the cost of p and $h(p)$ being its heuristic estimate. In cases where several partial plans have the same f value, we break ties by selecting a partial plan with higher cost; remaining ties are broken by the *LIFO* strategy thereby preferring the newest partial plan. As flaw selection function, we use a sequence of two flaw selection strategies. The first strategy prefers newest flaws (where all flaws detected in the same plan are regarded equally new). On a tie, we then use the *Least Cost Flaw Repair* strategy [11], which selects a flaw for which there are the least number of modifications, thereby minimizing the branching factor. Remaining ties are broken by chance. We configured our system to plan ground, because our current implementation only supports a ground problem encoding.

As POCL heuristics, we selected the two currently best-informed heuristics: the *Relax Heuristic* [16] and the *Additive Heuristic for POCL planning* [22]. They are adaptations of the *FF heuristic* [10] and the *add heuristic* [6], respectively. Both heuristics identify the open preconditions of the current partial plan and estimate the action costs to achieve them based on delete relaxation using a planning graph [2]. These heuristics are implemented natively in our system.[3]

[3] Our implementation of the Relax Heuristic takes into account *all* action costs in a relaxed plan, whereas the original version assumes cost 0 for all actions already present. We used our variant for the experiments, since it solved more problems than the original version.

As state-based heuristics, we chose the *LM-Cut* heuristic [8], which is a landmark-based heuristic and an admissible approximation to h^+, and the *Merge and Shrink* (M&S) heuristic [4, 9], which is an abstraction-based heuristic.

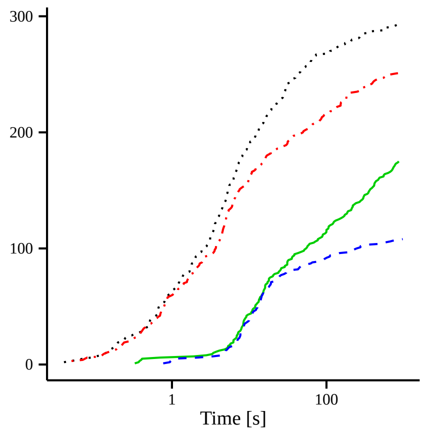
To evaluate heuristics from state-based planning, we chose to use an existing implementation. When planning using state-based heuristics, the POCL planner creates a domain and problem PDDL file for each search node encoding the corresponding partial plan, but ignoring causal links. We then use a modified version of the Fast Downward planning system [7] that exits after calculating the heuristic value for the initial state. While this approach saved us much implementation work, the obtained results are to be interpreted with care, since the process of calling another planning system in each search node is rather time-consuming: while the average runtime of the Add and Relax heuristics is at most 4 ms per search node over all evaluated problems, the LM-Cut heuristic has a mean runtime of 958 ms and a median of 225 ms. For the M&S heuristic[4], the mean is 7,500 ms and the median 542 ms. The very high runtimes of M&S are contributed to the fact that it performs a preprocessing step for every search node. Of course, in a native implementation of that heuristic in combination with our problem encoding, one would have to investigate whether an incremental preprocessing is possible as discussed in the last section.

Thus, the results using the state-based configurations are bound to be dominated by all others in terms of solved instances and runtime. Therefore, we focus our evaluation on the quality of the heuristics measured by the size of the produced search space in case a solution was found.

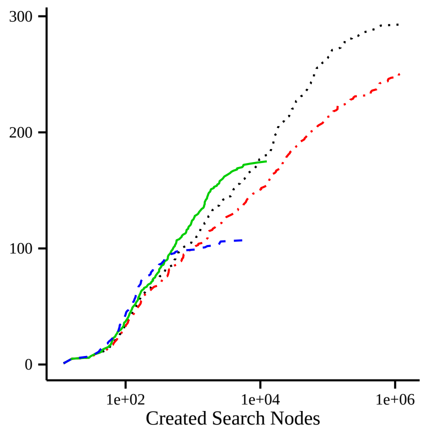
We evaluate on the non-temporal STRIPS problems taken from the International Planning Competitions (IPC) 1 to 5. Domains from the IPC 6 and 7 use action costs, which our system does not support. Missing domains from the IPC 1 to 5 use language features that cannot be handled either by our planner or by Fast Downward. From each domain we chose a number of instances consecutively, beginning with the smallest ones. The used domains and problems are given in Table 1; furthermore, the table contains the number of solved instances per domain by any of the four configurations. We also included results for the state-based planner Fast Downward. This planner, which is implemented in C++, clearly dominates our Java based POCL planner. For one problem, Fast Downward with M&S created 737 million nodes while our POCL planner created at most 2.9 million nodes, both for Add and Relax heuristic. Despite this discrepancy, the performance of the POCL planner using the Add heuristic surpasses Fast Downward using M&S in some domains.

The POCL planner was given 15 minutes of CPU time and 2GB of memory for each problem. For the configurations using the encoding, the call to Fast Downward also counts towards the time limit (including time spent generating the additional PDDL files), but not towards the memory limit. For the comparison with Fast Downward, we used a 15 minute wall time limit and no memory limit. All experiments were run on a 12 core Intel Xeon® with 2.4GHz.

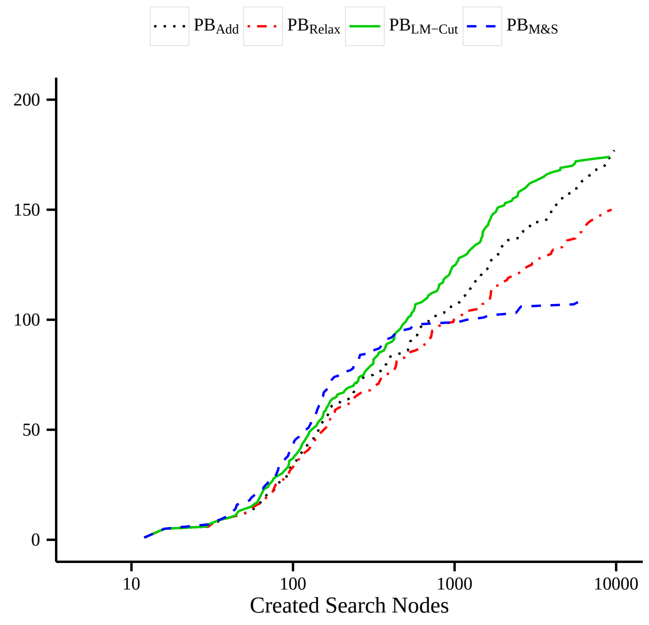
[4] We chose f-preserving abstractions and 1500 abstract states. We chose a rather low number of abstract states to speed up the calculation time.



(a) Solved instances over CPU time.



(b) Solved instances over created nodes.



(c) Enlarged view of 1b; solved instances over created nodes.

Fig. 1: These plots show the number of solved instances on their y-axis, while the x-axis shows either CPU time or the number of created nodes. The configurations PB$_{\text{Add}}$ and PB$_{\text{Relax}}$ stand for POCL planning using the Add or the Relax heuristic, respectively. PB$_{\text{LM-Cut}}$ and PB$_{\text{M\&S}}$ denote POCL planning using the LM-Cut and the Merge and Shrink heuristic.

Figure 1a shows the number of solved instances over the used CPU time. As we can observe the transformation-based heuristics are severely dominated by the natively implemented POCL heuristics, as we expected. Figures 1b and 1c show the size of the explored search space over the number of solved instances. This means that configurations with higher curves solve more problems using the same number of visited nodes. We can observe that both transformation-based heuristics act more informed than the existing POCL heuristics in the beginning. The transformation-based heuristic using LM-Cut remains above the best existing POCL heuristic (Add) for the complete range of problems it could solve. When using M&S the performance deteriorates for the more complex problems, which we attribute to the small number of abstract states. In fact, a reasonable number of abstract states should be chosen domain dependently [9]. It is also the case that many runs experienced time outs after having explored only a small number of nodes. This means that the true curves of the transformation-based heuristics are expected to lie higher than depicted.

In summary, we can state that the experiments offer a promising perspective on the usefulness of the proposed transformation-based heuristics. In particular the LM-Cut heuristic proved to act more informed than the currently best known POCL heuristic, in addition to being the first known admissible one. Since the calculation of LM-Cut does not rely on preprocessing, like the Merge and Shrink heuristic does, we are optimistic that a native implementation of LM-Cut for POCL planning will offer a significant performance boost for POCL planning.

6 Conclusion

We presented a technique which allows planners performing search in the space of plans to use standard classical planning heuristics known from state-based planning. This technique is based on a transformation which encodes a given partial plan by means of an altered planning problem, s.t. evaluating the goal distance for the given partial plan corresponds to evaluating the goal distance for the initial state of the new planning problem.

We evaluated our approach by running our POCL planning system with two of the currently best-informed heuristics for POCL planning and two state-of-the-art-heuristics from state-based planning based on the proposed transformation. Whereas the first two heuristics are natively implemented in our system, the latter two are obtained by running Fast Downward in each search node and extracting its heuristic estimate for the initial state. The empirical data shows that our encoding works well with the evaluated state-based heuristics. In fact, one of these heuristics is even more informed than the best evaluated POCL heuristic, as it creates smaller search spaces in order to find solutions.

In future work we want to implement the encoding of causal links and evaluate our technique using a native implementation of the (state-based) LM-cut heuristic. Furthermore, we want to investigate whether the LM-cut heuristic can be *directly* transferred to the POCL setting without the compilation.

Table 1: This table gives the number of used problems per domain (n) and the number of solved instances per configuration and domain. The first configurations use our *plan*-based configurations and the right-most columns are the results of the *state-based* Fast Downward planner. All problems in the same block belong to the same IPC, from 1 to 5. A bold entry specifies the most number of solved instances among all configurations of the POCL planning system.

Domain	n	PB_{Add}	PB_{Relax}	$\text{PB}_{\text{LM-Cut}}$	$\text{PB}_{\text{M\&S}}$	$\text{SB}_{\text{LM-Cut}}$	$\text{SB}_{\text{M\&S}}$
grid	5	**0**	**0**	**0**	**0**	2	2
gripper	20	14	**20**	1	1	20	8
logistics	20	**16**	15	6	0	16	1
movie	30	**30**	**30**	**30**	**30**	30	30
mystery	20	8	**10**	5	5	13	13
mystery-prime	20	3	**4**	2	1	12	12
blocks	21	2	3	3	**5**	21	21
logistics	28	**28**	**28**	27	5	28	15
miconic	100	**100**	53	65	29	100	68
depot	22	**2**	**2**	1	1	11	7
driverlog	20	7	**9**	3	3	15	12
freecell	20	**0**	**0**	**0**	**0**	6	6
rover	20	**20**	19	9	5	18	8
zeno-travel	20	4	4	3	**5**	16	13
airport	20	**18**	15	6	5	20	18
pipesworld-noTankage	20	**8**	5	1	1	18	19
pipesworld-Tankage	20	**1**	**1**	**1**	**1**	11	14
satellite	20	**18**	**18**	4	3	15	7
pipesworld	20	**1**	**1**	**1**	**1**	11	14
rover	20	**0**	**0**	**0**	**0**	18	8
storage	20	7	**9**	5	5	17	15
tpp	20	**9**	8	5	5	9	7
total	526	296	254	178	111	427	318

Acknowledgements

This work is done within the Transregional Collaborative Research Centre SFB/ TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

1. Bercher, P., Biundo, S.: Encoding partial plans for heuristic search. In: Proceedings of the 4th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2013). pp. 11–15 (2013)
2. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90, 281–300 (1997)
3. Coles, A., Coles, A., Fox, M., Long, D.: Forward-chaining partial-order planning. In: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010). pp. 42–49. AAAI Press (2010)

4. Dräger, K., Finkbeiner, B., Podelski, A.: Directed model checking with distance-preserving abstractions. In: Valmari, A. (ed.) SPIN. Lecture Notes in Computer Science, vol. 3925, pp. 19–34. Springer (2006)
5. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
6. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000). pp. 140–149. AAAI Press (2000)
7. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research (JAIR) 26, 191–246 (2006)
8. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: Whats the difference anyway? In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009). vol. 9, pp. 162–169 (2009)
9. Helmert, M., Haslum, P., Hoffmann, J.: Flexible abstraction heuristics for optimal sequential planning. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). pp. 176–183 (2007)
10. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research (JAIR) 14, 253–302 (May 2001)
11. Joslin, D., Pollack, M.E.: Least-cost flaw repair: A plan refinement strategy for partial-order planning. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994). pp. 1004–1009. AAAI Press (1994)
12. Kambhampati, S.: Refinement planning as a unifying framework for plan synthesis. AI Magazine 18(2), 67–98 (1997)
13. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991). pp. 634–639. AAAI Press (1991)
14. Muise, C., McIlraith, S.A., Beck, J.C.: Monitoring the execution of partial-order plans via regression. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 1975–1982. AAAI Press (2011)
15. Nebel, B., Bäckström, C.: On the computational complexity of temporal projection, planning, and plan validation. Artificial Intelligence 66(1), 125–160 (1994)
16. Nguyen, X., Kambhampati, S.: Reviving partial order planning. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001). pp. 459–466. Morgan Kaufmann (2001)
17. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In: Proceedings of the third International Conference on Knowledge Representation and Reasoning. pp. 103–114. Morgan Kaufmann (1992)
18. Ramírez, M., Geffner, H.: Plan recognition as planning. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1778–1783. AAAI Press (July 2009)
19. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012). pp. 225–233. AAAI Press (6 2012)
20. Vidal, V., Geffner, H.: Branching and pruning: An optimal temporal POCL planner based on constraint programming. Artificial Intelligence 170(3), 298–335 (2006)
21. Weld, D.S.: Systematic nonlinear planning: A commentary. AI Magazine 32(1), 101–103 (2011)
22. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research (JAIR) 20, 405–430 (2003)