A Hierarchical Goal-Based Formalism and Algorithm for Single-Agent Planning

Vikas Shivashankar[1] Ugur Kuter[2] Dana Nau[1] Ron Alford[1]
[1]University of Maryland, College Park, Maryland 20742 USA
[2]Smart Information Flow Technologies, Minneapolis, Minnesota 55401 USA
svikas@cs.umd.edu ukuter@sift.net nau@cs.umd.edu ronwalf@cs.umd.edu

ABSTRACT

Plan generation is important in a number of agent applications, but such applications generally require elaborate *domain models* that include not only the definitions of the actions that an agent can perform in a given domain, but also information about the most effective ways to generate plans for the agent in that domain. Such models typically take a large amount of human effort to create.

To alleviate this problem, we have developed a hierarchical goal-based planning formalism and a planning algorithm, GDP (Goal-Decomposition Planner), that combines some aspects of both HTN planning and domain-independent planning. For example, it allows the planning agent to use domain-independent heuristic functions to guide the application of both methods and actions.

This paper describes the formalism, planning algorithm, correctness theorems, and the results of a large experimental study. The experiments show that our planning algorithm works as well as the well-known SHOP2 HTN planner, using domain models only about half the size of SHOP2's.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Plan execution, formation, and generation*

General Terms

Algorithms

Keywords

AI planning, hierarchical planning, goal decomposition

1. INTRODUCTION

The ability to do effective planning is important for a wide variety of computerized agents. Examples include robotic agents (e.g., the Mars rovers [27]), game-playing agents (e.g., in card games [29, 25] and real-time strategy games [5]), web-service agents [19], and others. To build capable planners for agent environments, generally the planner must incorporate a *domain model* that includes not only the definitions of the basic actions that the agent can perform, but also information about the most effective ways to generate plans in the agent's environment. One way to incorporate domain models into planning agents is to custom-build a planning module for the application at hand. This approach was used successfully in many of the examples just mentioned, but it usually requires a huge development effort.

Another approach is to use a *domain-configurable* planner which reads the domain model as part of its input. Most such planners use *Hierarchical Task Network (HTN) planning*, in which the domain model includes *methods* for accomplishing *tasks* by dividing them into smaller and smaller subtasks. This approach has been used in a wide variety of planning domains, e.g., [33, 6, 22, 21, 26]. Writing a new domain model usually is much less work than building a new domain-specific planner, but in most cases it still requires a great deal of human effort.

We have developed a *Hierarchical Goal Network (HGN) planning* formalism and algorithm similar to the HTN formalism and algorithm in the SHOP planner [24], but with some important differences that make it easier to develop domain models. In the SHOP formalism, each task is a separate syntactic entity whose semantics depends entirely on what methods match it. In contrast, HGN tasks consist of *initial conditions* and *goal conditions* with the same semantics as in classical planning; and HGN methods and actions have applicability and relevance conditions similar to the ones for actions in classical planning. Our results are as follows:

Formalism: The HGN formalism provides (provably) as much expressive power as SHOP's HTN formalism (or equivalently, SHOP2's HTN formalism restricted to totally-ordered subtasks). Moreover, in contrast to problems with soundness (see Section 2) in HTN translations of classical planning domains, soundness is guaranteed for all HGN domain models of classical domains; and it is easier to analyze whether HGN translations are complete.

Planning algorithm: Our planning algorithm, *GDP (Goal Decomposition Planner)*, is sound and complete. It is similar in several ways to SHOP, but the HGN task and method semantics provide much more flexibility in applying methods and actions. For example, in writing GDP domain models we don't have to commit to a task name in the methods: we just specify what should be achieved instead of how to achieve it, and let the planner decide which methods and actions are relevant and applicable.

Heuristic function: The HGN task and method semantics enable the development of heuristic functions similar to the ones in classical planners (e.g., FF [16] and HSP [3]). We provide one such heuristic function, for optional use in GDP, to guide the selection of both methods and operators. For a domain model to work well, it is important to specify the order in which a set of methods and operators should be tried when more than one of them is applicable; and the heuristic function can make this easier by enabling the planner to deduce the best order on its own.

Experimental results: We have done an extensive experimen-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

tal comparison of three versions of GDP, two versions of SHOP2, and FF, in five different planning domains. On average, GDP's domain models were only about half as large as the equivalent SHOP2 domain models, yet provided roughly the same performance (i.e., planning time and plan lengths). The HGN domain models were so much simpler because the HGN task and method semantics obviates the need for a plethora of extra methods and bookkeeping operations needed in SHOP2 domain models.

2. RELATED WORK

Over the years, several well-known researchers (e.g., [10, 17]) have argued for combining HTN planning with other techniques, and several of the older HTN planners (e.g., SIPE [33, 32] and O-PLAN [6, 30]) combined hierarchical decomposition with goal-directed partial-order planning.[1] But in Erol *et al*'s influential HTN formalism [8] and most subsequent HTN planning research (e.g., [14, 23, 22]; a notable exception is [20]), the definition of a solution is tied so intimately to HTN decomposition that the planning problems are solvable in no other way.[2]

The lack of correspondence between tasks and goals makes it hard to translate classical planning problems correctly into HTN domain models (e.g., a common error is to translate a goal $g_1 \wedge g_2$ into a task sequence $\langle \mathsf{achieve}(g_1), \mathsf{achieve}(g_2) \rangle$, ignoring the possibility that the plan for $\mathsf{achieve}(g_2)$ may delete the previously achieved goal g_1). In the 2000 International Planning Competition, SHOP [24] was disqualified because of an incorrect HTN translation that caused SHOP to return an incorrect answer. The lack of correspondence between tasks and goals has also interfered with recent efforts to combine HTN planning with classical planning [1, 13], necessitating several *ad hoc* modifications and restrictions.

3. FORMALISM

Classical planning. Following Ghallab et al. [14, Chap. 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.

The next theorem proves that HGN planning is *sound*: any HGN solution is also a solution to the corresponding classical problem.

THEOREM 1 (HGN SOUNDNESS). *Let $D = (D', M)$ be an HGN planning domain. For every (s_0, 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 that in D', π is executable in s_0 and $\gamma(s_0, \pi) \models g$. □

THEOREM 2 (HGN COMPLETENESS). *For every classical planning domain D, there is a set of HGN methods M such that the classical planning problem $P = (D, s_0, g)$ and the HGN planning problem $P' = ((D, M), s_0, g)$ have the same set of solutions.*

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 following two theorems prove that the HGN formalism provides expressive power equal to that of SHOP's HTN formalism:

THEOREM 3 (HTN EXPRESSIVITY). *For any HGN problem (D, s_0, g_0), there exists a totally-ordered HTN problem (D', s_0, t_{g_0}) such that (D, s_0, g_0) is solvable if and only if (D', s_0, t_{g_0}) is solvable.*

[1] PRS [12] is also a hierarchical goal-based reasoning system, but its primary focus is *reactive execution* in dynamic environments; the actual planning is rather limited.

[2] One might expect Erol *et al.*'s formalism to allow classical goal achievement, because it allows *goal tasks* of the form $\mathsf{achieve}(\mathsf{goal})$. But just as with any other task, a goal task's solution plans can only be constructed by HTN decomposition: the only difference is a constraint that the goal must be true after executing the plan.

Proof Sketch. We proceed to translate an HGN planning problem (D, s_0, g_0) into an HTN planning problem as follows: each goal formula g in D is represented by a task symbol t_g; g_0 is represented by the task symbol t_{g_0}. For each HGN method $\langle pre, \langle g_1, \ldots, g_k \rangle \rangle$, we create a new HTN method accomplishing task t_{g_k} with preconditions pre and subtasks $\langle t_{g_1}, \ldots, t_{g_k} \rangle$. Then for each t_g, we create an HTN method having a precondition of g and no subtasks. Also for every method or operator u relevant to g, we have a method accomplishing t_g having a precondition of $\neg g$ and subtasks $\langle t_u, t_g \rangle$, t_u being the task symbol corresponding to u. We then return the HTN problem $(D' \cup O, s_0, t_{g_0})$ where D' is the set of translated HTN methods and O is the set of planning operators.

It is now easy to show that any HGN decomposition trace can be mapped to a corresponding trace of the HTN problem thus constructed and vice-versa. Thus the theorem follows. □

THEOREM 4 (HGN EXPRESSIVITY). *For any totally-ordered HTN planning problem (D, s_0, t_0), there is an HGN planning problem (D', s_0, g_{t_0}) such that (D, s_0, t_0) is solvable if and only if (D', s_0, g_{t_0}) is solvable.*

Proof Sketch. To translate an HTN planning problem[3] (D, s_0, t_0), we create predicates $fin_t(.)$ for each task $t(.)$ to represent task completion. We add an extra predicate *lead* that is asserted by an artificial operator with no preconditions. We have artificial operators *assert-fin-t(.)* for each task symbol t that has precondition $\langle lead \rangle$ and effect $\langle \neg lead, fin_t(.) \rangle$ Each HTN method for task t with subtasks $\langle t_1, t_2, \ldots, t_n \rangle$ is now converted to an HGN method with the same preconditions and a sequence of subgoals $\langle fin_{t_1}, \neg fin_{t_1}, fin_{t_2}, \neg fin_{t_2}, \ldots fin_{t_n}, \neg fin_{t_n}, lead, fin_t \rangle$. The $\neg fin(.)$ subgoals are used to cleanup the state for future decompositions. The HGN planning problem is $(D' \cup O, s_0, fin_{t_0})$, where D' is the set of translated HGN methods and O is the set of classical planning operators and additional artificial operators described above. It is now easy to show that every HTN decomposition trace can be mapped to a corresponding trace of the HGN planning problem thus constructed and vice-versa. The theorem follows. □

The above theorems provide procedures to translate HGN planning problems to HTN problems and vice-versa in low-order polynomial time. This proves that HGN planning has the same expressive power as totally-ordered HTN planning.

Let HGN-PLAN-EXISTENCE be the following problem: *Given an HGN planning problem P, is there a plan that solves P?*

THEOREM 5. HGN-PLAN-EXISTENCE *is decidable.*

Proof Sketch. Erol et al. [9] prove that the plan existence problem for totally-ordered HTN planning is decidable. From this and Theorem 3, the result immediately follows. □

4. PLANNING ALGORITHM

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

In Line 3, if G is empty then the goal has been achieved, so GDP returns π. Otherwise, GDP selects the first goal g in G (Line 4). If g is already satisfied, GDP 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 GDP returns failure. Otherwise, GDP nondeterministically chooses an action/method u from U.

[3]We assume a single task t_0 in the initial task network; this is without loss of generality as we can replace a totally-ordered initial task network with an artificial toptask and add an extra method decomposing the toptask to the initial task network.

Algorithm 1: A high-level description of GDP. 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 GDP(D, s, G, π)
2  begin
3      if G is empty then return π
4      g ← the first goal formula in G
5      if s ⊨ g then
6          remove g from G and return GDP(D, s, G, π)
7      U ← {actions and method instances that are relevant
               for g and applicable to s}
8      if U = ∅ then return failure
9      nondeterministically choose u ∈ U
10     if u is an action then
11         append u to π and set s ← γ(s, u)
12     else insert sub(u) at the front of G
13     return GDP(D, s, G, π)
14 end
```

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

4.1 Formal Properties

The following theorems show that GDP is sound and complete:

THEOREM 6 (GDP SOUNDNESS). *Let $P = (D, s_0, g)$ be an HGN planning problem. If a nondeterministic trace of* GDP$(D, s_0, \langle g \rangle, \langle \rangle)$ *returns a plan π, then π is a solution for P.*

Proof Sketch. 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 GDP returns a plan π of length $k < n$, then π is a solution for P. At an invocation suppose GDP returns π of length n. The proof proceeds by showing the following. When GDP 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 7 (GDP COMPLETENESS). *Let $P = (D, s_0, g)$ be an HGN planning problem. If π is a solution for P, then a nondeterministic trace of* GDP$(D, s_0, \langle g \rangle, \langle \rangle)$ *will return π.*

Proof Sketch. 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 GDP would return $\langle \rangle$ as a solution. Suppose that if P has a solution of length $k < n$, then GDP will return it. At any invocation, the proof proceeds to show by induction the following. If GDP chooses an action a, then one of the nondeterministic traces of the subsequent call to GDP must return $\pi = a \circ \pi'$ where π' is a solution for the problem $P' = (D, \gamma(s_0, a), g)$. If GDP 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 GDP 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. □

4.2 Domain-Independent Heuristics

GDP can easily be modified to incorporate heuristic functions similar to those used in classical planning. The modified algorithm, which we will call GDP-h (where h is the heuristic function) in

the experiments, is like Algorithm 1, except that Lines 9–13 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 π; remove g from G; $s \leftarrow \gamma(s, u)$
  else push sub(u) into G
  $\pi \leftarrow \text{GDP}(D, s, G, \pi)$
  if $\pi \neq$ failure **then return** π
return failure

Intuitively, this replaces the nondeterministic choice in GDP with a deterministic choice dictated by h. GDP-h uses h to order U, then attempts to decompose the current goal g in that order.

As an example, here is how we compute a variation of the Relaxed Graphplan heuristic used by the FF planner [16]. At the start of the planning process, we generate a relaxed planning graph PG from the start state s_0 to its fixpoint. Let $l_{PG}(p)$ be the first propositional level in which p appears in PG. Then $h_{s,G}(u)$, the heuristic value of applying action/method u in a state s to achieve the goals in the list G, is as follows:

$$h_{s,G}(u) =$$

$$\begin{cases} 1 + \max\limits_{p \in G} l_{PG}(p) - \max\limits_{p \in \gamma(s,u)} l_{PG}(p), & \text{if } u \text{ is an action,} \\ \max\limits_{p \in G \cup sub(u)} l_{PG}(p) - \max\limits_{p \in s} l_{PG}(p), & \text{if } u \text{ is a method.} \end{cases}$$

Intuitively, what h estimates is the distance between the first level in which the literals in G are asserted and the first level in which the current state is asserted. When u is a method, since any plan generated via u has to achieve $sub(u)$ enroute, it considers the set $G \cup sub(u)$ instead as the goal.

Note that this gives weaker heuristic values than the original FF heuristic since we do not generate a relaxed plan and use its length as the heuristic value. However, we use this variant of the heuristic since it is much more efficiently computable without compromising too much on search control. The strength of the heuristic is not as critical here as in classical planning, since the HGN methods themselves constrain what part of the space gets searched.

5. EXPERIMENTAL EVALUATION

We implemented GDP in Common Lisp, and compared it with SHOP2 and the classical planner FF in five different planning domains:[4] These included the well-known Logistics [31], Blocks-World [2], Depots [11], and Towers of Hanoi [1] domains, and a new *3-City Routing* domain that we wrote in order to provide a domain in which the planners' domain models would not be of much help. The following questions motivated our experiments:

- *How does GDP's performance (plan quality and running time) compare with SHOP2's?* In order to investigate this question, we were careful to use domain models for SHOP2 and GDP that encoded basically the same control information.[5]
- *What is the relative difficulty of writing domain models for GDP and SHOP2?* We had no good way to measure this directly;[6] but as a proxy for it, we (i) measured the relative sizes of the SHOP2 and GDP domain models, and (ii) examined the domain models to find out the reasons for the difference in size.
- *How useful is GDP-h's heuristic function when the domain model is strong?* For this, we compared GDP-h with GDP on the Logistics, Blocks World, and Depots domains.
- *When the domain model is weak, how much help does GDP-h's heuristic function provide?* For this, we compared GDP-h's performance with GDP's on the 3-City Routing domain.
- *Since GDP-h's heuristic function is loosely based on FF's, how does GDP-h's performance compare to FF's?* For this purpose, we included FF in our experiments.
- *Is GDP as sensitive as SHOP2 is to the order in which the methods appear in the domain model?* To investigate this question, we took our domain models for SHOP2 and GDP, and rearranged the methods into a random order. In experimental results that follow, we use the names SHOP2-r and GDP-r to refer to SHOP2 and GDP with those domain models.

The GDP source code, and the HGN and HTN domain models used in our experiments, are available at http://www.cs.umd.edu/projects/planning/data/shivashankar12hierarchical/.

Method for using truck ?t to move crate ?o
  from location ?l1 to location ?l2 in city ?c:
  Head: (move-within-city ?o ?t ?l1 ?l2 ?c)
  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)))

Method for using airplane ?plane to move crate ?o
  from airport ?a1 to airport ?a2:
  Head: (move-between-airports ?o ?plane ?a1 ?a2)
  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)))

Method for moving ?o from location ?l1 in city ?c1
  to location ?l2 in city ?c2, via airports ?a1 and ?a2:
  Head: (move-between-cities ?o ?l1 ?c1 ?l2 ?c2 ?a1 ?a2)
  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 4: HGN methods for transporting a package to its goal location in the Logistics domain.

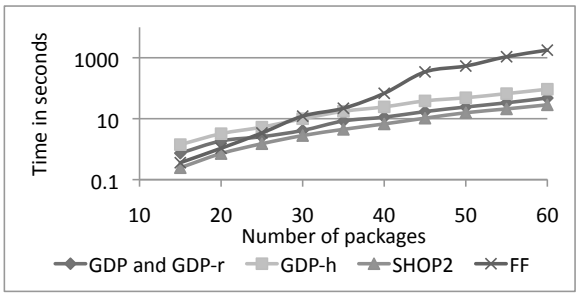
5.1 Planning Performance

To compile and execute GDP, GDP-h, and SHOP2, we used Allegro Common Lisp 8.0. For FF, we used the open-source C implementation from the FF web site. All experiments were run on 2GHz dual-core machines with 4GB RAM. We set a time limit of

[4]We used SHOP2 instead of SHOP for two reasons: (1) its algorithm is identical to SHOP's when restricted to totally-ordered subtasks, and (2) since its implementation includes many enhancements and optimizations not present in SHOP, it provides a more rigorous test of GDP.

[5]An important aspect of SHOP2's domain models is the use of Horn-clause inference to infer some of the preconditions. So that we could write GDP domain models equivalent to SHOP2's, we included an identical Horn-clause inference engine in GDP.

[6]That would have required a controlled experiment on a large number of human subjects, each of whom has equal amounts of training and experience with both GDP and SHOP2. We have no feasible way to perform such an experiment.



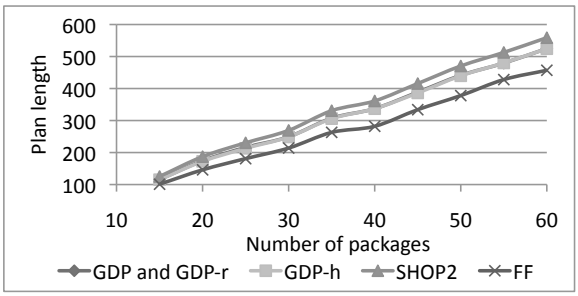
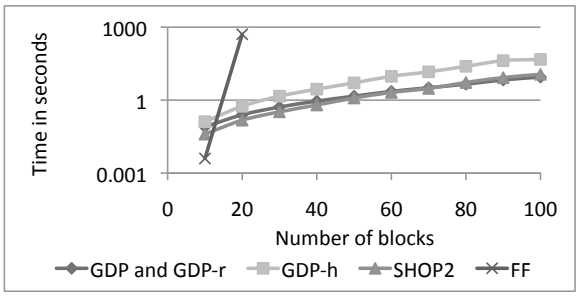


Figure 1: Average running times (in logscale) and plan lengths in the Logistics domain, as a function of the number of packages. Each data point is an average of the 10 problems from the SHOP2 distribution. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the methods had mutually exclusive preconditions.



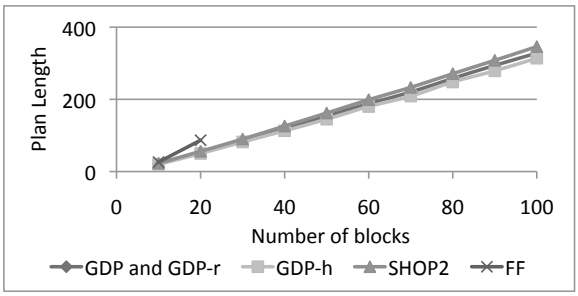


Figure 2: Average running times (in logscale) and plan lengths in the Blocks World domain, as a function of the number of blocks. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive. FF was unable to solve problems involving more than 20 blocks.

two hours per problem, and data points not solved within the required time limit were discarded.

The Logistics Domain. For SHOP2, we used the Logistics domain model in the SHOP2 distribution. For GDP and GDP-h, we wrote the methods in Fig. 4 (these methods are easy to prove complete [28]). For the experiments, we used the Logistics Domain problems in the SHOP2 distribution. These included ten n-package problems for each of $n = 15, 20, 25, \ldots, 60$.

Figure 1 shows a comparison of running times and plan lengths of the planners in this domain. The running times of GDP, GDP-h and SHOP2 were very similar, showing that even on easy domains with strong domain models, the heuristic does not add much overhead to GDP-h's running time. FF's running times, however, grew much faster: with 60 packages, FF was nearly two orders of magnitude slower than SHOP2.

The plans produced by GDP and GDP-h were of nearly the same length, and the plans produced by SHOP2 were slightly longer. FF produced the shortest plans; this indicates that its heuristic function was slightly stronger than the relaxed version we used in GDP-h.

SHOP2-r did not terminate on any of the instances, while GDP-r performed identically to GDP. In fact, we observed that the same was true across all of the domains in our experimental study. We defer the explanation of this to Section 5.3.

The Blocks World. For SHOP2, we used the domain model included in SHOP2's distribution. For GDP and GDP-h we used a much more compact domain model consisting of three methods (shown here as pseudocode):

- **To achieve** $\texttt{on}(x, y)$
  precond: y is in its final position[7]
  subgoals: achieve $\texttt{clear}(x)$, $\texttt{clear}(y)$ and $\texttt{on}(x, y)$
- **To achieve** $\texttt{clear}(x)$
  precond: $\texttt{on}(y, x)$
  subgoals: achieve $\texttt{clear}(y)$ and then $\texttt{clear}(x)$
- **To achieve** $\texttt{on-table}(x)$
  precond: None
  subgoals: achieve $\texttt{clear}(x)$ and then $\texttt{on-table}(x)$

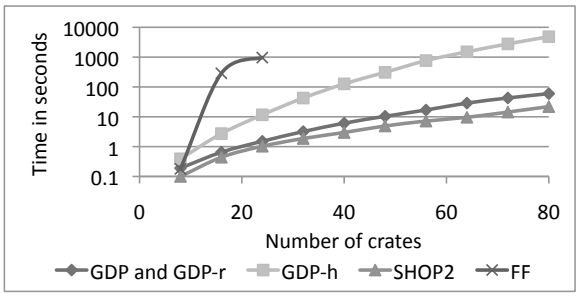
As shown in Figure 2, GDP and SHOP2 took nearly identical times to solve the problems, with GDP-h taking slightly longer due to its heuristic computation overhead. FF, which is known to have problems with the Blocks World [2], was unable to solve problems with more than 20 blocks.

As shown in the figure, GDP, GDP-h and SHOP2 produced solution plans of similar length, with GDP-h producing the shortest plans. FF produced significantly longer plans than the other three planners, even for the problems it managed to solve.

The Depots Domain. For SHOP2, we used the Depots domain model from the SHOP2 distribution. For GDP and GDP-h, we simply stitched together relevant parts of the Logistics and Blocks-World domain models, and adapted them to obtain an HGN Depots domain model that encoded the same control information.

As shown in Figure 3, GDP and SHOP2 took similar times to solve the problems. However, GDP-h's running times grew much faster than GDP or SHOP2, indicating that the overhead of the heuristic can increase with the complexity of the domain. FF was unable to solve any problems of size greater than 24 crates.

[7]Inferred using Horn clauses (see footnote 5).



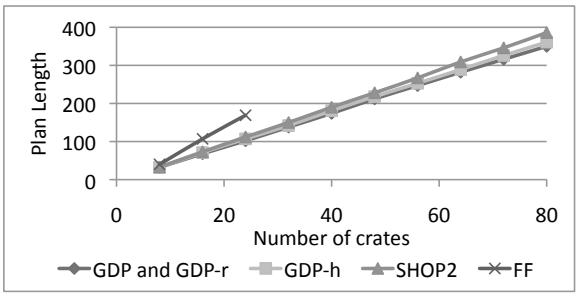
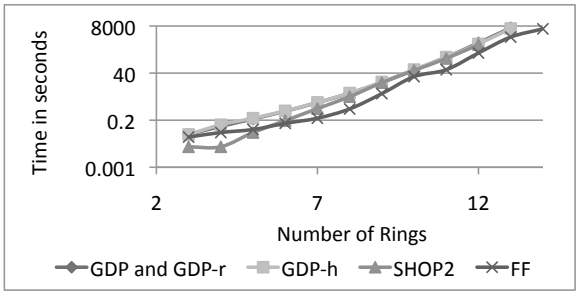


Figure 3: Average running times (in logscale) and plan lengths in the Depots domain, as a function of the number of crates. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive. FF was unable to solve problems involving more than 24 crates.



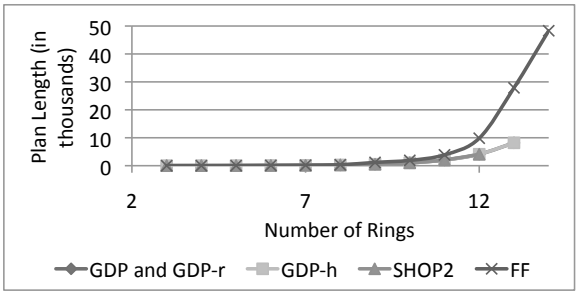


Figure 5: Average running times (in logscale) and plan lengths in the Towers of Hanoi domain, as a function of the number of rings. Each data point is an average of 10 runs. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive.

With respect to plan lengths, GDP and GDP-h produced almost identical plans, with SHOP2 producing slightly longer plans than GDP. For the problem sizes it could handle, FF produced significantly longer plans than the other three planners.

Towers of Hanoi. We wrote domain models for SHOP2 and GDP that encoded an algorithm to produce optimal solution plans (i.e., length $2^n - 1$ for an n-ring problem).

Figure 5 shows the planners' runtimes and plan lengths. As expected, GDP, GDP-h and SHOP2 returned optimal plans whereas FF returned significantly sub-optimal plans.

However, while GDP, GDP-h, SHOP2 and FF had similar runtimes up to problems of size 12, SHOP2 could not solve the larger problems due to a stack overflow, and GDP could not solve the 14-ring problem within the time limit. We believe this is basically an implementation issue: both GDP and SHOP2 had recursion stacks of exponential size, whereas FF (since it never backtracks) did not.

3-City Routing. In the four planning domains discussed above, the GDP and SHOP2 domain models pruned the search space enough that GDP-h's heuristic function could not reduce it much further (if at all). In order to examine the performance of the planners in a domain with a weak domain model, we constructed the *3-City Routing* domain. In this domain, there are three cities c_1, c_2 and c_3, each containing n locations internally connected by a network of randomly chosen roads. In addition, there is one road between a randomly chosen location in c_1 and a randomly chosen location in c_2, and similarly another road between locations in c_2 and c_3. The problem is to get from a location in c_1 or c_3 to a goal location in c_2.

We randomly generated 25 planning problems for each value of n, with n varying from 10 to 100. For the road networks, we used near-complete graphs in which 20% of the edges were removed at random. Note that while solutions to such problems are typically very short, the search space has extremely high branching factor, i.e. of the order of n. For GDP and GDP-h, we used a single HGN method, shown here as pseudocode:

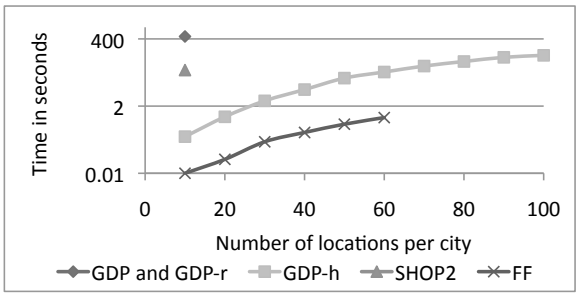
- **To achieve** `at(`b`)`
  precond: `at(`a`)`, `adjacent(`c, b`)`
  subgoals: achieve `at(`c`)` and then `at(`b`)`

By applying this method recursively, the planner can do a backward search recursively from the goal location to the start location.

To accomplish the same backward search in SHOP2, we needed to give it three methods, one for each of the following cases: (1) goal location same as the initial location, (2) goal location one step away from the initial location, and (3) arbitrary distance between the goal and initial locations.

As Figure 6 shows, GDP and SHOP2 did not solve the randomly generated problems except the ones of size 10, returning very poor solutions and taking large amounts of time in the process. GDP-h, on the other hand solved all the planning problems quickly, returning near-optimal solutions. The reason for the success of GDP-h is that the domain knowledge specified above induce an unguided backward search in the state space and the planner uses the domain-independent heuristic to select its path to the goal.

FF was able to solve all problems up to $n = 60$ locations, after which it could not even complete parsing the problem file. We believe this has to do with FF grounding all the actions right in the beginning, which it could not do for the larger problems.



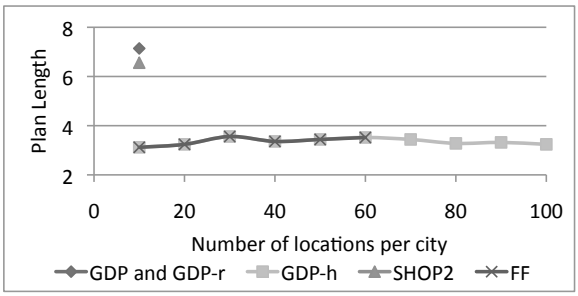


Figure 6: Average running times (in logscale) and plan lengths in the 3-City Routing domain, as a function of the number of locations per city. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it couldn't solve any problems. FF couldn't solve problems involving more than 60 locations while GDP and SHOP2 could not solve problems with more than 10 locations. GDP and GDP-r performed identically because there was only one method in the domain model.

5.2 Domain Authoring

When writing the domain models for our experiments, it seemed to us that writing the GDP domain models was easier than writing the SHOP2 domain models—so we made measurements to try to verify whether this subjective impression was correct.

Figure 7 compares the sizes of the HGN and HTN domain descriptions of the planning domains. In almost all of them, the domain models for GDP were much smaller than those for SHOP2. There are three main reasons why:

- 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: if the effects of m' 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.[8] 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.

The Towers of Hanoi domain was the only one where the HGN domain model was larger than the corresponding HTN domain model. In this domain, the HGN domain model needed two extra actions, *enable* and *disable*, to alternately insert and delete a special atom in the state. They were needed in order to control the applicability of the *move* operator to ensure optimality.

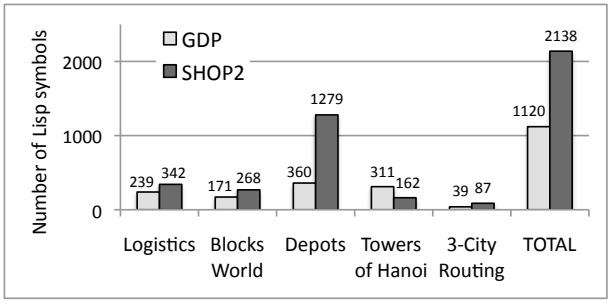


Figure 7: Sizes (number of Lisp symbols) of the GDP and SHOP2 domain models.

5.3 Discussion

We have seen from our experimental study that HGN domain models are considerably more succinct than the corresponding HTN models. We also saw that this compactness came at no extra cost; GDP's performance compared favorably to that of SHOP2's across all domains. Runtimes of the heuristic-enhanced planner GDP-h were, for the most part, comparable to those of GDP's and SHOP2's, indicating that our heuristic does not add a significant overhead to the planning time. Lengths of plans returned by GDP-h were nearly always better than GDP's and SHOP2's. This difference was especially amplified in cases where the planners had weak domain models; in such cases, the heuristic provided critical search control to GDP-h, thus helping it terminate quickly with good solutions.

In our experiments, SHOP2-r did not solve any of the problems. The reason for this was SHOP2's heavy reliance on the method order in its domain model, especially the placement of "base-cases" for recursion. For GDP-r, shuffling HGN methods had no effect at all on performance. This was because the methods in our HGN domain models had mutually exclusive preconditions, hence at most one of them was applicable. In domains where more than one method is applicable at once, GDP-r should (like SHOP2-r) perform badly when presented with methods in the wrong order.

[8] In the HTN formalism in [7], 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 in the SHOP2 distribution, (iii) is accomplished without backtracking by using Horn-clause inference to do some elaborate reasoning about stacks of blocks.

6. CONCLUSIONS

Our original motivation for HGN planning was to provide a task semantics that corresponded readily to the goal semantics of classical planning and gave stronger soundness guarantees when applied to classical planning domains. But our work also produced two other benefits that we had not originally expected: writing HGN methods was usually much simpler than writing HTN methods, and the HGN formalism can easily incorporate HGN extensions of classical-style heuristic functions to guide the search.

Our proof that HGN planning is as expressive as totally-ordered HTN planning means that it is capable of encoding complicated control knowledge, one of the main strengths of HTN planning. This suggests 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:

- GDP currently supports only totally ordered subtasks. We intend to generalize HGNs to allow partially-ordered subtasks.
- We intend to generalize HGNs to allow *partial* sets of methods analogous to the ones in [1]. This will provide an interesting hybrid of task decomposition and classical planning. Furthermore, it will make writing HGN domain models even easier while preserving 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 [18], temporal planning [4, 15], and to consult external information sources [19]. It should be straightforward to make similar extensions to HGN planning.

Acknowledgments. This work was supported, in part, by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037, and by a UMIACS New Research Frontiers Award. The views expressed are those of the authors and do not reflect the official policy or position of the funders.

7. REFERENCES

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

[2] F. Bacchus. The AIPS '00 planning competition. *AI Mag.*, 22(1):47–56, 2001.

[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] M. Chung, M. Buro, and J. Schaeffer. Monte carlo planning in RTS games. In *IEEE Symp. Comp. Intel. Games*, 2005.

[6] K. Currie and A. Tate. O-Plan: The open planning architecture. *Artif. Intell.*, 52(1):49–86, 1991.

[7] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.

[8] K. Erol, J. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254, June 1994. ICAPS 2009 influential paper honorable mention.

[9] K. Erol, J. Hendler, and D. S. Nau. Complexity results for hierarchical task-network planning. *AMAI*, 18:69–93, 1996.

[10] T. A. Estlin, S. Chien, and X. Wang. An argument for a hybrid HTN/operator-based approach to planning. In *ECP*, pages 184–196, 1997.

[11] M. Fox and D. Long. International planning competition, 2002. http://planning.cis.strath.ac.uk/competition.

[12] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.

[13] A. Gerevini, U. Kuter, D. S. Nau, A. Saetti, and N. Waisbrot. Combining domain-independent planning and HTN planning. In *ECAI*, pages 573–577, July 2008.

[14] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. May 2004.

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

[16] J. Hoffmann and B. Nebel. The FF planning system. *JAIR*, 14:253–302, 2001.

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

[18] U. Kuter, D. S. Nau, M. Pistore, and P. Traverso. Task decomposition on abstract states, for planning under nondeterminism. *Artif. Intell.*, 173:669–695, 2009.

[19] U. Kuter, E. Sirin, D. S. Nau, B. Parsia, and J. Hendler. Information gathering during planning for web service composition. *JWS*, 3(2-3):183–205, 2005.

[20] B. Marthi, S. Russell, and J. Wolfe. Angelic semantics for high-level actions. In *ICAPS*, 2007.

[21] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, H. Muñoz-Avila, J. W. Murdock, D. Wu, and F. Yaman. Applications of SHOP and SHOP2. *IEEE Intell. Syst.*, 20(2):34–41, Mar.-Apr. 2005.

[22] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, Dec. 2003.

[23] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In T. Dean, editor, *IJCAI*, pages 968–973, Aug. 1999.

[24] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. The SHOP planning system. *AI Mag.*, 2001.

[25] F. Sailer, M. Buro, and M. Lanctot. Adversarial planning through strategy simulation. In *IEEE Symp. Comp. Intel. Games*, 2007.

[26] B. Schattenberg. *Hybrid Planning & Scheduling*. PhD thesis, Universität Ulm, Mar. 2009.

[27] R. Sherwood, A. Mishkin, T. Estlin, S. Chien, P. Backes, B. Cooper, S. Maxwell, and G. Rabideau. Autonomously generating operations sequences for a mars rover using artifical intelligence-based planning. In *IROS*, Oct. 2001.

[28] V. Shivashankar, U. Kuter, and D. Nau. Hierarchical goal network planning: Initial results. Technical Report CS-TR-4983, Univ. of Maryland, May 2011.

[29] S. J. J. Smith, D. S. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[30] A. Tate, B. Drabble, and R. Kirby. *O-Plan2: An Architecture for Command, Planning and Control*. 1994.

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

[32] D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artif. Intell.*, 22(3):269–301, Apr. 1984.

[33] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA, 1988.