A Depth-Balanced Approach to Decompositional Planning for Problems where Hierarchical Depth is Requested

David R. Winer$^1$ and Rogelio E. Cardona-Rivera$^{1,2}$

$^1$School of Computing
$^2$Entertainment Arts and Engineering Program
University of Utah
Salt Lake City, UT, USA
{drwiner, rogelio}@cs.utah.edu

Abstract

Hybrid planning with task insertion for solving classical planning problems, or decompositional planning, combines partial-order causal link planning with hierarchical task networks, where steps in the plan may represent composite (i.e., compound) actions that are decomposable into sub-steps using hierarchical knowledge. We have designed a planning algorithm that responds to a request for maximizing the hierarchical depth of plans while minimizing the plan length. In some applications, plans that adhere to hierarchical constraints are preferred over other valid plans. One of the main obstacles of this challenge is to incentivize the planner to insert composite actions while avoiding excessive search on the depth attribute. We introduce plan scoring heuristics that avoid over-discounting and under-discounting depth using a novel way to measure plan depth. We evaluate these heuristics on test problems and demonstrate that we can generate deep, low-cost solutions to planning problems while avoiding excessive search.

Hybrid planning is a plan-space planning paradigm that combines partial-order causal link reasoning (Weld 1994) with hierarchical knowledge (Erol, Hendler, and Nau 1994) in order to solve a hybrid planning problem (the refinement of an initial partial plan into a plan with no flaws). While there exist several variants of hybrid planning (e.g. Young, Pollack, and Moore 1994, Lee-Urban 2012, Bechon et al. 2014), all variants afford some representation of task hierarchies through two kinds of tasks (i.e. steps): primitive and composite. The former are similar to steps in partial-order causal link (POCL) planning. The latter are drawn from hierarchical task network (HTN) planning (but also contain preconditions and effects as in POCL planning); they represent abstract tasks involving several more-primitive steps. Whereas a primitive step that has been added to a plan can be directly executed (assuming its preconditions hold), composite steps that have been added are not directly executable; a more primitive sub-plan for the composite step must be found that depends on the composite’s preconditions and that achieves the composite’s effects. Such a sub-plan may be input to a hybrid planner through a decomposition method.

Our planning applications make a non-standard depth request for hybrid planners. The request is that the planner maximizes the ratio of hierarchical depth (number of decomposition methods) to plan length (number of primitive tasks) of generated plans. The number of decomposition methods it uses to refine the initial plan is an integral part of the planning problem’s solution. Composite tasks are not wholly substituted for sub-plans that decompose them, but rather kept around to identify the hierarchical structure inherent in the plan. The underlying assumption is that the high-level structure of the plan (identified through the decomposition methods) is implicitly meaningful or useful for the planning agent. For example, in planning-based natural language generation (Garoufi 2014), plans may be preferred if they follow recognizable discourse patterns, which may be computed from data-driven observations and operationalized as hierarchical knowledge. Another example is the case of planning-based narrative generation (Young et al. 2013), wherein plans may be preferred if they follow normative narrative structure, often analytically identified as containing hierarchical segments (Prince 2003; Bordwell, Thompson, and Smith 1997).

Typically, hybrid planners will insert a composite task, rather than a primitive task, if the composite task is explicitly needed because it has some primary effect: an effect that none of the tasks in its decompositional refinement can establish on its own (Kambhampati, Mali, and Srivastava 1998). A primary effect can characterize something that is more than the sum of its parts; for example, an argument is made by refuting facts, establishing background, referring to evidence, and making a conclusion, but none of these items are sufficient on its own. In contrast, in the classic travel planning domain, a goal to be located at a destination is achieved by a primitive task of exiting a plane that is at said destination; thus, a composite task – e.g. travel-by-plane – is not inserted into a plan unless it is estimated to save the planner time and effort for repairing the same goal condition. Given that heuristics tend to underestimate effort saved, and composite tasks add more to the plan length than primitive tasks, a planner using a best-first search is going to evaluate a repair with a primitive task as cheaper than a plan that makes the same repair with a composite task. We address the problem of fulfilling the depth request in hybrid domains where composite tasks may not have primary effects.

The decision point we focus on is which task to insert to repair open conditions, which (in POCL planning) are flaws in the plan generated when a step has unsatisfied (i.e. open)
which does not allow task insertion is to compile a Task primitive or composite. and Geier and Bercher (2011). Inserted tasks can either be left open as in Kambhampati, Mali, and Srivastava (1998) task insertion is allowed to repair open conditions that are not allowed in the context of the problem being addressed). It is not always possible to specify all tasks in a decompositional method (Elkawkagy et al. 2012; Bercher, Keen, and Biundo 2014) and therefore task insertion is unneeded (or at least not allowed in the context of the problem being addressed). It is not always possible to specify all tasks in a decomposition method: some open condition of a task may not have a supplier within the same decompositional hierarchy. Thus, task insertion is allowed to repair open conditions that are left open as in Kambhampati, Mali, and Srivastava (1998) and Geier and Bercher (2011). Inserted tasks can either be primitive or composite. One hybrid planning approach by Elkawkagy et al. (2012) which does not allow task insertion is to compile a Task Decomposition Graph (TDG) composed of edges connecting tasks (primitive or composite) to decomposition methods and vice versa. The TDG is used to guide the planner to shorter plans from an initial task representing the initial partial plan so that the solution is a refinement of the initial task (Bercher, Keen, and Biundo 2014). The criteria for a solution is that all composite tasks are decomposed into primitive tasks, all tasks are fully grounded, and all open conditions are repaired by other tasks in the plan. In our approach, a composite task is fully decomposed and grounded before the entire sub-tree is inserted into a plan to repair an open condition. The decompositions are performed in a pre-caching stage where a max number of decompositional refinements (i.e. step height) is used to cutoff search. The decision points we consider in this work is not which decomposition method to select to decompose a composite task in a plan, but rather which fully ground and decomposed composite task tree to insert to repair an open condition in a plan. Because the sub-tasks in the tree may have open conditions, the TDG will under-estimate the number of decompositions in the plan because inserted tasks may also be composite. A planning domain and problem can (intentionally or inadvertently) require hierarchical depth in the plan to solve the problem. DPOCL-T (Jhala and Young 2010), a variant of DPOCL (Young, Pollack, and Moore 1994) for scheduling camera shots in narrative generation, is tested using a domain that is engineered to promote hierarchical depth by leveraging primary effects. At each “tier” in a multi-level hierarchy, high-level operators have preconditions that can only be fulfilled by other high-level operators at the same level. Thus, a problem whose goal is a primary effect of a high-level action will require the planner to create a high-level plan. In a similar vein, HiPOP (Bechon et al. 2014) uses stages to create plans with composite steps. In the first round, only composite steps are applicable and must be used for as long as possible without expanding them until no composite steps can be used to satisfy preconditions of other steps (or the goal conditions). If no solution can be found, then HiPOP will never proceed to the expansion round. The forward state-space hybrid planner UPS (To, Langley, and Choi 2015) favors states produced by composite steps when that state satisfies elements of the goal formula. Its heuristic counts the number of unmatched goal elements when selecting steps for expansion, greedily selecting composite steps. We suspect that UPS will excessively search through a large space of composite tasks because of this greedy heuristic. In future work, we plan to compare UPS to our own approach.

**The Language of Decompositional Planning**

The formal model of decompositional planning we adopt is taken from DPOCL, a planning system previously developed by Young, Pollack, and Moore (1994). DPOCL builds on POCI planning, which searches in the space of plans to find a partial plan (i.e. a set of steps S, a set of partial ordering relations O over S, and a set of causal links L) with no flaws; all preconditions must be satisfied (no condition may remain open) and no causal links may be threatened (i.e. it should not be possible for any step to be ordered such that}
it potentially undoes a causal link’s protected literal). Open conditions are repaired through *adding or reusing* a plan step and threatened causal links are repaired by *promoting or de-moting* the offending step to come after or before (respectively) the threatened link. For a more thorough introduction to POCL planning, we refer the reader to Weld (1994).

DPOCL has two kinds of steps. A *primitive step* is as in POCL planning. A *composite step* is a step that is decomposable into a partial-plan, called a *sub-plan*. Each composite step is a composite operator type paired with a set of bindings over operator parameters. In this paper, composite steps are associated with a single decomposition method and a set of bindings over decomposition parameters (i.e. a composite operator is rewritten with a specific decomposition method). A step in the sub-plan of a composite step is a *sub-step*. The decomposition specifies constraints over partially defined sub-steps that are useful in our application. A *decompositional link* relates a composite step to a sub-step. The *height* of a composite step is the longest path of decompositional links. Thus a decompositional plan is represented by a tuple \((S, O, L, D)\) where \(D\) is a set of decompositional links. The goal of our planner is to generate a DPOCL plan that solves an input decompositional planning problem and maximizes the ratio of decompositional links to primitive steps.

**Motivating Problem**

Our approach is broadly motivated by the goal of fostering successful human-computer communication. Humans implicitly use grammar to understand and recognize the meaning of speakers (Sperber and Wilson 1987). Strictly, a grammar defines what is a well-defined sequence. In human communication, a grammar may be informal and describe what is an easily recognized pattern of utterances. In human-in-the-loop planning, a planning agent can pose queries to a human operator in the decision of how to continue the planning process (Roth et al. 2004; Schirner et al. 2013). These systems can leverage the way human communicators structure information to more easily recognize user intent and select plans that are not cognitively demanding to parse.

The specific application for a decompositional planner we focus on is the task of directing film. A film director controls various details related to how agents (i.e. character actors) perform actions in an environment and how camera shots should convey those details. Film directors plan out visual details across shots to build up a hierarchically-structured editing pattern. An editing pattern is composed of camera shots, and the sequence that shots cut from one to the next (i.e. transition) affects the way that viewers focus on events. The general principle is that good cuts have matching visual details across shots. The more that shots conform to an editing pattern, the better the aesthetic quality of the sequence.

In computer graphics research, camera control systems find camera sequences that best adhere to a grammar of film (He, Cohen, and Salesin 1996; Christianson et al. 1996; Christie, Olivier, and Normand 2008).

Formulated as a decompositional planning task, the film director (i.e. the planner) selects the content and style of camera shots (primitive tasks) to compose a scene. Each task has preconditions and effects: preconditions correspond to the necessary world conditions for the actor to perform in the world to create the camera shot content, and effects correspond to conditions that change in the world state as a result of the actor’s performance. The more that camera shots adhere to editing patterns (decomposition methods), the better the quality of the result. The decomposition methods impose constraints on camera shot attributes (e.g. scale, angle) and character actions (e.g. orientation, timing) that give rise to good editing transitions. Thus, the quality of the solution is based on the degree that camera shots adhere to editing patterns and not entirely on the length of the plan. Figure 2 presents two short editing sequences: sequence \(A\) has good individual shots but does not adhere to a pattern, whereas sequence \(B\) adheres to an editing pattern and results in matched visual details across shots.

The “film directing” decompositional planning task motivates the depth request:

\[
\text{DEPTHREQUEST} = \max_\pi \frac{|\{s \in S(\pi) : \text{height}(s) > 0\}|}{|\{s \in S(\pi) : \text{height}(s) = 0\}|}
\]

maximize the ratio of decomposition methods to primitive plan steps in a solution to a decompositional planning problem. This ratio corresponds to the percentage of camera shots that adhere to an editing pattern. For each task that the planner inserts to repair an open condition, the planner must decide whether to add hierarchical depth to the plan’s structure, and if so, how much. Figure 1 shows an example grammar tree for film editing. The leaves of the tree are camera shots, with the exception of EP where another editing pattern can be refined into camera shots. To repair an open condition, the planner decides whether to insert a single camera shot (primitive task) or an editing pattern of some depth (composite task), and each may introduce new open condition flaws to establish the prerequisite world conditions.

**Approach**

The approach has two stages: first, a composite step is compiled for every valid sub-tree of a task decomposition graph (a sub-tree is valid just when its leaves are primitive tasks) up to positive non-zero tree height cutoff \(h_{\text{max}}\). These composite tasks are pre-cached in a similar manner to mod-

![Figure 1: Example Film Editing Grammar](image-url)
ern POCL planners (e.g. VHPOP, developed by Younes and Simmons (2003) pre-caches all possible ground instances of problem operators as steps). After grounding the primitive operators (defined to be $h = 0$), grounding continues with composite operators in a bottom-up manner. The first round of composite operators to be grounded ($h = 1$) can only use ground elements and steps of height 0. Inductively, composite steps that have been grounded at height $h$ can serve as sub-steps for grounding composite operators at height $h + 1$ up to $h_{\text{max}}$ (exclusive). Thus, a composite step is defined relative to the height where it has been grounded.

Definition 1 (Composite Step) Represents an instance of a composite operator $\lambda_c$ at height $h$. It is a tuple $\lambda^G_c = (\lambda_c, B, \pi_u, h)$, where $B$ is a set of consistent bindings for the variables in $\lambda_c$ and $\pi_u = (S, O, L, D)$ is a sub-plan for a decomposition of $\lambda_c$ such that $S$ contains at least one step whose height is exactly $h - 1$ and no steps with height greater than $h - 1$.

Our method is a dynamic programming approach to recursive HTN planning that can be used when the maximum depth is known a priori. Composite operators are grounded for every applicable decomposition method (at each height $h = 1$ to $h_{\text{max}}$). The decomposition method is used to calculate the sub-plan and variable bindings that are defined in a composite step. This dynamic programming method was developed by Winer and Young (2017).

The second stage is to solve the planning problem using pre-cached steps. The algorithm follows a classic POP with the addition that adding composite steps leads to the insertion of its pre-cached sub-plan and the decompositional links that point the composite step to its sub-steps.

Plan Search Tree and Plan Depth

Our novel notion of plan depth is based on the reasons that steps are added to the plan. It depends on two structures: decomposition links (as defined earlier) and add-repair arcs. To define add-repair arcs, it is useful to talk about the planning process by way of its search tree. The plan search tree represents the search space of plans and their refinements. A vertex is a pair $(\pi, F_\pi)$ where $\pi$ is a plan and $F_\pi$ is a set of flaws in $\pi$. Edges are of the form $(\pi, F_\pi) \xrightarrow{f, \rho} (\pi', F_{\pi'})$ where $f \in F_\pi$ is the flaw selected for $\pi$ (the parent) and $\pi'$ (the descendant) is the result of repairing flaw $f$ with refinement $\rho$ (one of {add, reuse, promotion, demote}). $F_{\pi'} = \{ \}$.
Algorithm 1 GDPOP

Input: Candidate map \( C_{\text{MAP}} \), threat map \( T_{\text{MAP}} \), initial plan \( \pi_0 \) with steps \( s_0, s_\infty \) (all of depth 0), a function \( \mathcal{F} \) returning flaws \( F \) for a plan, and a plan selection function \( \mathcal{E} \).

Output: A consistent plan with no flaws or failure.

1: \( \mathcal{OL} := \text{openList.push}(\pi_0) \)
2: while \( \mathcal{OL} \) not empty do
3: \( \pi := (S, O, L, D) := \arg \max_{\pi \in \mathcal{OL}} \mathcal{E}(\pi) \)
4: if cycle in \( O \), then skip
5: end if
6: \( F := \mathcal{F}(\pi, T_{\text{MAP}}) \), return \( \pi \) if \(|F| = 0 \)
7: \( f := \text{Flaw-Select}(F, C_{\text{MAP}}, T_{\text{MAP}}) \)
8: if \( f \) is an open condition then
9: \( \mathcal{OL}.\text{push}(\text{Add}(\pi, f, C_{\text{MAP}})) \)
10: \( \mathcal{OL}.\text{push}(\text{Reuse}(\pi, f, C_{\text{MAP}})) \)
11: else if \( f \) is a threatened causal link then
12: \( \mathcal{OL}.\text{push}(\pi_{\text{promote}} \text{ and } \pi_{\text{demote}}) \)
13: end if
14: end while
15: return \( \text{FAIL} \).

Algorithm 2 Add

Input: \( \pi, (s_{\text{need}}, p), C_{\text{MAP}} \)

Output: Expanded \( \pi \)

1: \( \Pi = \emptyset \)
2: for each step \( \lambda \) in \( C_{\text{MAP}}[p] \) do
3: \( s_{\text{new}} := \lambda.\text{clone}() \); \( s_{\text{new}.\text{depth}} = s_{\text{need}.\text{depth}} \)
4: \( \pi' := \pi.\text{clone}() \)
5: \( \text{Insert}(\pi', s_{\text{new}}) \); \( \text{Repair}(\pi', s_{\text{new}}, s_{\text{need}}, p) \)
6: \( \Pi.\text{add}(\pi') \)
7: end for
8: return \( \Pi \)

A Depth-Balancing Plan Selection Heuristic

The evaluation of a plan, which orders plans in the search frontier (open list) in a best-first plan-space search, is defined as \( G(\pi) = g(\pi) + h(\pi) \) where \( g(\pi) \) denotes the plan cost, i.e. the number of steps in the plan, and \( h(\pi) \) denotes the heuristic value, an estimate of the number of steps which need to be inserted into the plan to solve the problem.

We adopt VHPOP’s additive-reuse heuristic function for calculating the heuristic value. The function recursively simulates new open conditions created by inserting actions to make repairs until all open conditions hold initially.
Algorithm 3 Insert

Input: \(s_{new}, \pi\)

1: Add \(s_{new}\) to \(\pi\)
2: if height \((s_{new}) > 0\) then
3:         for each sub-step \(s\) in SUB\(-PLA N(s_{new})\) do
4:             Add decompositional link \(s_{new} \rightarrow s\) to \(\pi\)
5:         \(s\).depth = \(s_{new}\).depth + 1
6:     if \(s\).depth > \(\pi\).depth then
7:         \(\pi\).depth = \(s\).depth
8: end if
9:     Insert \((\pi, s)\)
10: end for
11: end if

The set of open conditions in a plan.

Preliminary Evaluation 1

We developed and compared GDPOP planning with different plan selection heuristics on a generic domain. Our focus is on the performance of the heuristics on each problem.

Table 1: A comparison of plan quality across experimental conditions. All values are averages produced across problems. \(g(\pi)_m\) is the mean cost of solutions, \(S\) is the number of solutions out of 320, \(d(\pi)_m\) is the mean depth of solutions, and \(d_{max}(\pi)_m\) is the mean maximum depth of solutions, where the maximum is defined over solutions for a given planning problem.

<table>
<thead>
<tr>
<th>Condition</th>
<th>(g(\pi)_m)</th>
<th>(S)</th>
<th>(d(\pi)_m)</th>
<th>(d_{max}(\pi)_m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(gPO)</td>
<td>6.89</td>
<td>320</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>(g_{Insert} E^1)</td>
<td>5.89</td>
<td>320</td>
<td>0.61</td>
<td>0.88</td>
</tr>
<tr>
<td>(g_{Insert} E^2)</td>
<td>6.58</td>
<td>320</td>
<td>0.30</td>
<td>1.13</td>
</tr>
<tr>
<td>(g_{Insert} E^3)</td>
<td>9.58</td>
<td>320</td>
<td>1.36</td>
<td>2.75</td>
</tr>
<tr>
<td>(g_Add E^1)</td>
<td>3.34</td>
<td>287</td>
<td>1.01</td>
<td>2.38</td>
</tr>
<tr>
<td>(g_Add E^2)</td>
<td>2.79</td>
<td>152</td>
<td>2.10</td>
<td>3.71</td>
</tr>
<tr>
<td>(g_Add E^3)</td>
<td>3.45</td>
<td>290</td>
<td>1.69</td>
<td>2.88</td>
</tr>
<tr>
<td>(g_Add E^3)</td>
<td>5.91</td>
<td>281</td>
<td>1.98</td>
<td>4.38</td>
</tr>
</tbody>
</table>

In addition to \(E^1\), \(E^2\), \(E^3\) as candidate depth-balancing plan selection functions, we also considered the following algorithm variants:

- \(g_{PO}\) (primitive-only) indicates that only primitive steps are provided as input.
- \(g_{Insert}\) (with composite steps) adds 1 to the cost for every Insert operation.
- \(g_{Add}\) (with composite steps) adds 1 to the cost for every Add operation (and therefore sub-steps are inserted for free).

Methods Python was used to prototype the idea and hypothesis. Subsequently, C# was used as part of the port to the Unity Game Engine for film directing, and demonstrates the run time efficiency more realistically. We ran the prototype implementation of GDPOP on sample problems which vary in number of objects, initial conditions, or goal conditions. The composite operators in the domain are based on filming the classic travel domain. Eight (8) problems were constructed that averaged 2 agents, 2,125 vehicles, 2.5 locations, 1,625 goal conditions. First, the steps are pre-cached. On average, the problems included 45.25 compiled primitive steps (\(\lambda^C\)_1), and 201 compiled composite steps (\(\lambda^C\)_2). The maximum step height is 2. The largest problem (8#) has 4 agents, 4 locations, 2 vehicles, and 2 goal conditions. The planner is run on a 64-bit Windows 7 machine with an Intel i7-3770 CPU at 3.40 GHz and 16 GB of RAM. For each experimental condition, the planner was run until 40 solutions were generated or 400 seconds had elapsed. The conditions of the experiment are \(g_{PO}\), \(g_{Insert} E^1\), and \(g_{Add} E^i\) for \(i = 0 \rightarrow 3\); a total of 9 conditions.

Results We analyzed the performance of the planner on the basis of runtime and nodes expanded. In general, the primitive-only condition is fastest but expands far more nodes (as it should given that composite steps can add many primitive steps in a single node). The \(g_{Insert}\) and \(g_{Insert} E^1\)

1https://github.com/drwiner/PyDPOCL
conditions perform very similarly, expanding less nodes than PO but more than the $g_{add}$ conditions. The results suggest that the $g_{add}$ cost function with the $\mathcal{E}^3$ scoring function expands the fewest nodes on average.

Table 1 shows the quality of solutions for each experimental condition across planning problems. Across the experimental conditions, we observe that cost is weakly sacrificed for plan H-depth, and that $\mathcal{E}^3$ performs best for finding the deepest solutions. Although $g_{add}$ $\mathcal{E}^1$ appears to find the best average depth, it does not find solutions on one of the planning problems in the allotted time (problem 8) and generally struggled on other problems, whereas $g_{add}$ $\mathcal{E}^2$ and $\mathcal{E}^3$ found 40 solutions on this problem before the time cutoff.

### Preliminary Evaluation 2

The first experiment sought to evaluate plan selection criteria that would promote hierarchical depth. However, the depth request ratio was not measured. We ran a new experiment to evaluate the average depth request ratio, to compare against baseline heuristics, and to compare against other selection functions that are potentially depth-balancing. The GDPOP algorithm is reimplemented in C#² as part of the film directing application. Table 2 shows performance and quality averages for the first solution across the 8 problems used in the previous experiment, for each combination of evaluation function and heuristic function, applying a cutoff time of 6,000 milliseconds.

<table>
<thead>
<tr>
<th>Ev</th>
<th>Heu</th>
<th>RT</th>
<th>Op</th>
<th>Exp</th>
<th>Co</th>
<th>D</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0</td>
<td>AR</td>
<td>7/34</td>
<td>174/4</td>
<td>185</td>
<td>4.5</td>
<td>0.3</td>
<td>0.4</td>
<td>8</td>
</tr>
<tr>
<td>E0</td>
<td>OC</td>
<td>346</td>
<td>1087</td>
<td>180</td>
<td>3.3</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>E1</td>
<td>AR</td>
<td>697</td>
<td>1557</td>
<td>165</td>
<td>6.3</td>
<td>1.1</td>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>E1</td>
<td>OC</td>
<td>354</td>
<td>1048</td>
<td>175</td>
<td>2.3</td>
<td>1</td>
<td>1.4</td>
<td>7</td>
</tr>
<tr>
<td>E2</td>
<td>AR</td>
<td>690</td>
<td>1526</td>
<td>161</td>
<td>6.3</td>
<td>1.1</td>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>E2</td>
<td>OC</td>
<td>332</td>
<td>1012</td>
<td>168</td>
<td>3.5</td>
<td>1</td>
<td>1.4</td>
<td>7</td>
</tr>
<tr>
<td>E3</td>
<td>AR</td>
<td>125</td>
<td>360</td>
<td>41.4</td>
<td>6.4</td>
<td>1.1</td>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>E3</td>
<td>OC</td>
<td>191</td>
<td>730</td>
<td>87</td>
<td>6.3</td>
<td>1.1</td>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>E4</td>
<td>AR</td>
<td>720</td>
<td>174/4</td>
<td>185</td>
<td>4.6</td>
<td>0.5</td>
<td>0.4</td>
<td>8</td>
</tr>
<tr>
<td>E4</td>
<td>OC</td>
<td>355</td>
<td>1087</td>
<td>180</td>
<td>3.3</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>E5</td>
<td>AR</td>
<td>29</td>
<td>7</td>
<td>12</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>E5</td>
<td>OC</td>
<td>401</td>
<td>1027</td>
<td>199</td>
<td>9</td>
<td>2</td>
<td>0.4</td>
<td>2</td>
</tr>
<tr>
<td>E6</td>
<td>AR</td>
<td>47</td>
<td>166</td>
<td>21</td>
<td>4.2</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>E6</td>
<td>OC</td>
<td>44</td>
<td>232</td>
<td>28</td>
<td>4.3</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>E6</td>
<td>Zero</td>
<td>1509</td>
<td>5430</td>
<td>906</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>DFS</td>
<td>Zero</td>
<td>11</td>
<td>61</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>0.02</td>
<td>1</td>
</tr>
</tbody>
</table>

**Results**

The plan selection functions that perform best on the depth request are $\mathcal{E}^1$, $\mathcal{E}^2$, and $\mathcal{E}^3$ with the add-reuse heuristic. $\mathcal{E}^3$ also performs well with the number of open conditions heuristic, but overall does not perform as consistently and does not solve all 8 problems. The zero heuristic ($h(\pi) = 0$) typically did not find solutions in time.

### Conclusion

Hybrid planning techniques are popular in theory and practice, but (as also noted by Shivashankar et al. 2016) little effort has been devoted to guiding the search using hierarchical information in a planner-independent way. This planning paradigm is used in domains where hierarchical knowledge characterizes phenomena of interest that is not expressible in non-hierarchical formalisms; for example, the recognition of higher-level concepts on the basis of more primitive event information (Lesh, Rich, and Sidner 1999; Cardona-Rivera and Young 2017) or the generation of narratives where hierarchies represent communicative patterns and story plots (Winer and Young 2017). With our work, search in these domains can generate deep, low-cost solutions in a more principled manner.

One of the key obstacles we overcome with our approach is avoiding excessive search on the hierarchical depth attribute. A naive strategy is to simply discount composite tasks that add depth. However, this strategy causes the planner to always search through the space of plans that insert composite tasks before considering primitive tasks. The magnitude of the discount determines how excessively the planner searches in the direction of inserting composite tasks to repair open conditions. At some point, the discount is outweighed by the size of the plan and backtracking occurs such that shallower tasks are considered by the planner to make the same repairs. Although this approach prioritizes maximum depth, it is slow because it behaves like brute force search on the hierarchical depth attribute to find the best depth/cost ratio.

On the other hand, if the least-cost plans are always expanded first and no discount is offered for inserting tasks that add hierarchical depth (as in the standard case), then the planner will start shallow and insert composite tasks just when it is strictly more efficient. A depth-balanced approach neither over-discounts nor under-discounts hierarchical depth. In this work, we formulate a dynamic discount for depth that exploits the hierarchical depth of the sub-trees of composite tasks to guide the planner to “deep” solutions. When a plan is “shallow” (hierarchically), deep composite tasks are discounted, but as the plan becomes “deeper”, this discount recedes. We introduced a new way to measure plan depth, and used this measurement as part of a search strategy for decompositional planning where deep solutions are preferred. Our preliminary evidence supports a claim that our dynamic discount is useful for achieving a depth-balanced approach. Our method may also be useful for state-space decompositional planning, which has historically suffered from similar issues.

Importantly, we tested our scoring functions on a single planning domain. To verify that our results are generalizable, we also need to test the scoring functions with different conditions.
hierarchical domains. The effects of differently structured hierarchical knowledge is less clear than primitive-only domains (Chrpa, McCluskey, and Osborne 2015); an evaluation with such differently structured hierarchical knowledge warrants a follow-up investigation that is beyond our scope here. This study would compare domains where the amount of decomposition knowledge provided as input is controlled, and its effects on the search for deep solutions is examined.

References


