Hierarchical Goal Network Planning: Initial Results

Vikas Shivashankar
Dept. of Computer Science
University of Maryland
College Park, MD 20742 USA
svikas@cs.umd.edu

Ugur Kuter
Smart Information Flow Technologies
211 North 1st Street
Minneapolis, MN 55401 USA
ukuter@sift.net

Dana S. Nau
Dept. of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742 USA
nau@cs.umd.edu

Technical Report CS-TR-4983 and UMIACS-TR-2011-09

May 31, 2011

Abstract

In applications of HTN planning, repeated problems have arisen from the lack of correspondence between HTN tasks and classical-planning goals. We describe these problems and provide a new *Hierarchical Goal Network (HGN)* planning formalism that overcomes them. HGN tasks have syntax and semantics analogous to classical planning problems, and this has several benefits: HGN methods can be significantly simpler to write than HTN methods, there is a clear criterion for whether the HGN methods are correct, and classical-planning heuristic functions can be adapted for use in HGN planning.

We define the HGN formalism, illustrate how to prove correctness of HGN methods, provide a planning algorithm called GNP (Goal Network Planner), and present experimental results showing that GNP's performance compares favorably to that of SHOP2. We provide a planning-graph heuristic for optional use in GNP, and give experimental results showing the kinds of situations in which it helps or hurts GNP's performance.

1 Introduction

HTN planning, in which the planner decomposes *tasks* recursively into smaller and smaller *subtasks* until *primitive* tasks are reached that correspond directly to actions, has been widely used in practical applications of AI planning; e.g., see [16, 18, 13].

Most research on HTN planning has been based on a formalism (Erol et al. [5]) in which HTN tasks did not correspond to classical goals, but instead were syntactic terms denoting complex activities. The semantics of a task t depended on what methods were applicable to t, and what those methods did. The definition's authors (one of whom is the third co-author of this paper) thought this was an advantage because it gave HTN planning strictly more expressivity than classical planning. But in practice, it has caused several problems:

- The incompatibility between tasks and goals has made it difficult to ascertain whether a set of HTN methods provides a sound way to solve a planning problem. For example, when SHOP [14] was entered in the 2000 International Planning Competition, it was disqualified because the SHOP team's HTN version of the Freecell planning domain was unsound: it contained an error that sometimes caused SHOP to return plans that were not solutions to Freecell problems.

- Consider a classical planning problem in which the goal is $G = \{g_1, g_2\}$. Suppose we have chosen to represent g_1 and g_2 by HTN tasks t_1 and t_2, and to write methods m_1 and m_2 for t_1 and t_2, respectively. As an HTN representation of G, it would be tempting to use the task sequence $\langle t_1, t_2 \rangle$; but this is incorrect because it doesn't allow for deleted-condition interactions. If $\neg g_1$ is a side-effect of the plan produced by decomposing m_2, then a plan produced from $\langle m_1, m_2 \rangle$ will accomplish the task sequence $\langle t_1, t_2 \rangle$, but this plan will not achieve G.

- Conversely, if an HTN method m always produces a state that satisfies $G = \{g_1, \ldots, g_n\}$, then we might want an HTN planner to use m for any goal $G' \subseteq G$. To specify this, we would need an HTN task for each $G' \subseteq G$. This can produce an exponential blowup in the number of tasks and methods.

This paper describes *Hierarchical Goal Network (HGN)* planning, a new planning formalism designed to overcome the above problems. It is similar to HTN planning, but has a different syntax and semantics for tasks and methods.

In HGN planning, a task consists of *initial conditions* and *goal conditions* that have the same semantics as in classical planning. An HGN method m has preconditions and subgoals; and m's applicability and relevance are defined analogously to how an action's applicability and relevance are defined in classical planning. Our definitions guarantee *soundness* of HGN methods, and make it possible to prove whether a set of methods M is *complete* for a set of goals G.

In this paper we make the following contributions:

- **Formalism:** We define the HGN formalism, prove that for any classical planning domain there is a provably correct set of HGN methods, and show how to write a correctness proof for a set of HGN methods.

- **Algorithm:** We provide *GNP (Goal Network Planner)*, a provably sound/complete HGN planning algorithm. GNP is somewhat like SHOP2 [15], and our experiments show that its performance compares favorably to SHOP2's.

- **Heuristic functions:** It is straightforward to adapt classical-planning heuristic functions for use in HGN planning. We provide a planning-graph heuristic for optional use in GNP. We provide experimental results illustrating the kinds of situations in which such a heuristic is useful, and the kinds of situations in which it is not.

- **Ease of domain authoring:** For the planning domains in our experiments, we found it significantly easier to write HGN methods than HTN methods. We explain why.

2 Formalism

Classical planning. Following (Ghallab et al. [6, Chapter 2]), we define a classical planning domain D 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 triple $o = (\text{head}(o), \text{pre}(o), \text{eff}(o))$, where $\text{pre}(o)$ and $\text{eff}(o)$ are sets of literals called o's *preconditions* and *effects*, and $\text{head}(o)$ includes o's *name* and *argument list* (a list of the variables in $\text{pre}(o)$ and $\text{eff}(o)$).

An action a is executable in a state s if $s \models \text{pre}(a)$, in which case the resulting state is $\gamma(a) = (s - \text{eff}^-(a)) \cup \text{eff}^+(a)$, where $\text{eff}^+(a)$ and $\text{eff}^-(a)$ are the atoms and negated atoms, respectively, in $\text{eff}(a)$. 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 we let $\gamma(s, \pi)$ be the state produced by executing the entire plan.

A *classical planning problem* is a triple $P = (D, s_0, g)$, where D is a classical planning domain, s_0 is the initial state, and g (the *goal formula*) is a set of ground literals. A plan π is a solution for P if π is executable in s_0 and $\gamma(s_0, \pi) \models g$.

HGN planning. An *HGN method* m has a head $\text{head}(m)$ and preconditions $\text{pre}(m)$ like those of a planning operator, and a sequence of subgoals $\text{sub}(m) = \langle g_1, \ldots, g_k \rangle$, where each g_i is a goal formula (a set of literals). We define the *postcondition* of m to be $\text{post}(m) = g_k$ if $\text{sub}(m)$ is nonempty; otherwise $\text{post}(m) = \text{pre}(m)$.

An action a (or method instance m) is *relevant* for a goal formula g if $\text{eff}(a)$ (or $\text{post}(m)$, respectively) entails at least one literal in g and does not entail the negation of any literal in g.

Some notation: if $\pi_1, \ldots, \pi_n$ are plans or actions, then $\pi_1 \circ \ldots \circ \pi_n$ denotes the plan formed by concatenating them.

An *HGN planning domain* is a pair $D = (D', M)$, where D' is a classical planning domain and M is a set of methods. An *HGN planning problem* $P = (D, s_0, g)$ is like a classical planning problem except that D is an HGN planning domain. The set of solutions for P is defined recursively:

Case 1. If $s_0 \models g$, then the empty plan is a solution for P.

Case 2. Let a be any action that is relevant for g and executable in s_0. Let π be any solution to the HGN planning problem $(D, \gamma(s_0, a), g)$. Then $a \circ \pi$ is a solution to P.

Case 3. Let m be a method instance that is applicable to s_0 and relevant for g and has subgoals $g_1, \ldots, g_k$. Let π_1 be any solution for (D, s_0, g_1); let π_i be any solution for $(D, \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_{i-1})), g_i)$, $i = 2, \ldots, k$; and let π be any solution for $(D, \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_k)), g)$. Then $\pi_1 \circ \pi_2 \circ \ldots \circ \pi_k \circ \pi$ is a solution to P.

In the above definition, the relevance requirements in Cases 2 and 3 prevent classical-style action chaining unless each action is relevant for either the ultimate goal g or a subgoal of one of the methods. This requirement is analogous to (but less restrictive than) the HTN planning requirement that actions cannot appear in a plan unless they are mentioned explicitly in one of the methods. As in HTN planning, it gives an HGN planning problem a smaller search space than the corresponding classical planning problem.

Theorem 1 (Soundness of HGN planning). *Let $D = (D', M)$ be an HGN planning domain. For every initial state s_0 and goal g, the set of solutions to the HGN planning problem $P = (D, s_0, g)$ is a subset of the set of solutions to the classical planning problem $P' = (D', s_0, g)$.*

Proof. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be any solution for P. From the definition of a solution, it follows that in the HGN domain D, π is executable in s_0 and $\gamma(s_0, \pi) \models g$. But $D = (D', M)$, so any action that is executable in D is also executable in the classical domain D' and produces the same effects. Thus it follows by induction that in D', π is executable in s_0 and $\gamma(s_0, \pi) \models g$. □

Theorem 2 (Completeness of HGN planning). *For every classical planning domain D, there is a set of HGN methods M such that every planning problem P has the same solutions in both D and (D, M).*

Proof. Let X be the set of all *simple* paths in D. For each path x in X, suppose M contains methods that will specify goals for each state on x as subgoals. Thus, each subgoal will be achieved by a single action such that when the sequence of actions applied from the start of x, and the result will be the end state. Then the theorem follows. □

The above theorem gives a way to construct a complete set of methods for any planning domain, but in practice those methods would be quite unsatisfactory. There are exponentially many of them, they are fully ground, and they include methods to solve many planning problems that one would never encounter in practice. Later, in the *Domain Authoring* section, we illustrate how to write a practical set of methods and prove their correctness.

3 Planning with HGNs

GNP, our HGN planning algorithm, is shown in Algorithm 1. It works as follows (where G is a stack of goal formulas that need to be achieved):

In Line 2, if G is empty then the goal has been achieved, so GNP returns π. Otherwise, GNP selects the first goal g in G (Line 3). If g is already satisfied, GNP removes g from G and calls itself recursively on the remaining goal formulae.

In Lines 7-8, if no actions or methods are applicable to s and relevant for g, then GNP returns failure. Otherwise, GNP nondeterministically chooses one them and calls it u.

If u is an action, then GNP computes the next state $\gamma(s, u)$ and appends u to π. Otherwise u is a method, so GNP inserts u's subgoals at the front of G. Then GNP calls itself recursively on G.

The following theorems show that GNP is sound and complete:

Theorem 3 (Soundness of GNP). *Let $P = (D, s_0, g)$ be an HGN planning problem. If a nondeterministic trace of* GNP$(D, s_0, \langle g \rangle, \langle\rangle)$ *returns a plan π, then π is a solution for P.*

Algorithm 1: A high-level description of GNP. Initially, D is an HGN planning domain, s is the initial state, g is the goal formula, $G = \langle g \rangle$, and π is $\langle\rangle$, the empty plan.

```
1  Procedure GNP($D, s, G, \pi$)
2  if $G$ is empty then return $\pi$
3  $g \leftarrow$ the first goal formula in $G$
4  if $s \models g$ then
5      remove $g$ from $G$
6      return GNP($D, s, G, \pi$)
7  $U \leftarrow \{$actions and method instances that are relevant for $g$ and applicable to $s\}$
8  if $U = \emptyset$ then return failure
9  nondeterministically choose $u \in U$
10 if $u$ is an action then
11     append $u$ to $\pi$
12     set $s \leftarrow \gamma(s, u)$
13 else insert sub($u$) at the front of $G$
14 return GNP($D, s, G, \pi$)
```

Sketch of proof. The proof is by induction on n, the length of π. When $n = 0$ (i.e. $\pi = \langle\rangle$), this implies that s_0 entails g. Hence, by Case 1 of the definition of a solution, π is a solution for P. Suppose that if GNP returns a plan π of length $k < n$, then π is a solution for P. At an invocation suppose GNP returns π of length n. The proof proceeds by showing the following. When GNP chooses an action or a method for the current goal at any invocation, then by induction, the plans returned from those calls are solutions to the HGN planning problems in those calls. Hence, by definition of solutions for P, π is a solution for P. ☐

Theorem 4 (Completeness of GNP). *Let $P = (D, s_0, g)$ be an HGN planning problem. If π is a solution for P, then a nondeterministic trace of* GNP$(D, s_0, \langle g\rangle, \langle\rangle)$ *will return π.*

Sketch of proof. The proof is by induction on n, the length of π. When $n = 0$, this implies that the empty plan is a solution for P and that $s_0 \models g$. Hence GNP would return $\langle\rangle$ as a solution. Suppose that if P has a solution of length $k < n$, then GNP will return it. At any invocation, the proof proceeds to show by induction the following. If GNP chooses an action a, then one of the nondeterministic traces of the subsequent call to GNP must return $\pi = a \circ \pi'$ where π' is a solution for the problem $P' = (D, \gamma(s_0, a), g)$. If GNP chooses a method m relevant to g with subgoals $g_1, g_2, \ldots g_l$, then there must exist a sequence of plans $\pi_1, \pi_2, \ldots, \pi_{l+1}$ that constitute π and GNP will return each π_i as a solution each goal g_i from the state $\gamma(s_0, (\pi_1 \circ \pi_2 \circ \cdots \circ \pi_{i-1}))$. Then the theorem follows. ☐

4 Heuristic Search in HGN Planning

GNP can easily be modified to incorporate heuristic functions similar to those used in classical planning. The modified algorithm, which we will call GNP^h (where h is the heuristic function), is like Algorithm 1, except that Lines 9–14 are replaced with the following:

```
sort $U$ with $h(u), \forall u \in U$
foreach $u \in U$ do
    if $u$ is an action then
        append $u$ to $\pi$
        remove $g$ from $G$
        $s \leftarrow \gamma(s, u)$
    else push sub($u$) into $G$
    $\pi \leftarrow$ GNP($D, s, G, \pi$)
    if $\pi \neq$ failure then return $\pi$
return failure
```

Intuitively, the above computation replaces the nondeterministic choice in GNP with a choice that is guided by h. GNP^h first sorts the set of actions and methods using the heuristic values for each action and method in the set. The heuristic value of an action a in a state s is given by $h(\gamma(s,a),G)$. The heuristic value of a method in s is given by $h(s,\text{sub}(m) \wedge G)$.

Once the set U of actions and methods for G is sorted, GNP^h selects the action or the method that has the best heuristic value compared to the others. With the selected action or method, GNP^h performs the same operations as GNP to update the current planning problem (s, G) and the partial plan π, and calls itself recursively.

If the return value π of the recursive invocation is a failure then GNP^h considers the second best action or method for G and performs the above operations for that action or method. Otherwise, the algorithm returns π. If the recursive invocations for all actions and methods for G in s return failure, then GNP^h returns failure.

Theorem 5. GNP^h *is sound and complete.*

Proof. The heuristic selection does not prune any actions or methods for the goal G, but only changes the order that they will be searched at any invocation of GNP^h. Therefore, the theorem follows from the proofs of Theorems 3 and 4. □

An HGN extension of the FF heuristic. Here is how the well-known FF heuristic [8] can be extended for use in HGN planning. Suppose we are given the current state s, a goal formula g and $\mathcal{A}$ and $\mathcal{M}$, the set of operator and method instances of the domain D. Let PG be the relaxed planning graph rooted at s, generated using the action set $\mathcal{A}$ exactly in the same way as FF does. Also, let $P_{PG}(i)$ denote the i^{th} proposition level and $A_{PG}(i)$ the i^{th} action level of this planning graph PG. Then, if $U_{\mathcal{A}} \subseteq \mathcal{A}$ and $U_{\mathcal{M}} \subseteq \mathcal{M}$ are the set of operator and method instances *relevant* to g in state s, the HGN heuristic function h_{FF} can be defined for all $a \in U_{\mathcal{A}}$, $m \in U_{\mathcal{M}}$ as follows :

$$\begin{aligned} h_{FF}(s,a,PG) &= 1 \\ h_{FF}(s,m,PG) &= \arg\min_i P_{PG}(i) \models \text{sub}(m) \end{aligned}$$

For an action a, $h_{FF}(s,a,PG)$ is 1 since (a subset of) g can be achieved in exactly one step from s via a. On the other hand, to estimate the length of a plan generated by a method m, we can utilize the fact that any plan generated by decomposing g using m is constrained to achieve sub(m) enroute and hence, its length is lower-bounded by $h_{FF}(s,m,PG)$ as defined in the second equation.

The function $h_{FF}(.)$ therefore provides lower-bound estimates of plan lengths for the various planning options available in the current state.

Reachability via a method. As defined above, h_{FF} extends the FF heuristic to incorporate the notion of reachability of a goal *via* a method m. Just as the notion of state reachability was defined in [2] where a set of propositions p are reachable from the initial state only if they appear in some level of the planning graph, we can define an analogous necessary condition for a goal g to be reachable from s *via* a method m as follows :

$$\exists i, P_{PG}(i) \models \text{sub}(m)$$

Therefore, if the subgoals of a method instance m are not reachable from the current state, we can prune out m. It is interesting to note that in contrast, no HTN planner can employ such a technique to prune its search space simply because there are no semantics associated with the subtasks of an HTN method.

5 Domain Authoring

One advantage of HGN planning is the ability to write a practical set of methods and prove them correct. Fig. 1 gives the HGN methods used in our experiments (see the next section) on the well-known Logistics domain [17]; we now sketch a correctness proof for them.

In most planning domains there are atoms that can appear in the subgoals of actions or methods, but not in the planning problem's goal formula. For example, the goal of a Logistics problem will not include statements such as (in-airplane package3 plane1) or (truck-in truck1 city2). We only need to consider problems $P = (D, s_0, g)$ in which the goal formula g consists entirely of atoms of the form (obj-at ?object ?location). Let $p_1, \ldots, p_n$ be the packages mentioned in g.

To prove completeness, we only need to look at cases where P is solvable. Thus, we can assume that every package mentioned in g has an initial location specified for it in s_0, that every city contains at least one truck and at least one airport, and that for every location mentioned in the planning problem, s_0 tells what city the location is in. We now sketch how the methods in Fig. 1, together with the standard operators of the Logistic domain, can be used to create plans $\pi_1, \ldots, \pi_n$ such that the concatenated plan $(\pi_1 \circ \ldots \circ \pi_n)$ will get all of the packages to their destinations.

The proof is by induction. For $i = 1, \ldots, n$, let p_i be the i'th package, and assume $\pi_1, \ldots, \pi_{i-1}$ have already been constructed. Let $s_{i-1} = \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_{i-1}))$. In s_{i-1}, suppose that p_i is at location l in city c, and that in g, p_i is at location l' in city c'. Let t and a be any truck and airport in c, let t' and a' be any truck and airport in c', and let e be any airplane. For constructing π_i, there are three cases:

1. If $l' = l$, then π_i is the empty plan.

2. If $l' \neq l$ and $c = c'$, then π_i moves t to l if needed, loads p into t, moves t to l', and unloads p.

3. If $c \neq c'$ then π_i moves t to l if needed, loads p into t, moves t to a if needed, moves e to a if needed, transfers p to e, flies e to a', moves t' to a' if needed, transfers p to t', moves t' to l' if needed, and unloads p.

5.1 Ease of Domain Authoring

In our experience, writing HGN methods can be significantly simpler than writing HTN methods. For example, consider the Logistics domain. The SHOP2 distribution at `http://www.cs.umd.edu/projects/shop/` contains the HTN definition of this domain that SHOP2 used in the 2002 International Planning Competition. Counting the clauses of SHOP2's if-then-else method construct as separate methods, the HTN domain definition contains ten methods. In contrast, our HGN domain definition (see Fig. 1) consisted of just three methods—and as shown later in our Experimental Evaluation section, this domain definition produced better results (in terms of both time and plan quality) than SHOP2's domain definition. There are three main reasons why SHOP2 needed so many more methods:

- To specify how to achieve a logical formula p in the HTN formalism, one must create a new task name t and one or more methods such that (i) the plans generated by these methods will make p true and (ii) the methods have syntactic tags saying that they are relevant for accomplishing t. If there is another method m' that makes p true but does not have such a syntactic tag, the planner will never consider using m' when it is trying to achieve p. In contrast, relevance of a method in HGN planning is similar to relevance of an action in classical planning: hence if m' effects include p, then m' is relevant for p.

- Furthermore, suppose p is a conjunct $p = p_1 \wedge \ldots \wedge p_k$ and there are methods $m_1, \ldots, m_k$ that can achieve $p_1, \ldots, p_k$ piecemeal. In HGN planning, each of these methods is relevant for p if it achieves some part of p and does not negate any other part of p. In contrast, those methods are not relevant for p in HTN planning unless the domain description includes (i) a method that decomposes t into tasks corresponding to subsets of $p_1, \ldots, p_k$, (ii) methods for those tasks, and (iii) an explicit check for deleted-condition interactions.[1] This can cause the number of HTN methods to be much larger (in some cases exponentially larger) than the number of HGN methods.

- In recursive HTN methods, a "base-case method" is needed for the case where nothing needs to be done. In recursive HGN methods, no such method is needed, because the semantics of goal achievement already provide that if a goal is already true, nothing needs to be done.

5.2 Method Ordering

One other consideration can sometime makes it easier to write HGN methods. If multiple methods are applicable to a task, SHOP2 (and GNP, unless one uses a heuristic function) tries them in the order that they appear in the input file, and the domain author must choose this order carefully to achieve efficient planning. A domain author who uses GNP with a heuristic function will not need to bother with this, because GNP will use the heuristic function to choose the order in which to try the methods.

[1] In the HTN formalism in [5], one way to accomplish (iii) is to specify t as the syntactic form $achieve(p)$, which adds a constraint that p must be true after achieving t. But that approach is inefficient in practice because it can cause lots of backtracking. In the blocks-world implementation that is included in the SHOP and SHOP2 distributions, (iii) is accomplished without backtracking by using Horn-clause inference to do some elaborate reasoning about stacks of blocks.

```
HGN Method: for transportation by truck in a city
  Pre: ((obj-at ?o ?l1) (in-city ?l1 ?c)
        (in-city ?l2 ?c) (truck ?t ?c) (truck-at ?t ?l3))
  Sub: ((truck-at ?t ?l1) (in-truck ?o ?t)
        (truck-at ?t ?l2) (obj-at ?o ?l2)))

HGN Method: for transportation between airports
  Pre: ((obj-at ?o ?a1) (airport ?a1)
        (airport ?a2) (airplane ?plane))
  Sub: ((airplane-at ?plane ?a1) (in-airplane ?o ?plane)
        (airplane-at ?plane ?a2) (obj-at ?o ?a2)))

HGN Method: for transportation between cities
  Pre: ((obj-at ?o ?l1) (in-city ?l1 ?c1) (in-city ?l2 ?c2)
        (different ?c1 ?c2) (airport ?a1) (airport ?a2)
        (in-city ?a1 ?c1) (in-city ?a2 ?c2))
  Sub: ((obj-at ?o ?a1) (obj-at ?o ?a2) (obj-at ?o ?l2)))
```

Figure 1: HGN methods for transporting a package to its goal location in the Logistics domain.

6 Experimental Evaluation

Our experiments were motivated by the following questions:

- As we discussed earlier, GNP domain descriptions often are much simpler than SHOP2 domain descriptions, because GNP automatically handles several things that SHOP2 domain authors must specify manually. *How does this affect GNP's running time and solution quality?*
- In classical planners that use heuristic functions, the heuristic function's computational overhead is outweighed by the benefit of reducing the number of nodes explored. In HGN planning, the heuristic function cannot provide as great a benefit, since HGN decomposition already focuses the search on a small part of the classical planner's search space. *Will the heuristic function provide enough benefit to outweigh its computational overhead?*

To investigate these questions, we compared SHOP2, GNP, and GNP^{FF} (GNP with the h_{FF} heuristic described earlier) in three planning domains: the well-known Logistics domain [17], and two new *Character Sequencing* domains that we constructed in order to investigate situations in which h_{FF} does or doesn't help.

- The *One-Set Character Sequencing* domain, the objective is to produce a plan for constructing a sequence of characters over some alphabet C, by starting with a single character called the *starting character* and appending characters to the end of the sequence, one at a time, until a character called the *goal character* is reached. There are several constraints that the sequence of characters must satisfy:
  - no character can appear more than once;
  - the last character in the sequence must be the goal character;
  - after each character x, the next character may be y only if the domain contains a *permissibility assertion*, $\mathrm{x} \rightarrow \mathrm{y}$, saying that y may come after x.

  There is a single planning operator, $\mathsf{append}(x, y)$, that appends y to any sequence that ends with x, provided that the precondition $x \rightarrow y$ holds.

  Planning problems in this domain are constructed by starting with a set of characters C, selecting two characters in C as a starting character and goal character, and adding a set of permissibility assertions.

  In this domain, when the number of permissibility assertions is large, we would not expect GNP^{FF} to have an advantage over GNP. In such a case the search space is likely to contain a large number of short character sequences that end with the goal character, so it should be possible for GNP to find a permissible sequence very quickly.

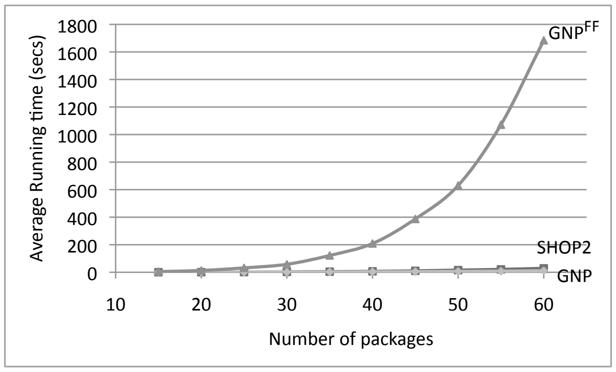

Figure 2: Average running times of SHOP2, GNP, and GNP^{FF} in the Logistics domain, as a function of the number of packages. Each data point is an average of 100 problems.

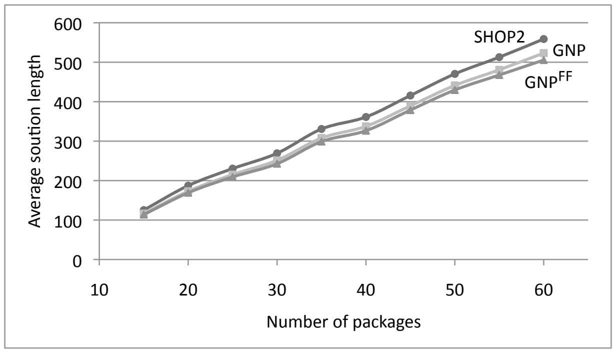

Figure 3: Average length of solutions generated by SHOP2, GNP, and GNP^{FF} in the Logistics domain, as a function of the number of packages, on the problems in Fig. 2.

- The *3-Set Character Sequencing* domain is like the One-Set Character Sequencing domain, but with the following changes:
  - C is partitioned into three subsets C_1, C_2, C_3;
  - the starting character is in C_1, and the goal character is in C_2 or C_3;
  - there is a single permissibility assertion $x \to y$ where $x \in C_1$ and $y \in C_2$;
  - there is a single permissibility assertion $z \to u$ where $z \in C_1$ and $u \in C_3$;
  - for all of the other permissibility assertions $v \to w$, both v and w must be in the same subset (i.e., both in C_1, or both in C_2, or both C_3).

  In this domain, when the number of permissibility assertions is large, we *would* expect GNP^{FF} to have an advantage over GNP. For example, suppose the goal character is in C_3. Without any guidance, it should be very easy for a planner to go from C_1 into C_2, where it can waste lots of time before backtracking from C_2 and going to C_3 instead. But if a planner has a good heuristic to guide it, it is much more likely to go directly to C_3 and thence to the goal character.

Logistics. For our Logistics experiments, we used the test problems provided in the SHOP2 distribution. In this suite, there are 10 n-package problems for each value of n, where $n = 15, 20, 25, \ldots, 60$. For GNP and GNP^{FF} we used the HGN methods in Figure 1, and for SHOP2 we used the HTN methods in the SHOP2 distribution.

As shown in Figures 2 and 3, GNP's running times were slightly faster than SHOP2's, and GNP generated solutions that were slightly shorter than SHOP2's. GNP^{FF}'s heuristic function had a huge computational overhead, hence its running times were much worse than GNP's and SHOP2's; and the heuristic function provided only a very small improvement in plan length.

One-Set Character Sequencing We randomly generated 100 planning problems for each n, with n varying from 10 to 200. To generate the permissibility assertions, we constructed near-complete digraphs in which 20% of the edges were removed at random; and created a permissibility assertion $x \to y$ for each edge of the digraph.

For GNP and GNP^{FF}, we used a single HGN method:

```
Pre: ((permissible ?x ?y))
Sub: ((in-string ?x) (in-string ?y))
```

According to this method, if there exists a permissibility assertion $x \to y$, then the problem can be solved by finding a way of adding x into the string and then appending y. By applying this method recursively, the planner can therefore do a backward search from the goal character to the initial character.

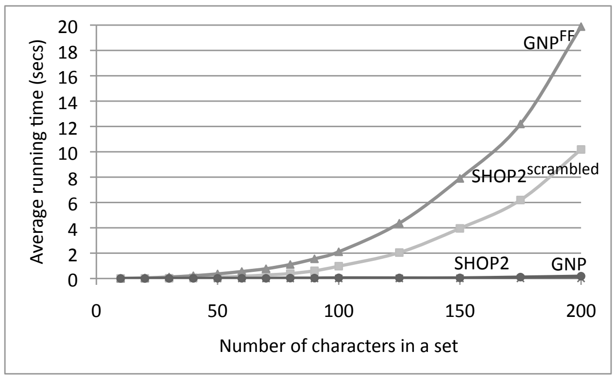

Figure 4: Average running times of SHOP2, SHOP2$^{\text{scrambled}}$, GNP, and GNP^{FF} in Character Sequencing, as a function of the number of characters. Each data point is an average of 100 problems.

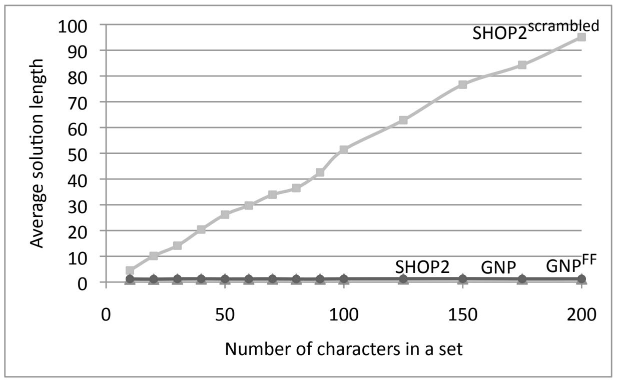

Figure 5: Average length of solutions generated by SHOP2, SHOP2$^{\text{scrambled}}$, GNP, and GNP^{FF} in Character Sequencing, as a function of the number of characters, on the problems in Fig. 4.

To accomplish the same backward search in SHOP2, we gave it three methods, one for each of the following cases: (1) goal character same as the initial character, (2) goal character one step away from the initial character, and (3) arbitrary distance between the goal and initial characters. This ensured that both GNP and SHOP2 have the same amount of domain knowledge to work with.

As we mentioned earlier in the Method Ordering section, a heuristic function like the one in GNP^{FF} removes the need for the domain author to care about the order in which the methods in the knowledge base. When multiple methods are applicable to the same task, GNP^{FF}'s performance remains essentially the same regardless of the order in which the methods appear in the knowledge base, because it uses the heuristic function to decide which method to try next. In contrast, SHOP2 tries the methods in the same order which they appear in the knowledge base, hence its performance can vary widely depending on whether the knowledge-base author picked a good order for them. Although this does not matter very much in the Logistics domain, it can matter a lot in the Character Sequencing domains. To demonstrate this, we ran SHOP2 two different ways: once with the methods in the best order to avoid backtracking (namely the order given above), and once with the methods ordered randomly. In Figures 4 and 5, we call these SHOP2 and SHOP2$^{\text{scrambled}}$, respectively.[2]

As shown in Figures 4 and 5, GNP and SHOP2 had excellent (and nearly identical) running times and solution lengths. As we expected, SHOP2$^{\text{scrambled}}$ had large running times and found poor plans. GNP^{FF}'s heuristic enabled it to find plans that were much shorter; but because of the overhead in its heuristic computations, it had the worst running times.

3-Set Character Sequencing. In order to understand situations in which heuristic computations can help GNP, we compared GNP, GNP^{FF}, and SHOP2 on 100 randomly-generated 3-Set Character Sequencing problems for each value of n, with n varying from 5 to 100. We expected that GNP^{FF}'s heuristic function might be quite useful in these problems, in order to guide it toward a solution, rather than spending huge amounts of time searching for the goal character in the wrong set.

Figures 6 and 7 show that our expectation was correct. GNP and SHOP2 did huge amounts of backtracking; hence they took very large amounts of time, produced very poor solutions, and could only solve the smallest problems. On the problems with more than a very small number of characters, both SHOP2 and GNP exceeded our time limit of 45 minutes per problem without finding a solution.

In contrast, GNP^{FF} solved all of the problems, found much better solutions, and found them much more quickly. The reason for this was that GNP^{FF}'s heuristic guided it directly to the best solution.

Discussion. The main results of our experimental study can be summarized as follows:

[2]In principle, method ordering could also matter for GNP (when using GNP without a heuristic). But in the Character Sequencing domains it doesn't matter at all, since GNP only has one method!

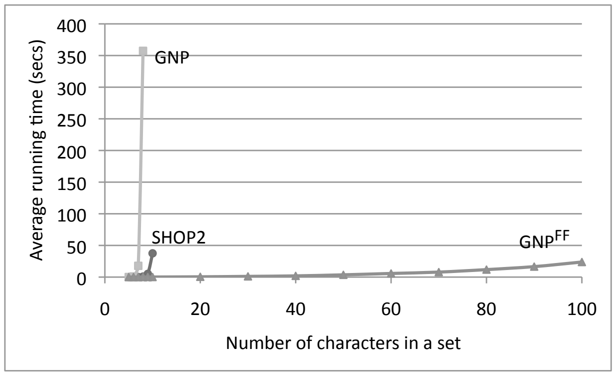

Figure 6: Average running times of SHOP2, GNP, and GNP^{FF} in 3-Set Character Sequencing, as a function of number of characters per set. Each data point is an average of 100 problems.

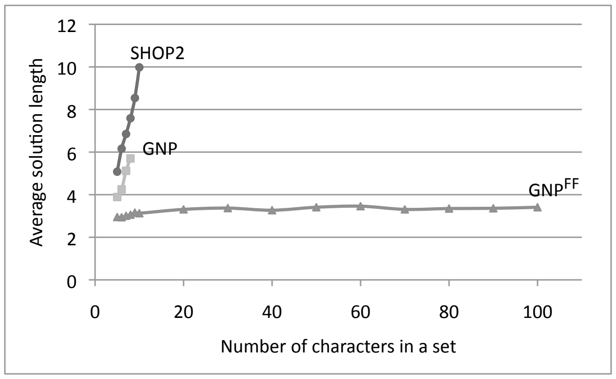

Figure 7: Average length of solutions generated by SHOP2, GNP, and GNP^{FF} in 3-Set Character Sequencing, as a function of the number of characters per set, on the problems in Fig. 6.

- GNP's performance compared favorably to that of SHOP2's, both in terms of plan quality and running times. This was achieved with *significantly* more compact domain descriptions for GNP.
- SHOP2's performance was heavily dependent on the exact order of methods in its knowledge base. This could be inferred from SHOP2$^{\text{scrambled}}$'s inferior performance in the *One-Set Character Sequencing* domain, both in terms of running time and plan quality.
- GNP^{FF}'s performance, as it turned out, was domain-dependent: in domains that are highly connected and have highly tuned knowledge bases (such as Logistics), the heuristic computation took more time than a naive search, thus not being very beneficial. On the other hand, on the *3-Set Character Sequencing* domain where the control knowledge was simple and the domain itself was not very highly connected, the heuristic used in GNP^{FF} proved to be invaluable in both boosting plan quality and significantly decreasing the running time. The latter scenario corresponds to many real-world planning problems where writing sophisticated knowledge bases is hard and prone to errors. In such a case, a heuristic-enhanced planner such as GNP^{FF} can go a long way in speeding up the planning process while keeping the control knowledge simple.

7 Related Work

A few HTN-planning papers provide constructs that look superficially similar to HGN goal decomposition; but upon closer inspection each of them turns out to be more like the HTN task decomposition in (Erol et al. [5]):

- (Erol et al. [5]) itself included a construct called a *goal task* of the form achieve(goal); but its solutions were defined identically to the solutions for other tasks, except for an additional constraint that the solution plan must produce a state in which the goal is true.
- In both [9] and [12], the tasks are abstract actions. In the former, abstract actions can be decomposed using *reduction schemas* whereas in the latter, the description of an abstract action directly includes such schemas. But the decomposition mechanisms still are conceptually similar to (Erol et al. [5]), because a reduction schema is applicable to an abstract action o only if the schema specifically mentions o. In contrast, our HGN tasks do not have names, and an HGN method can be used on any HGN task for which it is relevant.

There are several classical planning algorithms that guide the search using heuristic estimates of the distance to a solution. The heuristic estimates are sometimes based on planning graphs (e.g., FastForward [8]) and sometimes on other relaxations of the planning problem (e.g., HSP [3]). GNP can utilize any of these heuristics in its HGN planning process, provided that the heuristic computation only requires an initial state and a goal formula as input.

8 Conclusions

Our original motivation for HGN planning was to provide a task semantics that corresponded readily to the goal semantics of classical planning, to make it easier to tell whether or not a set of HGN methods is correct. But our work also produced two other benefits that we had not originally expected: writing HGN methods can be much simpler than writing HTN methods, and the HGN formalism can easily incorporate HGN extensions of classical-style heuristic functions to guide the search. This suggests to us that HGN planning has the potential to be very useful both for research purposes and in practical applications.

With that in mind, we have several ideas for future work:

- We intend to generalize HGNs to allow *partial* sets of methods analogous to the ones in (Alford et al. [1]). From a theoretical standpoint, this will provide an interesting hybrid of task decomposition and classical planning. From a practical standpoint, it will make it even easier to write HGN domain descriptions while preserving much of the efficiency advantages of HGN planning.
- Replanning in dynamic environments is becoming an increasingly important research topic. We believe HGN planning is a promising approach for this topic.
- HTN planning has been extended to accommodate actions with nondeterministic outcomes [10], temporal planning [4, 7], and to consult external information sources during planning [11]. It should be straightforward to make similar extensions to HGN planning.

Acknowledgments. This work was supported in part by DARPA and and U.S. Army Research Laboratory contract W911NF-11-C-0037. The information in this paper does not necessarily reflect the position or policy of the U.S. Government, and no official endorsement should be inferred.

References

[1] Ron Alford, Ugur Kuter, and Dana S. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *IJCAI*, July 2009.

[2] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *IJCAI*, pages 1636–1642, 1995.

[3] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *ECP*, Durham, UK, 1999.

[4] L. Castillo, J. Fdez-Olivares, O. Garcıa-Pérez, and F. Palao. Efficiently handling temporal knowledge in an HTN planner. In *ICAPS*, 2006.

[5] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.

[6] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. May 2004.

[7] R.P. Goldman. Durative planning in HTNs. In *ICAPS*, 2006.

[8] J. Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[9] S. Kambhampati, A. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *AAAI*, pages 882–888, 1998.

[10] Ugur Kuter, Dana S. Nau, Marco Pistore, and Paolo Traverso. Task decomposition on abstract states, for planning under nondeterminism. *Artif. Intell.*, 173:669–695, 2009.

[11] Ugur Kuter, Evren Sirin, Dana S. Nau, Bijan Parsia, and James Hendler. Information gathering during planning for web service composition. *JWS*, 3(2-3):183–205, 2005.

[12] Bhaskara Marthi, Stuart J. Russell, and Jason Wolfe. Angelic Hierarchical Planning: Optimal and Online Algorithms. In *ICAPS*, 2008.

[13] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Héctor Muñoz-Avila, J. William Murdock, Dan Wu, and Fusun Yaman. Applications of SHOP and SHOP2. *IEEE Intell. Syst.*, 20(2):34–41, March-April 2005.

[14] Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. The SHOP planning system. *AI Mag.*, 2001.

[15] Dana S. Nau, Héctor Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *IJCAI*, Seattle, August 2001.

[16] Austin Tate, Brian Drabble, and Jeff Dalton. O-Plan: a knowledge-based planner and its application to logistics. In Austin Tate, editor, *Advanced Planning Technology*. May 1996.

[17] Manuela M. Veloso. Learning by analogical reasoning in general problem solving. PhD thesis CMU-CS-92-174, Carnegie Mellon University, 1992.

[18] D. E. Wilkins and R. V. Desimone. Applying an AI planner to military operations planning. In M. Fix and M. Zweben, editors, *Intelligent Scheduling*, pages 685–709. 1994.