Cost-Optimal Algorithms for Hierarchical Goal Network Planning: A Preliminary Report

Vikas Shivashankar[1] vikas.shivashankar@knexusresearch.com
Ron Alford[2] ralford@mitre.org
Mark Roberts[3] mark.roberts.ctr@nrl.navy.mil
David W. Aha[4] david.aha@nrl.navy.mil

[1]Knexus Research Corporation, National Harbor, MD
[2]MITRE, McLean, VA
[3]NRC Postdoctoral Fellow, Naval Research Laboratory, Code 5514, Washington DC
[4]Naval Research Laboratory, Code 5514, Washington DC

Abstract

There is an impressive body of work in developing search heuristics and other reasoning algorithms to guide domain-independent planning algorithms towards (near-) optimal solutions. However, very little effort has been expended in developing analogous techniques to guide search towards high-quality solutions in domain-configurable planning formalisms, such as HTN planning. In lieu of such techniques, the domain-specific knowledge often needs to provide the necessary search guidance to the planning algorithm; this not only imposes a significant burden on the domain author, but can also result in brittle or error-prone domain models.

This work attempts to address this gap by extending recent work on a new hierarchical planning formalism called Hierarchical Goal Network (HGN) Planning to develop the ***H**ierarchically-**Op**timal **G**oal **D**ecomposition **P**lanner (*HOpGDP*)*, a HGN planning algorithm that computes *hierarchically-optimal* plans. HOpGDP is guided by h_{HL}, a new HGN planning heuristic that extends existing admissible landmark-based heuristics from Classical Planning in order to compute admissible cost estimates for HGN planning problems. Preliminary experimental results show that our planner compares favorably to the current state-of-the-art.

1 Motivation and Background

A primary research focus in AI planning is developing efficient search heuristics and auxiliary reasoning techniques that can help the planner find high-quality plans efficiently. Formalisms for automated planning developed in the literature to represent and solve planning problems broadly fall into either *domain-independent planning* or *domain-configurable planning*. Domain-independent planning formalisms, such as *classical planning* requires that the users only provide models of the base actions executable in the domain. In contrast, domain-configurable planning formalisms such as *Hierarchical Task Network (HTN) Planning* allow users to supplement the action models with additional domain-specific knowledge structures that increases the expressivity and scalability of the planning systems.

An impressive body of work exploring search heuristics that has helped scale up search for high-quality solutions in classical planning. Concretely, search heuristics such as the relaxed planning graph heuristic (Hoffmann and Nebel 2001), landmark generation algorithms (Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010), and landmark-based heuristics (Richter and Westphal 2010; Karpas and Domshlak 2009) dramatically improved optimal and anytime planning algorithms by guiding search towards (near-) optimal solutions to planning problems.

Yet, relatively little effort has been devoted to develop analogous techniques to guide search towards high-quality solutions in domain-configurable planning systems. In lieu of such search heuristics, domain-configurable planners often require additional domain-specific knowledge to provide the necessary search guidance. This requirement not only imposes a significant burden on the user, but also sometimes leads to brittle or error-prone domain models. To address this gap, this paper leverages recent work on a new hierarchical planning formalism called *Hierarchical Goal Network* (HGN) Planning (Shivashankar et al. 2012; 2013), which combines the hierarchical structure of HTN planning with the goal-based nature of classical planning.

In this paper, we develop the ***H**ierarchically-**Op**timal **G**oal **D**ecomposition **P**lanner (*HOpGDP*)*, a HGN planning algorithm that uses admissible heuristic estimates to generate *hierarchically-optimal* plans, i.e plans that are both valid and optimal with respect to the given hierarchical knowledge. In particular, our contributions are as follows:

- **Admissible Heuristic**: We present h_{HL}(**H**GN **L**andmark heuristic), a HGN planning heuristic that extends landmark-based admissible heuristics from classical planning to derive admissible cost estimates for HGN planning problems. To the best of our knowledge, h_{HL} is the first admissible heuristic for hierarchical planning[1].
- **Optimal Planning Algorithm**: We describe HOpGDP, an A* search algorithm that uses h_{HL} to generate *hierarchically-optimal* plans.
- **Preliminary Experimental Results**: We provide preliminary experimental evidence showing that HOpGDP outperforms optimal classical planners due to its ability to exploit hierarchical knowledge. We also see that h_{HL} provides useful search guidance by showing that it compares favorably both in terms of runtime and nodes explored to $\text{HOpGDP}_{\text{blind}}$, the variant of HOpGDP that uses the trivial heuristic $h = 0$, despite a significant computation overhead.

[1]We are of course not counting the trivial heuristic of $h = 0$.

2 Preliminaries

In this section we detail the classical planning model, review how landmarks are constructed for classical planning and an admissible landmark-based heuristic h_L, and describe goal network planning using examples from assembly planning.

2.1 Classical Planning

We define a *classical planning domain* $D_{classical}$ as a finite-state transition system in which each state s is a finite set of ground atoms of a first-order language L, and each action a is a ground instance of a planning operator o. A planning operator is a 4-tuple $o = (\text{head}(o), \text{precond}(o), \text{effects}(o), \mathsf{cost}(o))$, where precond($o$) and effects($o$) are conjuncts of literals called o's *preconditions* and *effects*, and head(o) includes o's *name* and *argument list* (a list of the variables in precond(o) and effects(o)). cost(o) represents the non-negative cost of applying operator o.

Actions. An action a is executable in a state s if $s \models \text{precond}(a)$, in which case the resulting state is $\gamma(a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$, where $\text{effects}^+(a)$ and $\text{effects}^-(a)$ are the atoms and negated atoms, respectively, in effects(a). A plan $\pi = \langle a_1, \ldots, a_n \rangle$ is executable in s if each a_i is executable in the state produced by a_{i-1}; and in this case $\gamma(s, \pi)$ is the state produced by executing the entire plan. If π and π' are plans or actions, then their concatenation is $\pi \circ \pi'$.

We define the *cost* of $\pi = \langle a_1, \ldots, a_n \rangle$ as the sum of the costs of the actions in the plan, i.e. $\mathsf{cost}(\pi) = \sum_{i \in \{1 \ldots n\}} a_i$.

2.2 Generating Landmarks for Classical Planning

There are several landmark generation algorithms suggested in the literature, such as (Hoffmann, Porteous, and Sebastia 2004) and LAMA (Richter and Westphal 2010). The general approach used in generating sound landmarks is to relax the planning problem, generate sound landmarks for the relaxed version, and then use those for the original planning problem. In this paper, we use LAMA's landmark generation algorithm, which uses relaxed planning graphs and domain-transition graphs in tandem to generate landmarks.

2.3 h_L: an Admissible Landmark-based Heuristic

We provide some background on h_L, the landmark-based admissible heuristic for classical planning problems proposed by Karpas and Domshlak (Karpas and Domshlak 2009) that we will be using in our heuristic.

Consider a classical planning problem $P = (D, s_0, g)$ and a landmark graph $LG = (L, Ord)$ computed using any off-the-shelf landmark generation algorithms (e.g., LAMA (Richter and Westphal 2010)). Then, we can define $\mathsf{Unreached}(L, s, \pi) \subseteq L$ to be the set of landmarks that need to be achieved from s onwards, assuming we got to s using the plan π. Note that $\mathsf{Unreached}(L, s, \pi)$ is path-dependent: it can vary for the same state when reached by different paths. It can be computed as follows:

$$\mathsf{Unreached}(L, s, \pi) = L \setminus (\mathsf{Accepted}(L, s, \pi) \setminus \mathsf{ReqAgain}(L, s, \pi))$$

where $\mathsf{Accepted}(L, s, \pi) \subseteq L$ is the set of landmarks that were true at some point along π. $\mathsf{ReqAgain}(L, s, \pi) \subseteq L$ is the set of landmarks that were accepted but are required again; an accepted landmark l is required again if (1) it does not hold true in s, and (2) it is greedy-necessarily ordered before another landmark l' in L that is not accepted.

Karpas and Domshlak show that it is possible to partition the costs of the actions A in D over the landmarks in $\mathsf{Unreached}(L, s, \pi)$ to derive an admissible cost estimate for the state s as follows: let $cost(\phi)$ be the cost assigned to the landmark ϕ, and $cost(a, \phi)$ be the portion of a's cost assigned to ϕ. Furthermore, let us suppose these costs satisfy the following set of inequations:

$$\forall a \in A : \sum_{\phi \in \mathsf{Unreached}(a|L,s,\pi)} cost(a, \phi) \leq \mathsf{cost}(a)$$
$$\forall \phi \in \mathsf{Unreached}(L, s, \pi) : cost(\phi) \leq \min_{a \in ach(\phi|s,\pi)} cost(a, \phi) \quad (1)$$

where $ach(\phi|s, \pi) \subseteq A$ is the set of possible achievers of ϕ along any suffix of π, and $ach(a|L, s, \pi) = \{\phi \in \mathsf{Unreached}(L, s, \pi) | a \in ach(\phi|s, \pi)\}$.

Informally, what these equations are encoding is a scheme to partition the cost of each action across all the landmarks it could possibly achieve, and assigns to each landmark ϕ a cost no more than the minimum cost assigned to ϕ by all its achievers. Given this, they prove the following useful claim:

Lemma 1. *Given a set of action-to-landmark and landmark-to-action costs satisfying Eqn. 1,* $h_L(L, s, \pi) = cost(\mathsf{Unreached}(L, s, \pi)) = \sum_{\phi \in \mathsf{Unreached}(L,s,\pi)} cost(\phi)$ *is an admissible estimate of the optimal plan cost from* s.

Note that the choice of exactly how to do the cost-partitioning is left open. One of the schemes Karpas and Domshlak propose is an *optimal cost-partitioning* scheme that uses an LP solver to solve the constraints in Eqn. 1 with the objective function $\max \sum_{\phi \in L(s,\pi)} cost(\phi)$. This has the useful property that given two sets of landmarks L and L', if $L \subseteq L'$, then $h_L(L, s, \pi) \leq h_L(L', s, \pi)$. In other words, the more landmarks you provide to h_L, the more informed the heuristic estimate.

2.4 Goal Networks and HGN Methods

We extend the definitions of (Shivashankar et al. 2012) of HGN planning to work with partially-ordered sets of goals, which we call a goal network.

A *goal network* is a way to represent the objective of satisfying a partially ordered multiset of goals. Formally, it is a pair $gn = (T, \prec)$ such that:

- T is a finite nonempty set of nodes;
- each node $t \in T$ contains a *goal* g_t that is a DNF (disjunctive normal form) formula over ground literals;
- $\prec$ is a partial order over T.

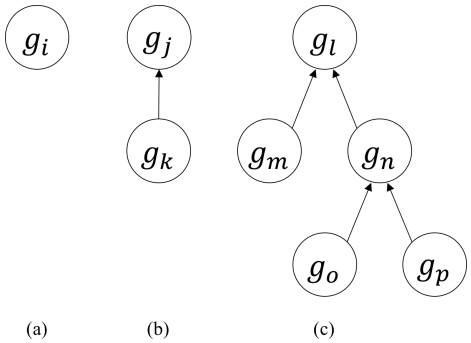

Figure 1: Three generic goal networks we use for examples of the various relationships within a goal network.

We will provide examples of both generic and concrete goal networks. Figure 1 shows three generic goal networks. Each subfigure is itself a goal network denoted gn_a, gn_b, gn_c. Directed arcs indicate a subgoal pair (e.g., (g_k, g_j) from gn_b) such that the first goal must be satisfied before the second goal. Consider the network gn_b where g_k is a subgoal of g_j, then $gn_b = (\{g_j, g_k\}, (g_k \prec g_j))$. Network gn_c shows a partial ordering, where $(\{g_m, g_n\} \prec g_l)$. Similarly, $(\{g_o, g_p\} \prec g_n)$ and this implies both must occur before g_l. Consider a network gn_x that is composed of gn_a and gn_b. Then $gn_x = (\{g_i, g_j, g_k\}, g_k \prec g_j)$. Note that gn_x is a partially ordered forest of goal networks.

Figure 2 shows a concrete goal network for an automated manufacturing domain. $\mathsf{joined}(x, y)$ denotes the goal of assembling the parts x and y together, while $\mathsf{at}(x, loc)$ represents the goal of getting x to location loc. In this goal network, the two goals $\mathsf{joined}(p_2, p_1)$ and $\mathsf{joined}(p_3, p_1)$ are unordered with respect to one another. Furthermore, $\mathsf{joined}(p_2, p_1)$ has three subgoals that need to be achieved before achieving it, i.e the goals of getting the parts p_1, p_2 and the *tool* to the assembly table. These subgoals are also unordered with respect to one another, indicating that the goals can be accomplished in any order.

HGN Methods An *HGN method* m is a 4-tuple (head(m), goal(m), precond(m), network(m)) where the head head(m) and preconditions precond(m) are similar to those of a planning operator. goal(m) is a conjunct of literals representing the goal m decomposes. network(m) is the goal network that m decomposes into. By convention, network(m) has a last node t_g containing the goal goal(m) to ensure that m accomplishes its own goal.

Figure 3 describes the goal network that the deliver-obj method decomposes a goal into. This method is relevant to $\mathsf{at}(x, loc)$ goals (since that's the last node), and its preconditions are precond(deliver-obj) $= \{\neg\mathsf{reserved}(agent), \mathsf{can\text{-}carry}(agent, p) \ldots\}$.

Whether a node has predecessors impacts the kinds of operations we allow. We refer to any node in a goal network gn having no predecessors as an *unconstrained* node of gn, otherwise the node is *constrained*. The constrained nodes of Figure 1 include g_j, g_l, g_n and the remaining are unconstrained. The unconstrained nodes in Figure 2 include all the at nodes as well as the $\mathsf{joined}(p3, p1)$ node.

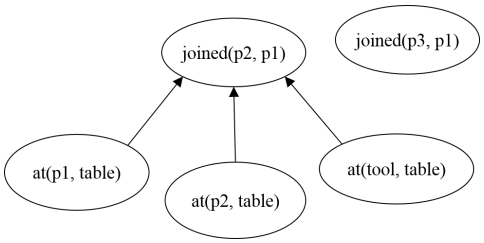

Figure 2: Sample Goal Network for an Automated Manufacturing domain

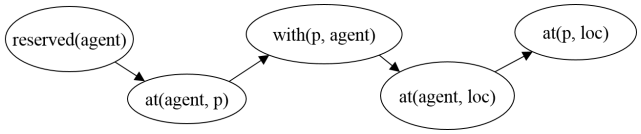

Figure 3: Subgoal network of $\mathsf{deliver\text{-}obj}(p, loc, agent)$, a HGN method to deliver the part p to loc using $agent$.

We define the following operations over any goal network $gn = (T, \prec)$:

1. **Goal Release**: Let $t \in T$ be an unconstrained node. Then the removal of t from gn, denoted by $gn - t$, results in the goal network $gn' = (T', \prec')$ where $T' = T \setminus \{t\}$ and $\prec'$ is the restriction of $\prec$ to T'.
2. **Method Application**: Let $t \in T$ be an unconstrained node. Also, let m be a method applied to t with network(m) $= (T_m, \prec_m)$. Finally, recall that network(m) always contains a 'last' node that contains goal(m); let t_g be this node. Then the application of m to gn via t, denoted by $gn \circ_t m$, results in the goal network $gn' = (T', \prec')$ where $T' = T \cup T_m$ and $\prec' = \prec \cup \prec_m \cup \{(t_g, t)\}$. Informally, this operation adds the elements of network(m) to gn, preserving the order specified by subgoals(m) and setting goal(m) as a predecessor of t.

2.5 HGN Domains, Problems and Solutions

A *HGN domain* is a pair $D = (D_{classical}, M)$ where $D_{classical}$ is a classical planning domain and M is a set of HGN methods.

A *HGN planning problem* is a triple $P = (D, s_0, gn_0)$, where D is a HGN domain, s_0 is the initial state, and $gn_0 = (T, \prec)$ is the initial goal network.

Definition 2 (Solutions to HGN Planning Problems)**.** *The set of* solutions *for P is defined as follows:*

Base Case. *If T is empty, the empty plan is a solution for P.*

In the following cases, let $t \in T$ be an unconstrained node.

Unconstrained Goal Satisfaction. *If $s_0 \models g_t$, then any solution for $P' = (D, s_0, gn_0 - t)$ is also a solution for P.*

Action Application. *If action a is applicable in s_0 and a is relevant to g_t, and π is a solution for $P' = (D, \gamma(s_0, a), gn_0)$, then $a \circ \pi$ is a solution for P.*

Method Decomposition. *If m is a method applicable in s and relevant to g_t, then any solution to $P' = (D, s_0, gn_0 \circ_t m)$ is also a solution to P.*

Note that HGN planning allows an action to be applied only if it is *relevant* to an unconstrained node in gn; this prevents action chaining as done in classical planning and allows for tigher control of solutions as in HTN planning. In fact, prior work (Shivashankar et al. 2012) showed that HGN planning is as expressive as HTN planning when both are restricted to totally-ordered methods, i.e. the subtask/subgoal networks are totally ordered.

Let us denote $\mathcal{S}(P)$ as the set of solutions to a HGN planning problem P as allowed by Definition 2. Then we can define what it means for a solution π to be *hierarchically optimal* with respect to P as follows:

Definition 3 (Hierarchically Optimal Solutions)**.** *A solution $\pi^{h,*}$ is hierarchically optimal with respect to P if $\pi^{h,*} = argmin_{\pi \in \mathcal{S}(P)} \mathsf{cost}(\pi)$.*

3 h_{HL}: An Admissible Heuristic for HGN Planning

At a high level, we will proceed to construct h_{HL} as follows:

1. We define a relaxation of HGN planning that ignores the provided methods and allows unrestricted action chaining as in classical planning, which expands the set of allowed solutions,
2. We will extend landmark generation algorithms for classical planning problems to compute sound landmark graphs for the relaxed HGN planning problems, which in turn are sound with respect to the original HGN planning problems as well, and finally
3. We will use admissible classical planning heuristics like h_L on these landmark graphs to compute admissible cost estimates for HGN planning problems.

3.1 Relaxed HGN Planning

Definition 4 (Relaxed HGN Planning)**.** *A relaxed HGN planning problem is a triple $P = (D_{classical}, s_0, gn_0)$ where D is a classical planning domain, s_0 is the initial state, and gn_0 is the initial goal network. Any sequence of actions π that is executable in state s_0 and achieves the goals in gn_0 in an order consistent with the constraints in gn_0 is a valid solution to P.*

Relaxed HGN planning can thus be viewed as an extension of classical planning to solve for goal networks, where there are no HGN methods and the objective is to generate sequences of actions that satisfy the goals in gn_0 in an order consistent with gn_0. In fact, it is easy to show that relaxed HGN planning, in contrast to HGN planning, is no more expressive than classical planning, and relaxed HGN planning problems can be compiled into classical planning problems quite easily.

Next, we will show how to leverage landmark generation algorithms for classical planning to generate landmark graphs for relaxed HGN planning.

3.2 Generating Landmarks for Relaxed HGN Planning

This section describes a landmark discovery technique that can use any landmark discovery technique for classical planning (referred to as LMGEN_C here) such as (Richter and Westphal 2010) to compute landmarks for relaxed HGN planning problems. The main difference here is that while classical planning problems are $(state, goal)$ pairs, relaxed HGN planning problems are $(state, goal\text{-}network)$ pairs; every goal in the goal network can be thought of as a landmark. Therefore, there is now a partially ordered set of goals to compute landmarks from, as opposed to a single goal in classical planning.

Algorithm 1 Procedure for computing landmarks for relaxed HGN planning problems.

```
1: function computeHGNLandmarks(s, gn)
2:     queueSeeds ← gn
3:     queue ← ∅
4:     while queueSeeds is not empty do
5:         choose a g w/o successors from queueSeeds, and remove it along with all associated orderings
6:         addLM(g), add g to queue
7:         add any orderings g shares with other goals from gn already added to LG
8:         while queue is not empty do
9:             pop landmark ψ from queue and use LMGEN_C to generate the new set of landmarks Φ
10:            for φ ∈ Φ do ADDLM(φ, φ →_gn ψ)
11:    return LG
12:
13: function addLM(φ)
14:    if φ is a fact and ∃φ' ∈ LG : φ' ≠ φ ∧ φ ⊨ φ' then
15:        remove φ' from LG and all orderings it is part of
16:    if ∃φ' ∈ LG : φ' ⊨ φ then return φ'
17:    if φ ∉ LG then add φ to queue and return φ
18:
19: function addLMandOrdering(φ, φ →_x ψ)
20:    η ← addLM(φ)
21:    add ordering η →_x ψ to LG
```

We therefore need to generalize classical planning landmark generation techniques to work for relaxed HGN planning problems. The computeHGNLandmarks algorithm (Algorithm 1) describes one such generalization. At a high level, computeHGNLandmarks proceeds by computing landmark graphs for each goal g in gn (which in fact is a classical planning problem) and merging them all together to create the final landmark graph LG.

computeHGNLandmarks takes as input a relaxed HGN planning problem (s, gn) and generates LG, a graph of landmarks. First, queueSeeds is initialized with a copy of gn (Line 2). This is because unlike in classical planning where

we have a single goal to generate landmarks from, in HGN planning we have a partially ordered set of goals to seed the landmark generation; `queueSeeds` stores these seeds. We also initialize `queue`, the openlist of landmarks, to $\emptyset$.

While there is a goal g from gn that we have not yet computed landmarks for (Line 4), we do the following: we remove it from `queueSeeds` along with all induced orderings and add it to `queue` (Lines 5–6). We also add g to LG using addLM; we also add any ordering constraints it might have with other elements of gn that have already been added to LG. This `queue` is then used as a starting point by LMGEN_C to begin landmark generation. We iteratively use LMGEN_C to pop landmarks off the `queue` and generate new landmarks by backchaining until we can no longer generate any more landmarks (Lines 8–10). Each new landmark is added to LG by the addLMandOrdering procedure. Once all goals in gn have been handled, the landmark generation process is completed and the algorithm returns LG.

The addLM procedure takes as input a computed landmark ϕ, adds it to LG and returns a landmark η. There are three cases to consider:

- ϕ subsumes another landmark ϕ' in LG, implying we can remove ϕ' and replace it with ϕ (since ϕ is a stronger version of ϕ'), and return ϕ (Lines 14–15)
- ϕ is subsumed by another landmark ϕ' in LG, implying we can ignore ϕ (Lines 16). In this case, we don't add any new landmark to LG and simply return ϕ'
- ϕ is a new landmark, in which case we can simply add it to LG and return ϕ (Lines 17)

The addLMandOrdering procedure takes as input a landmark ϕ and an ordering constraint $\phi \rightarrow_x \psi$ and adds them to LG. More precisely, it adds ϕ to LG using addLM, which returns the added landmark η. It then adds the ordering constraint between η and ψ in LG.

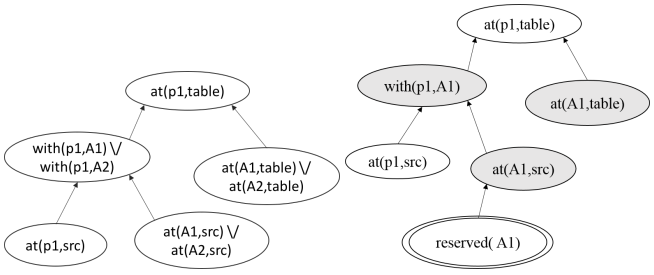

Figure 4: (a) LM graph on goal network containing a single goal $\mathsf{at}(p_1, table)$. (b) LM graph after decomposing $\mathsf{at}(p_1, table)$ with $\mathsf{deliver\text{-}obj}(p_1, table, A1)$. The double-circled landmarks represent new landmarks inferred after the method decomposition, while the landmarks colored gray are new landmarks that subsumed an existing one in (a).

LM graph computation example. Figure 4 illustrates the working of computeHGNLandmarks. Let us assume the goal network gn contains only one goal $g = \mathsf{at}(p_1, table)$. Figure 4a illustrates the output of computeHGNLandmarks on g. This is identical to what LMGEN_C would generate, since gn contains only one goal, making the relaxed HGN problem equivalent to a classical planning problem.

Now, let us assume that we decompose gn using the $m = \mathsf{deliver\text{-}obj}(p_1, table, A1)$, and get the new goal network gn', which essentially looks like an instantiated version of the network in Figure 3. Now if we run computeHGNLandmarks on gn', we end up generating the landmark graph in Figure 4b, which is a more focused version of the first landmark graph. This is because the goals in gn' are landmarks that must be accomplished, which constrains the set of valid solutions that can be generated. For instance, since we've committed to agent $A1$, every solution we can generate from gn' will involve the use of $A1$. We can, as a result, generate more focused landmarks than we otherwise could have from just the top-level goal g. This includes fact landmarks that replace disjunctive landmarks (the ones in gray in Fig. 4b) as well as completely new landmarks that arise as a result of the method; e.g. $\mathsf{reserved}(A1)$ is not a valid landmark for gn, but is one for gn'.

An important point to note at this point is that the subgoals in gn' are not *true* landmarks for g; they are landmarks once we commit to applying method m. However, this actually ends up being useful to us, since it allows us to generate different landmark graphs for different methods; for instance, if we had committed to $A2$, we would have obtained a different set of landmarks specific to $A2$. Now, landmark-based heuristics when applied to these two graphs would get us different heuristic estimates, thus allowing to differentiate between these two methods by using the specific subgoals each method introduces.

It is easy to show that computeHGNLandmarks generates sound landmark graphs for relaxed HGN planning problems:

Claim 5. *Given a relaxed HGN planning problem* $P = (D_{classical}, s_0, gn_0)$, $LG = \mathsf{computeHGNLandmarks}(s_0, gn_0)$ *is a sound landmark graph for* P.

Let $P = ((D_{classical}, M), s_0, gn_0)$ be a HGN planning problem, and let $P' = (D_{classical}, s_0, gn_0)$ be the corresponding relaxed version. Then by definition, any solution to P is a solution to P'. Therefore, it is easy to see that a landmark of P' is also a sound landmark of P. More generally, a landmark graph generated for P' is going to be sound with respect to P as well:

Claim 6. *Given a HGN planning problem* P, *then* $LG = \mathsf{computeHGNLandmarks}(s_0, gn_0)$ *is a sound landmark graph for* P.

3.3 Computing h_{HL}

The main insight behind h_{HL} is the following: *since the* computeHGNLandmarks *algorithm generates sound landmarks and orderings for relaxed (and therefore regular) HGN planning problems, we can use any admissible landmark-based heuristic from classical planning to derive an admissible cost estimate for HGN planning problems.*

In particular, h_{HL} uses h_L as follows: given an HGN search node (s, gn), the landmark graph is given by

$LG_{HGN} = \mathsf{computeHGNLandmarks}(s, gn)$. Then

$$h_{HL}(s, gn, \pi) = h_L(LG_{HGN}, s, \pi) \quad (2)$$

where π is the plan generated to get to (s, gn).

3.4 Admissibility of h_{HL}

Claim 6 shows that given a HGN problem $P = (D, s_0, gn_0)$, $LG = \mathsf{computeHGNLandmarks}(s_0, gn_0)$ is a sound landmark graph with respect to P. Furthermore, Lemma 1 shows that $h_L(LG, s_0, \langle\rangle)$ provides an admissible cost estimate of the optimal plan starting from s_0 that achieves all the landmarks in LG. Since every solution to P has to achieve all the landmarks in LG in a consistent order, $h_L(LG, s_0, \langle\rangle)$ provides an admissible estimate of the optimal cost to P as well. However, from Eq. 2, $h_L(LG, s_0, \langle\rangle) = h_{HL}(s_0, gn_0, \langle\rangle)$. Therefore, we have the following theorem:

Theorem 7 (Admissibility of h_{HL})**.** *Given a HGN planning domain D, a search node (s, gn, π) and its cost-optimal solution $\pi^{*,HGN}_{s,gn}$, $h_{HL}(s, gn, \pi) \leq \pi^{*,HGN}_{s,gn}$.*

4 The HOpGDP Algorithm

Algorithm 2 describes HOpGDP. It takes as input a HGN domain $D = (D', M)$, the initial state s_0 and the initial goal network gn_0. It returns a plan if it finds one, or failure if the problem is unsolvable.

Initialization. It starts off by initializing open (Line 2), which is a priority queue that sorts the HGN search nodes yet to be expanded by their f-value, where $f((s, gn, \pi)) = \mathsf{cost}(\pi) + h_{HL}(s, gn)$. open initially contains the initial search node $(s_0, gn_0, \langle\rangle)$. It also initializes searchSpace (Line 3), the set of all nodes seen during the search process. This data structure keeps track of the best known path currently known for each state,goal-network pair, and is thus helpful to detect when we find a cheaper path to a previously seen HGN search node.

Search. HOpGDP now proceeds to do an A* search in the space of HGN search nodes starting from the initial node. While open is not empty, it does the following (Lines 4–14): it removes the HGN search node $N = (s, gn, \pi)$ with the best f-value from open (Line 5) and first checks if gn is empty (Line 6). If this is true, this means that all the goals in gn_0 have been solved, and π is the optimal solution to the HGN planning problem.

If gn is not empty, then the algorithm proceeds by using the getSuccessors subroutine to compute N's successor nodes (Line 7). For each successor node (s', gn', π'), it proceeds to do the following: it checks to see if another path η to (s', gn') exists in searchSpace (Line 9). If this is the case and if η is costlier than π' (Line 10), it updates searchSpace with the new path; and reopens the search node (Line 14); if η is cheaper than the new plan π', it simply skips this successor (Line 12).

If (s', gn') has not been seen before, it adds $N' = (s', gn', \pi')$ to searchSpace to track the currently best-known plan π' to (s', gn') (Line 13). It also evaluates the f-value of N' and adds it to open (Line 14).

If there are no more nodes left in open, this implies that it has exhausted the search space without finding a solution, and therefore returns failure (Line 15).

Computing Successors. The procedure getSuccessors computes the successors of a given HGN search node (s, gn, π) in accordance with Definition 2. First, we check to see if there are any unconstrained goals g in gn that are satisfied in the current state s. We then proceed to create new HGN search nodes by removing all such goals from gn (Line 19–20). Next, we compute all actions applicable in s and relevant to an unconstrained goal in gn (Line 21) and create new search nodes by progressing s using these actions (Line 22–23). We compute all pairs (m, g) such that m is a HGN method applicable in s and relevant to an unconstrained goal g in gn (Line 24) and create new search nodes by decomposing g in gn using m (Line 25–26). Finally, we return the set of generated successor nodes (Line 27).

Algorithm 2 Pseudocode of HOpGDP. It takes as arguments the domain description $D = (D_{classical}, M)$, the initial state s_0, and the initial goal network gn_0. It either returns a plan if it finds one, or failure if it doesn't.

```
1:  function HOpGDP(D, s_0, gn_0)
2:      open ← (s_0, gn_0, ⟨⟩)
3:      searchSpace ← (s_0, gn_0, ⟨⟩)
4:      while open is not empty do
5:          rem. (s, gn, π) with lowest f-value from open
6:          if gn is empty then return π
7:          successors ← getSuccessors(D, s, gn, π)
8:          for (s', gn', π') ∈ successors do
9:              if ∃(s', gn', η) ∈ searchSpace then
10:                 if cost(π') < cost(η) then
11:                     replace (s', gn', η) with (s', gn', π')
                        in searchSpace
12:                 else continue
13:             else add (s', gn', π') to searchSpace
14:             eval. f-value of (s', gn', π') and add to open
15:     return failure
16:
17: function getSuccessors(D, s, gn, π)
18:     successors ← ∅
19:     for unconstrained g ∈ gn satisfied in s do
20:         add the node (s, gn − {g}, π) to successors
21:     A ← actions in D applicable in s and relevant to an
        unconstrained goal in gn
22:     for a ∈ A do
23:         add the node (γ(s, a), gn, π ∘ a) to successors
24:     M ← {(m, g) s.t. m ∈ M is applicable in s and
        relevant to an unconstrained goal g in gn}
25:     for (m, g) ∈ M do
26:         add the node (s, gn ∘_g m, π) to successors
27:     return successors
```

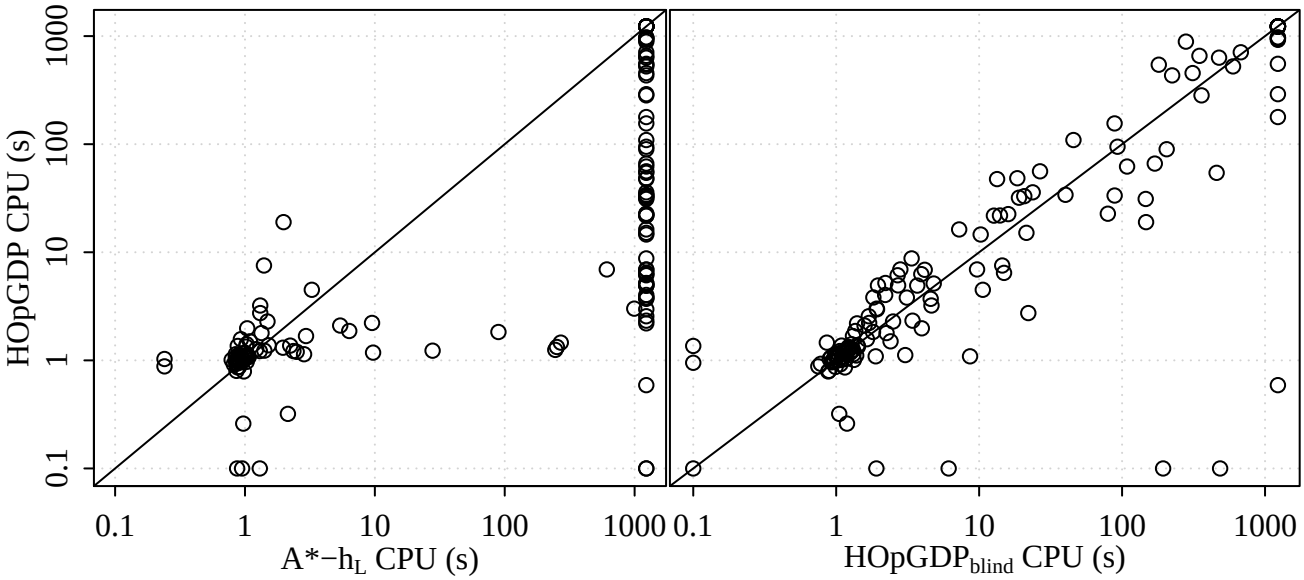

Figure 5: Log-scale scatter plot comparing HOpGDP planning time vs. A*-h_L (left) and $\mathsf{HOpGDP}_{\mathsf{blind}}$ (right) planning time on Blocksworld and Logistics problems.

	Count	A*-h_L	$\mathsf{HOpGDP}_{\mathsf{blind}}$	HOpGDP
bw	114	60	102	110
log	28	11	22	22
Total	142	71	124	132

	A*-h_L		$\mathsf{HOpGDP}_{\mathsf{blind}}$		HOpGDP	
	$\bar{s}$	σ	$\bar{s}$	σ	$\bar{s}$	σ
bw	42.7	157.2	8.0	30.9	1.8	2.6
log	3.1	2.8	1.2	0.2	1.4	0.5

	A*-h_L		$\mathsf{HOpGDP}_{\mathsf{blind}}$		HOpGDP	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
bw	1097840	3805838	22047	96523	1194	3806
log	106281	133890	769	775	569	608

Table 1: The coverage (top) and the mean CPU seconds (middle) and mean number of node expansions (bottom) for A*-$h_L$using h_L, $\mathsf{HOpGDP}_{\mathsf{blind}}$, and HOpGDPusing h_{HL}. For runtime and nodes, we include the sample mean ($\bar{x}$) and standard deviation (σ). Other than coverage, all statistics are over the subset of problems solved by all three variants.

5 Preliminary Experiments

Our evaluation of HOpGDP focuses on two questions: Is the heuristic informative in guiding search, and is its guidance sufficient to overcome its computation time. We chose the well-known Blocksworld and Logistics domains for our preliminary study, using the HGN methods described in the GoDeL evaluation. For logistics, we limited truck capacity to one package to ensure that the optimal solutions were the same between the HGN and non-HGN planners.

We implemented HOpGDP in the GoDeL framework (Shivashankar et al. 2013), which is derived from the Fast-Downward (Helmert 2009) code base. We chose three variants of HOpGDP to compare: A*-h_L, which ignores all methods and corresponds to A* with the classical h_L heuristic; $\mathsf{HOpGDP}_{\mathsf{blind}}$, which corresponds to Algorithm 2 with $h = 0$, so that the f-value is always g, the distance from the start; and the full HOpGDP algorithm with the h_{HL} heuristic. Both A*-h_L and HOpGDP break ties on lower h values. We ran all problems on a Xeon E5-2639 with a per problem limit of 4 GB of RAM and 20 minutes of planning time.

Table 1 (top) shows the coverage for the three algorithms. HOpGDP and $\mathsf{HOpGDP}_{\mathsf{blind}}$ have nearly twice the coverage of A*-h_L, which confirms the power of pruning in search, even when comparing blind and heuristic search. The difference between HOpGDP and $\mathsf{HOpGDP}_{\mathsf{blind}}$ is more subtle, with HOpGDP covering all of $\mathsf{HOpGDP}_{\mathsf{blind}}$'s problems plus eight more.

Figure 5 gives a scatter plot of run times of HOpGDP vs. A*-h_L and $\mathsf{HOpGDP}_{\mathsf{blind}}$. Table 1 (middle) summarizes the runtimes for the set of problems solved by all three variants. The results roughly match the coverage trends, but we make

no statistical claims.

Note that the runtime for HOpGDP on logistics is higher than $\mathsf{HOpGDP}_{\mathsf{blind}}$. The bottom section of Table 1 gives some context to the results. A*-h_L has an order of magnitude faster node-expansion rate than $\mathsf{HOpGDP}_{\mathsf{blind}}$, which in turn has a four times higher expansion rate than HOpGDP. Part of the reason for this is architecture. HGN methods, as with HTN methods, tend to have high numbers of free variables. Grounding these methods bloated the domain beyond use in sample runs, and so methods are unified against a state by calling out to SHOP2's unifier over local sockets. Part of the difference between A*-h_L and HOpGDP is inherent, though, since h_L calculates one landmark graph per problem and h_{HL} must calculate a new landmark graph for every goal network it encounters.

While these trends are generally positive, the results are hardly conclusive. We plan on expanding the set of problems and more closely analyze the results in future work.

6 Related Work

HTN planners solve planning problems in one of two ways, either (1) by forward state-space search, such as in the SHOP (Nau et al. 1999) and SHOP2 (Nau et al. 2003) HTN planners, or (2) by partial-order causal-link planning (POCL) techniques, such as in UMCP (Erol, Hendler, and Nau 1994) and in the hybrid planning literature (Elkawkagy et al. 2012; Bercher, Keen, and Biundo 2014).

Due to very little work in the way of search heuristics for forward-search HTN planning, planners often end up providing other domain-specific mechanisms for users to encode search strategies. For example, SHOP2 allows the domain-specific knowledge, known as *HTN methods*, to be specified in a 'good' order according to the user, and tries them out in the same order. SHOP2 also provides support for external function calls (Nau et al. 2003) that can call arbitrary code to do the heavy lifting in the problem, thus minimizing the choices that need to be made during search. For example, in the 2002 Planning Competition for hand-tailored planners, the authors of SHOP2 implemented a shortest-path algorithm in the DriverLog domain that SHOP2 could call externally to generate optimal paths.[2]

Waisbrot et al (Waisbrot, Kuter, and Konik 2008) developed $H2O$, a HTN planner that augments SHOP2 with classical planning heuristics to make local decisions on which method to apply next by estimating how close the method's goal is to the current state. $H2O$, however, retains the depth-first search structure of SHOP2, making it to difficult to generate high-quality plans.

Marthi et al (Marthi, Russell, and Wolfe 2007; 2008) propose an HTN-like formalism called *angelic hierarchical planning* which allows users to annotate abstract tasks with additional domain-specific information in the form of lower and upper bounds on the costs of the possible plans they decompose to. They then use this information to compute hierarchically-optimal plans. In contrast, we require only costs of the primitive actions and use domain-independent search heuristics to compute hierarchically-optimal plans.

[2]personal communication with Ugur Kuter.

There has been recent work on developing search heuristics for POCL HTN planners (Elkawkagy et al. 2012; Bercher, Keen, and Biundo 2014). However, these heuristics typically provide estimates on how many more plan refinement steps need to be taken from a search node in order to get to a solution, as opposed to plan quality estimates, which is what we are focused on in this paper.

Hierarchical Goal Network (HGN) Planning combines the hierarchical structure of HTN planning with the goal-based nature of classical planning. It therefore allows for easier infusion of techniques from classical planning into hierarchical planning, such as adapting the FF heuristic to do *method ordering* in the GDP planner (Shivashankar et al. 2012), and using landmark-based techniques to plan with partial amounts of domain knowledge in GoDeL (Shivashankar et al. 2013). Both planners, however, use depth-first search and inadmissible heuristics, so they cannot provide any guarantees of plan quality.

Another domain-configurable planning formalism is *Planning with Control Rules* (Bacchus and Kabanza 2000), where domain-specific knowledge is encoded in the form of *linear-temporal logic* (LTL) formulas. TLPlan, one of the earliest planners developed under this formalism, used control rules written in LTL to prune away trajectories deemed suboptimal by the user. There have also been attempts to develop heuristic search planners that can plan with LTL_f, a simplified version of LTL that works with finite traces. This has been used to incorporate search heuristics to solve for temporally extended goals written in LTL_f (Baier and McIlraith 2006) as well as to express landmark-based heuristics that guide classical planners (Simon and Roger 2015).

7 Conclusion

Despite the popularity of hierarchical planning techniques both in theory and practice, very little effort has been devoted to developing domain-independent search heuristics that can provide useful search guidance towards high-quality solutions. As a result, end-users need to encode domain-specific heuristics into the domain models, which can make the domain-modeling process tedious and error-prone.

To address this issue, this paper leverages recent work on HGN planning, which allows tighter integration of hierarchical planning and classical planning, to develop (1) h_{HL}, an admissible HGN planning heuristic, and (2) HOpGDP, an A* search algorithm guided by h_{HL} to compute *hierarchically-optimal* plans.

There are several avenues for future work, such as:

- **Theoretical Analysis**: We show that h_{HL} returns admissible cost estimates for HGN planning problems. However, we believe that it has other interesting theoretical properties as well. In particular, we conjecture that it dominates h_L, since it can, in general, compute more focused landmarks, which can translate to more informed heuristic estimates, a property of optimal cost partitioning with h_L. On a related note, we believe that h_{HL} has the nice property that despite some of the steps being zero-cost (i.e. method applications, which don't change the state), it can help us avoid f-value plateaus since the method ap-

plication can result in a more informative heuristic value. We plan on doing a more detailed theoretical analysis to verify these conjectures.

- **Extension to Anytime Planning**: While we believe it is theoretically interesting that h_{HL} can help us find optimal solutions, it would be of practical interest to design *anytime* HGN planning algorithms.

Acknowledgment This work is sponsored in part by OSD ASD (R&E). The information in this paper does not necessarily reflect the position or policy of the sponsors, and no official endorsement should be inferred. Ron Alford performed part of this work under an ASEE postdoctoral fellowship at NRL.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artif. Intell.* 116:123–191.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI Conference on Artificial Intelligence*.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *Proc. of the Seventh Annual Symposium on Combinatorial Search (SoCS)*, 35–43. AAAI Press.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *AAAI Conference on Artificial Intelligence*, 1763–1769.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. 249–254. ICAPS 2009 influential paper honorable mention.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF planning system. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1728–1733.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Angelic semantics for high-level actions. In *International Conference on Automated Planning and Scheduling*.

Marthi, B.; Russell, S.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *International Conference on Automated Planning and Scheduling*, 222–231.

Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In Dean, T., ed., *International Joint Conference on Artificial Intelligence*, 968–973.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)* 39:127–177.

Shivashankar, V.; Kuter, U.; Nau, D.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. of the 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, 981–988. Int. Foundation for Autonomous Agents and Multiagent Systems.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. 2013. The GoDeL planning system: a more perfect union of domain-independent and hierarchical planning. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2380–2386. AAAI Press.

Simon, S., and Roger, G. 2015. Finding and exploiting ltl trajectory constraints in heuristic search. In *Symposium on Combinatorial Search*.

Waisbrot, N.; Kuter, U.; and Konik, T. 2008. Combining heuristic search with hierarchical task-network planning: A preliminary report. In *International Conference of the Florida Artificial Intelligence Research Society*.