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) preconditions and repaired by ensuring that the preconditions are established by some earlier action; prior work has focused on selecting a decomposition method for a composite task. State-of-the-art heuristics that exploit task decomposition (e.g. Minimal Modification Effort by Bercher, Keen, and Biundo 2014) are not of use here. These heuristics bias the planner to refine the initial plan to primitive tasks in the least number of refinements (as they should do to generate the shortest plans). In our methodology, each composite task is fully decomposed before it is inserted into a plan. Thus, the hierarchical depth of a composite task is known at the moment of insertion, and its entire decomposition refinement tree (its sub-plan) is inserted into the plan. The main question we ask is *how should the planner best add hierarchical depth to the plan during task insertion to fulfill the depth request*? Our approach is to have the planner leverage the known depth of the decomposition tree of inserted composite tasks. Incentivizing the insertion of composite tasks comes with a tradeoff: the planner must search a much larger space of plans and significantly reduce efficiency. The key problem we address is how to have the planner incentivize inserting composite tasks without a significant tradeoff to efficiency.

Contributions We introduce a plan selection function that reifies the tradeoff between inserting deep and shallow tasks within hybrid planning with task insertion. This function incentivizes selecting plans with composite tasks on the search frontier and depends on a novel way to measure plan depth. We evaluated our function's effectiveness on the basis of runtime performance and solution depth on test problems, and demonstrate we can find deep, low-cost solutions while avoiding brute-force search on hierarchical depth.

Related Work

Our planning task is an instance of hybrid planning with task insertion. Terminologically, we use the term "decompositional planning" to describe what our planner is doing: solving a problem given in terms of goal literals as in classical planning, but wherein abstract tasks can be inserted as well (i.e. there are no initial abstract tasks, but rather solving proceeds from an initial dummy plan given by the initial conditions and the goal literals). While we recognize that there are planning variants that differ terminologically and semantically (e.g. "hybrid planning" by Kambhampati, Mali, and Srivastava 1998), reconciling all variants is beyond the scope of this paper. In some variants, all (causally necessitated) tasks are specified as part of a decomposition 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 precaching 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 POCL 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 *demoting* 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 decomposition links. Thus a decompositional plan is represented by a tuple $\langle S, O, L, D\rangle$ 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 decomposition 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

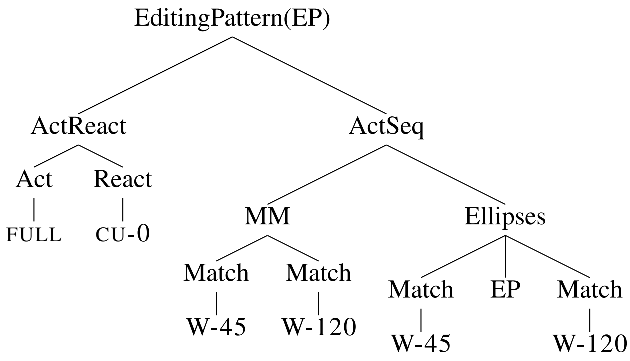

Figure 1: Example Film Editing Grammar

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 resulting film. 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:

$$\textsc{DepthRequest} = \max_{\pi} \frac{|\{s \in S(\pi) : height(s) > 0\}|}{|\{s \in S(\pi) : height(s) = 0\}|}$$

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 $h_{\max}$. These composite tasks are pre-cached in a similar manner to mod-

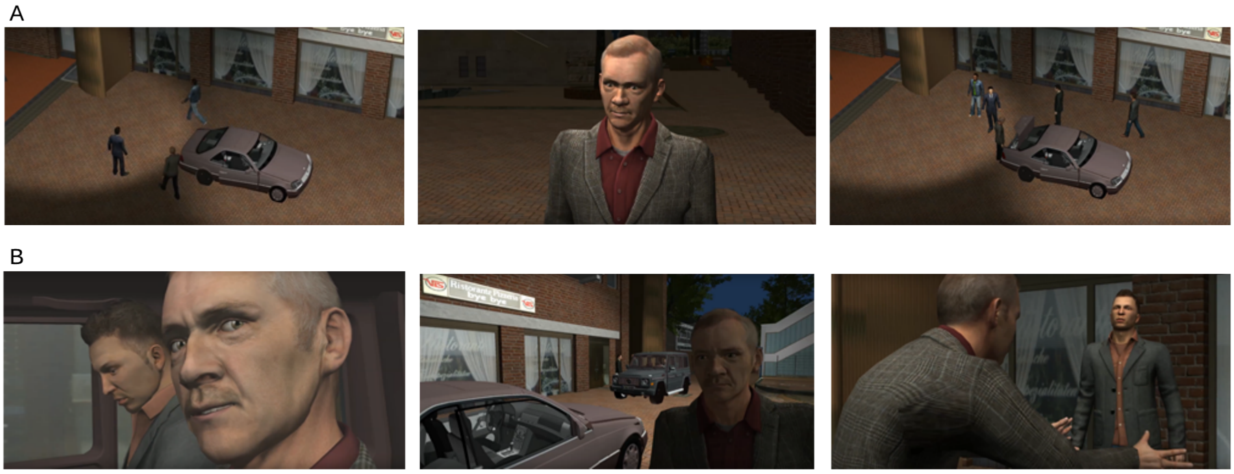

Figure 2: Sequences **A** and **B** are camera shots sampled from different versions of the same film created by a film director. **A** has good individual shots but sequences in the film are discontinuous and do not follow editing patterns. **B** contains sequences developed to adhere to editing patterns. This results in matched visual details across shots and improved focus on events.

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 $h_{\max}$ (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 λ_c at height h. It is a tuple $\lambda_c^G = \langle \lambda_c, B, \pi_u, h \rangle$, where B is a set of consistent bindings for the variables in $V \in \lambda_c$ and $\pi_u = \langle \mathcal{S}, \mathcal{O}, \mathcal{L}, \mathcal{D} \rangle$ is a sub-plan for a decomposition of λ_c such that $\mathcal{S}$ 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 $h_{\max}$). 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.

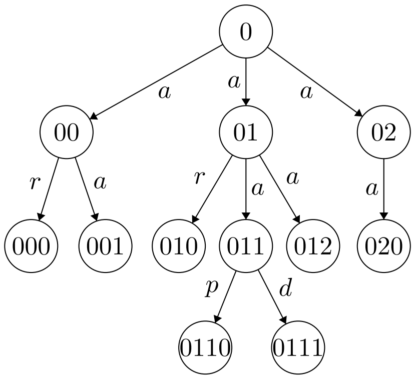

Figure 3: A plan search tree. Nodes are plans and edges are refinements to the plan. The edge label represents the repair method; one of $a$dd, $r$euse, $p$romotion, and $d$emotion.

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 (π, F_π) where π is a plan and F_π is a set of flaws in π. Edges are of the form $(\pi, F_\pi) \xrightarrow{(f,\rho)} (\pi', F_{\pi'})$ where $f \in F_\pi$ is the flaw selected for π (the parent) and π' (the descendant) is the result of repairing flaw f with refinement ρ (one of $\{\text{add}, \text{reuse}, \text{promote}, \text{demote}\}$). $F_{\pi'} =$

$F_\pi - f \cup F_\rho$ where F_ρ are flaws detected in π' after ρ. A **path** is a sequence of nodes and edges connecting a vertex with a descendant.

Let q be a path on a plan search tree, and $\pi' = \langle S, O, L, D\rangle$ be a plan on a leaf vertex. An **add-repair arc** exists between two steps $s_i, s_j \in S$ from s_j to s_i just when there exists a causal link of the form $s_i \xrightarrow{p} s_j \in L$ such that s_i was added to S through an edge $(\pi, F_\pi) \xrightarrow{(\langle s_i, p\rangle,\ \text{add})} (\pi', F_{\pi'})$ (i.e. an open condition p for step s_i repaired through adding a step) for some parent π.

Definition 2 (Deep Path and Plan H-depth) *A* **deep path** *δ on a plan π is a traversal of decomposition links and add-repair arcs in the plan; the length of a deep path $len(\delta)$ is the number of decomposition links in the traversal. The* **H-depth** *of π is the longest deep path on π.*

When steps are reused to repair an open condition, no plan H-depth is added. H-depth is recalculated only after adding steps and requires only constant time to track.

Algorithm

The *Ground Decompositional Partial Order Planning algorithm* (GDPOP) presented in Algorithm 1 is a propositional POCL planner that uses composite steps whose decomposition refinements are already performed. GDPOP takes as input a set of ground primitive steps Λ_p^G, a set of ground composite steps Λ_c^G, the plan-space initial plan, a **candidate map**, which maps every open condition of every step $\langle s_{\text{need}}, p\rangle$ to every step with an effect p), and a **threat map**, which maps every open condition of every step $\langle s_{\text{need}}, p\rangle$ to every step with an effect $\neg p$. Initially, the plan has depth 0 and steps are assigned a depth of 0. When a composite step is *added* to repair an open condition (*a la* Algorithm 2), sub-steps are *inserted* and depth is propagated (*a la* Algorithm 3).

At each iteration, GDPOP identifies all flaws in the plan under consideration, selects a flaw to fix, and adds to the search fringe all the plans that represent every way in which the flaw could be repaired. Selecting which flaw to repair affects the order that plans are visited in the search space (Younes and Simmons 2003); we adopted a simplified version which is

1. open conditions that are static (are unchangeable given the problem's operators)
2. threatened causal link flaws
3. open conditions that hold initially
4. most unsafe open conditions first (with at least one potential risk detected using the threat map)
5. open conditions with at least 1 candidate for reuse (detected using the candidate map, sorted by random hash code)
6. open conditions with no option for reuse

A risk s_{risk} in a plan π for an open condition of the form $\langle s_{need}, pre\rangle$ is a step that may undo the open condition because $s_{risk} \in T_{\text{MAP}}[pre]$ and $\nexists s_{need} \prec s_{risk} \in O(\pi)$. The number of risks for an open condition is set at the moment it is created (we did not update the number of risks when inserting new steps or remove risks when a step is no longer a risk). A candidate s_{cndt} in a plan for an open condition is a step in $C_{\text{MAP}}[pre]$ that can be ordered before s_{need}. Similarly, no update is made to update the number of candidates after the open condition is created.

Algorithm 1 `GDPOP`

Input: Candidate map C_{MAP}, threat map T_{MAP}, initial plan π_0 with steps s_0, s_∞ (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.

$\mathcal{OL} := openList$.push(π_0)
while $\mathcal{OL}$ not empty **do**
  $\pi = \langle S, O, L, D\rangle := \arg\max_{\pi \in \mathcal{OL}} \mathcal{E}(\pi)$
  if cycle in O, **then** skip
  end if
  $F := \mathcal{F}(\pi, T_{\text{MAP}})$, **return** π if $|F| = 0$
  $f := Flaw\text{-}Select(F, C_{\text{MAP}}, T_{\text{MAP}})$
  if f is an open condition **then**
    $\mathcal{OL}$.push(`Add`(π, f, C_{MAP}))
    $\mathcal{OL}$.push($Reuse(\pi, f, C_{\text{MAP}})$)
  else if f is a threatened causal link **then**
    $\mathcal{OL}$.push(π_{promote} and π_{demote})
  end if
end while
return FAIL

Algorithm 2 `Add`

Input:π, $\langle s_{need}, p\rangle$, C_{MAP};
Output: Expanded π

$\Pi = \varnothing$
for each step λ in $C_{\text{MAP}}[p]$ **do**
  $s_{new} := \lambda.clone()$; $s_{new}.depth = s_{need}.depth$
  $\pi' := \pi.clone()$
  `Insert`(π', s_{new}); $Repair(\pi', s_{new}, s_{need}, p)$
  $\Pi.add(\pi')$
end for
return Π

We detail the plan refinement operations that use our novel constructs (i.e. Algorithm 2, Algorithm 3) and leave un-specified the plan refinement operations that are taken from standard POCL planning (i.e. $Reuse$ in Algorithm 1, $Repair$ in Algorithm 2, and the calculation of plans π_{promote} and π_{demote} that promote and demote steps that threaten causal links, respectively).

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 $\mathcal{E}^0(\pi) = g(\pi) + h(\pi)$ 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. Nota-

Algorithm 3 **`Insert`**

Input: π, s_{new}

Add s_{new} to π
if $height(s_{new}) > 0$ **then**
  for each sub-step s in SUB-PLAN(s_{new}) **do**
    Add decompositional link $s_{new} \rightharpoondown s$ to π
    $s.depth = s_{new}.depth + 1$
    if $s.depth > \pi.depth$ **then**
      $\pi.depth = s.depth$
    end if
    `Insert`(π, s)
  end for
end if

tion: C_{MAP} is a candidate map, which maps each precondition to the set of actions that can repair it, and $\mathcal{OC}$ returns the set of open conditions in a plan.

$$h^r_{add}(\pi) = \sum_{\xrightarrow{q} a_i \in \mathcal{OC}(\pi)} \begin{cases} 0 & \text{if } \exists a_j \in S \cap C_{\text{MAP}}[q] \\ & \text{and } a_i \prec a_j \notin O \\ h_{add}(q) & \text{otherwise} \end{cases}$$

$$h_{add}(q) = \begin{cases} 0 & \text{if } q \text{ holds initially} \\ \min_{a \in C_{\text{MAP}}[q]} h_{add}(a) & \text{if } C_{\text{MAP}}[q] \neq \varnothing \\ \infty & \text{otherwise} \end{cases}$$

$$h_{add}(a) = 1 + \sum_{p \in pre(a)} h_{add}(p)$$

In addition to cost g and heuristic h, we introduce a value d that corresponds to H-depth (as in Definition 2). We experimented with several ways to discount this value:

1. $\mathcal{E}^1(\pi) = g(\pi) + h^r_{add}(\pi) - d(\pi)$
2. $\mathcal{E}^2(\pi) = g(\pi) + h^r_{add}(\pi) - \log_2(d(\pi) + 1)$
3. $\mathcal{E}^3(\pi) = \frac{g(\pi)}{1+\log_2(d(\pi)+1)} + h^r_{add}(\pi)$
4. $\mathcal{E}^4(\pi) = g(\pi) + h^r_{add}(\pi) - |\{s \in S(\pi) : height(s) > 0\}|$
5. $\mathcal{E}^5(\pi) = \frac{g(\pi)^2}{1+|\{s \in S(\pi) : height(s) > 0\}|} + h^r_{add}(\pi)$
6. $\mathcal{E}^6(\pi) = |\{s \in S(\pi) : height(s) > 0\}| + h^r_{add}(\pi)$

In $\mathcal{E}^2$ and $\mathcal{E}^3$, the logarithm helps diminish the extent to which plan H-depth drives the score. The gist of these functions is that composite steps are considered less valuable as repairs the deeper the plan gets, and therefore protects against over-incentivizing depth. $\mathcal{E}^2$ subtracts depth and $\mathcal{E}^3$ divides by it; we compared them to evaluate the sensitivity of the search to how depth is factored into the score. We hypothesized that $\mathcal{E}^3$ would find the best depth-request ratios.

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 experiment conditions. All values are averages produced across problems; $g(\pi)_{\text{m}}$ is the mean cost of solutions, S is the number of solutions out of 320, $d(\pi)_{\text{m}}$ is the mean depth of solutions, and $d_{\text{max}}(\pi)_{\text{m}}$ is the mean maximum depth of solutions, where the maximum is defined over solutions for a given planning problem.

	$g(\pi)_{\text{m}}$	S	$d(\pi)_{\text{m}}$	$d_{\text{max}}(\pi)_{\text{m}}$
g_{PO}	6.89	320	0.00	0.00
$g_{Insrt}\ \mathcal{E}^0$	5.89	320	0.61	0.88
$g_{Insrt}\ \mathcal{E}^1$	6.58	320	0.30	1.13
$g_{Insrt}\ \mathcal{E}^3$	9.58	320	1.36	2.75
$g_{Add}\ \mathcal{E}^0$	3.34	287	1.01	2.38
$g_{Add}\ \mathcal{E}^1$	2.79	152	2.10	3.71
$g_{Add}\ \mathcal{E}^2$	3.45	290	1.69	2.88
$g_{Add}\ \mathcal{E}^3$	**5.91**	**281**	**1.98**	**4.38**

In addition to $\mathcal{E}^1, \mathcal{E}^2, \mathcal{E}^3$ 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_{Insrt} (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[1]. 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 (λ^G_p), and 201 compiled composite steps (λ^G_c). 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}, $g_{Insrt}\ \mathcal{E}^i$, and $g_{Add}\ \mathcal{E}^i$ for $i = 0 - 3$; 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_{Insrt} and $g_{Insrt}\ \mathcal{E}^1$

[1] `https://github.com/drwiner/PyDPOCL`

Table 2: Plan Evaluation Results (Preliminary Evaluation 2). *Legend*: Ev: plan scoring function, Heu: heuristic, AR: "AddReuse", OC: "number of open conditions", RT: runtime (milliseconds), Op: "number of nodes opened", Exp: "number of nodes expanded", Co: cost, D: number of decomposition methods, R: depth request ratio, S: number of problems solved out of 8. The "zero" heuristic was also run for each evaluation, but not included in cases where no problems are solved. We also ran a breadth-first search which did not solve any problems.

Ev	Heu	RT	Op	Exp	Co	D	R	S
E0	AR	734	1744	185	4.5	0.3	0.4	8
E0	OC	346	1087	180	3.3	0	0	7
E1	**AR**	**697**	**1557**	**165**	**6.3**	**1.1**	**1.5**	**8**
E1	OC	334	1044	175	5.3	1	1.4	7
E2	**AR**	**690**	**1526**	**161**	**6.3**	**1.1**	**1.5**	**8**
E2	OC	332	1012	168	5.3	1	1.4	7
E3	**AR**	**123**	**360**	**41.4**	**6.4**	**1.1**	**1.5**	**8**
E3	**OC**	**191**	**730**	**87**	**6.3**	**1.1**	**1.5**	**8**
E4	AR	720	1744	185	4.6	0.3	0.4	8
E4	OC	333	1087	180	3.3	0	0	7
E5	AR	29	77	12	5	1	0.6	3
E5	OC	401	1027	199	9	2	0.4	2
E6	AR	47	166	21	4.2	0	0	6
E6	OC	44	232	28	4.3	0	0	8
E6	Zero	1509	5430	966	3	0	0	5
DFS	Zero	11	61	13	8	1	0.02	1

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.

[2]`https://github.com/drwiner/gdpop`

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 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

Bechon, P.; Barbier, M.; Infantes, G.; Lesire, C.; and Vidal, V. 2014. HiPOP: Hierarchical Partial-Order Planning. In *Proceedings of the 7th European Starting AI Researcher Symposium at the 21st European Conference on Artificial Intelligence*, 51–60.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid Planning Heuristics Based on Task Decomposition Graphs. In *Proceedings of the 7th Annual Symposium on Combinatorial Search*, 35–43.

Bordwell, D.; Thompson, K.; and Smith, J. 1997. *Film Art: An Introduction*. McGraw-Hill New York.

Cardona-Rivera, R. E., and Young, R. M. 2017. Toward combining domain theory and recipes in plan recognition. In *Proceedings of the Plan, Activity, and Intent Recognition Workshop at the 31st AAAI*, 796–803.

Christianson, D. B.; Anderson, S. E.; He, L.-w.; Salesin, D. H.; Weld, D. S.; and Cohen, M. F. 1996. Declarative camera control for automatic cinematography. In *AAAI/IAAI, Vol. 1*, 148–155.

Christie, M.; Olivier, P.; and Normand, J.-M. 2008. Camera control in computer graphics. *Computer Graphics Forum* 27(8):2197–2218.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2015. On the Completeness of Replacing Primitive Actions with Macro-actions and its Generalization to Planning Operators and Macro-operators. *AI Communications* 29(1):163–183.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving Hierarchical Planning Performance by the Use of Landmarks. In *Proceedings of the 26th AAAI*, 1763–1769.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and Expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1123–1128.

Garoufi, K. 2014. Planning-based models of natural language generation. *Language and Linguistics Compass* 8(1):1–10.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd IJCAI*, 1955–1961.

He, L.-w.; Cohen, M. F.; and Salesin, D. H. 1996. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 217–224. ACM.

Jhala, A., and Young, R. M. 2010. Cinematic visual discourse: Representation, generation, and evaluation. *IEEE Txn on Comp. Intelligence and AI in Games* 2(2):69–81.

Kambhampati, S.; Mali, A.; and Srivastava, B. 1998. Hybrid Planning for Partially Hierarchical Domains. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 882–888.

Lee-Urban, S. M. 2012. *Hierarchical Planning Knowledge for Refining Partial-Order Plans*. Ph.D. Dissertation, Lehigh University.

Lesh, N.; Rich, C.; and Sidner, C. L. 1999. Using plan recognition in human-computer collaboration. In *Proceedings of the 7th International Conference on User Modeling*, 22–32.

Prince, G. 2003. *A Dictionary of Narratology*. University of Nebraska Press.

Roth, E. M.; Hanson, M. L.; Hopkins, C.; Mancuso, V.; and Zacharias, G. L. 2004. Human in the loop evaluation of a mixed-initiative system for planning and control of multiple uav teams. In *Proceedings of the Human Factors and Ergonomics Society 48th Annual Meeting*, 280–284.

Schirner, G.; Erdogmus, D.; Chowdhury, K.; and Padir, T. 2013. The future of human-in-the-loop cyber-physical systems. *Computer* 46(1):36–45.

Shivashankar, V.; Alford, R.; Roberts, M.; and Aha, D. W. 2016. Cost-optimal algorithms for planning with procedural control knowledge. In *Proceedings of the 22nd European Conference on Artificial Intelligence*, 1702–1703.

Sperber, D., and Wilson, D. 1987. Précis of relevance: Communication and cognition. *Behavioral and brain sciences* 10(4):697–710.

To, S. T.; Langley, P.; and Choi, D. 2015. A Unified Framework for Knowledge-Lean and Knowledge-Rich Planning. In *Proceedings of the 3rd Annual Conference on Advances in Cognitive Systems*.

Weld, D. 1994. An Introduction to Least Commitment Planning. *AI Magazine* 15(4):27.

Winer, D. R., and Young, R. M. 2017. Merits of Hierachical Story and Discourse Planning with Merged Languages. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Younes, H. L., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *JAIR* 20:405–430.

Young, R. M.; Ware, S.; Cassell, B.; and Robertson, J. 2013. Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives. *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative* 37(1-2):41–64.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and causality in partial-order planning. In *Proceedings of the Conference on Artificial Intelligence Planning Systems*, 188–194.