2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning

Hierarchical Goal Networks and Goal-Driven Autonomy: Going where AI Planning Meets Goal Reasoning

Vikas Shivashankar SVIKAS@CS.UMD.EDU
Department of Computer Science, University of Maryland, College Park, MD 20742

Ron Alford RONWALF@CS.UMD.EDU
Department of Computer Science, University of Maryland, College Park, MD 20742

Ugur Kuter UKUTER@SIFT.NET
SIFT, LLC, 211 North 1st Street, Suite 300, Minneapolis, MN 55401-2078

Dana Nau NAU@CS.UMD.EDU
Department of Computer Science and Institute for Systems Research, University of Maryland, College Park, MD 20742

Abstract

Planning systems are typically told what goals to pursue and cannot modify them. Some methods (e.g., for contingency planning, dynamic replanning) can respond to execution failures, but usually ignore opportunities and do not reason about the goals themselves. Goal-Driven Autonomy (GDA) relaxes some common assumptions of classical planning (e.g., static environments, fixed goals, no unpredictable exogenous events). This paper describes our Hierarchical Goal Network formalism and algorithms that we have been working on for some time now and discusses how this particular planning formalism relates to GDA and may help address some of GDA's challenges.

1. Motivation

In the real world, intelligent agents typically have to operate in environments that are open-world, partially observable and dynamic. Therefore, pure offline planning or even online plan-repair isn't sufficient to handle unanticipated situations in such environments: external events often necessitate postponing or even abandoning your current goals in order to pursue newer goals of higher priority. *Goal-driven autonomy* (GDA) (Molineaux, Klenk, & Aha, 2010) is a conceptual model of such a goal reasoning process.

As one can imagine, enabling agents to operate as described above requires advanced planning capabilities, in particular capabilities that are not well supported by the current state of the art automated planning formalisms and algorithms. In particular, most existing planning approaches assume *inactive* goals; i.e., goals for planning problems are specified at the start of the planning process and never change during the course of that process. Reasoning about *active* goals and

planning for them is a core foundation for GDA, and we believe, there should be more research on integrating AI planning techniques with active goal reasoning.

In this paper, we will firstly pin down some key capabilities for active goal reasoning in planning that would be useful in the GDA architecture. We will then describe our ongoing research on a knowledge-based planning formalism called **H**ierarchical **G**oal **N**etwork (HGN) planning that makes first steps towards achieving some of these capabilities. HGN planning already supports some of the features desirable from a GDA standpoint such as:

- **Specification of complex domain-specific knowledge as HGN methods** These are similar to methods in Hierarchical Task Network (HTN) planning (Erol, Hendler, & Nau, 1994), but decompose goals into subgoals (instead of tasks into subtasks). When multiple methods can be used to achieve the given goal, we can use domain-independent heuristics to automatically detect the most promising methods.

- **Planning with Incomplete Models.** Knowledge-based planning formalisms are often too dependent on the input domain-specific knowledge; in particular, any missing/incorrect knowledge can break the planner. HGN planning on the other hand allows specification of arbitrary amounts of planning knowledge: it uses the given knowledge whenever applicable and falls back to domain-independent planning techniques including landmark reasoning (Richter & Westphal, 2010) and action chaining to fill in the gaps.

- **Managing Agenda of Goals.** In the GDA model, managing the agenda of goals and deciding which goals to achieve next is outside the purview of the planner. However, generating plans for each candidate goal and comparing the results is often the best way to decide which goal should be pursued next. The core data structure in HGN planning is a *goal network*, which captures precisely this functionality.

HGNs also have a potential to help achieve more advanced planning functionalities including:

- **Goal-based Plan Repair.** In dynamic environments, it is hard for the domain author to anticipate and provide knowledge for all the various scenarios that the agent could face. Since HGN planning is robust to incomplete and incorrect domain models, it is much better equipped to plan in unanticipated situations for which the user has not provided any domain knowledge.

- **Temporal and Cost-Aware Planning.** Since HGN planning uses goals, it is easier to adapt techniques from domain-independent planning literature related to cost-aware planning and temporal planning than in HTN planning.

Planning algorithms in the GDA model ought to be able to continuously assess their current situation, recognize and reason about both serendipitous and harmful events that might change the world around them exogenously, adapt their goals in accordance with these events, and finally generate plans to achieve these goals. HGNs provide a formal foundation for developing algorithms to provide these capabilities.

The remainder of this paper is structured as follows. Section 2 provides some background on GDA. Section 3 presents the HGN planning formalism (Shivashankar et al., 2012). Section 4 then

reviews **G**oal **D**ecomposition **P**lanner (GDP), the algorithm we proposed to solve HGN planning problems in (Shivashankar et al., 2012). Section 5 then presents the **Go**al **De**composition with **L**andmarks (GoDeL) algorithm, an extension of GDP that can work with partial domain-specific knowledge which we proposed in (Shivashankar et al., 2013). Section 6 discusses the potential role of HGNs in providing planning capabilities within GDA. Finally, we conclude in Section 7.

2. GDA Preliminaries

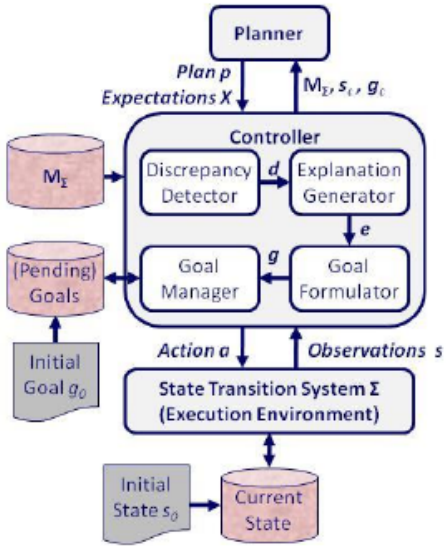

Figure 1. Conceptual Model of GDA (Molineaux, Klenk, & Aha, 2010)

Figure 1 shows a schematic representation of the GDA model. It consists of three components:

- **Planner** This module takes in the current planning problem $P = (M_\Sigma, s_c, g_c)$, where M_Σ is the current model of the environment and s_c, g_c respectively are the current state and goal, and computes a plan p for the agent to execute. It also returns a sequence of expectations $X_c = [x_c, \ldots, x_{c+n}]$ which models constraints on the states $[s_c, \ldots, s_{c+n}]$ that the planner expects the agent to pass through when executing p.
- **Controller** This takes the plan p and sends the actions one by one to the environment Σ and processes the resulting state observations as follows:
  - *Discrepancy Detector* compares actual state observations with its expectations and generates a set of discrepancies D,
  - *Explanation Generator* hypothesizes one or more explanations E for D,
  - *Goal Formulator* generates zero or more goals G in response to E,

  - *Goal Manager* updates G_{pend}, the set of pending goals, with G and optionally changes the set of active goals to be given to the planner.

- **Environment** This takes an action and executes it in the current state and returns observations from the resulting state.

From this model, we can infer some key capabilities that the Planner module has to support:

- **Compatibility with Goals** Since problems in GDA are modeled as goals that need to be achieved, planners need to be able to solve goal achievement problems. In addition, planners should also support complex goal representations such as open-world quantified goals (Talamadupula et al., 2010) and maintenance-style goals such as in Linear Temporal Logic (LTL).

- **Plan Generation** Given a planning problem, the planner should be able to compute plans efficiently.

- **Online Plan Repair** In case exogenous events break the current plan or necessitate change in goals, the planner should be able to repair the current plan.

- **Online Plan Optimization** The planner should continuously optimize the current plan during execution as more time is given. Additionally, It should be able to take advantage of serendipitous events to improve the quality of the plan.

- **Goal Management** In the original GDA model, the goal management is done outside of the planner. However, it is often hard to decide which goal to plan for next among multiple alternatives without actually planning for these goals and comparing the plan qualities. Therefore, we believe that managing the set of goals and deciding which goal to pursue next should also be done by the planner.

2.1 Evaluating Suitability of Domain-Independent Planning for GDA

Domain-Independent planning (Ghallab, Nau, & Traverso, 2004) algorithms in principle fits very well into the GDA model since they work with goals. However, since they cannot exploit any domain-specific knowledge outside of the action models, the performance of these planners rarely scales in complex, real-world domains. Moreover, most domain-independent planners follow assumptions from *classical planning* which restrict goals to conjunctive boolean formulas.

There exists some work on augmenting domain-independent planners with plan repair capabilities (Fox et al., 2006; Hammond, 1990) which perform better than simple replanning strategies. There also exist anytime planning algorithms such as LAMA (Richter & Westphal, 2010) which compute initial solutions and then proceed to optimize these solutions if given additional time. However, similar scalability problems exist as with the plan generation algorithms.

2.2 Evaluating Suitability of Hierarchical Task Network Planning for GDA

HTN planning (Erol, Hendler, & Nau, 1994; Nau et al., 2003), in contrast to domain-independent planning, models planning problems as accomplishment of complex tasks. Another difference

from domain-independent planning is that HTN planning takes as input additional domain-specific knowledge in the form of *HTN methods* which provide ways to decompose tasks into simpler subtasks. HTN planning algorithms thus can be finely tuned to achieve much better performance, especially in complex domains. However, HTN planning algorithms are extremely sensitive to the correctness and completeness of the input method set (Shivashankar et al., 2012): missing/incorrect methods can result in the algorithm not finding valid solutions[1].

Moreover, while there were a number of attempts to combine HTN planning with domain-independent planning (Kambhampati, Mali, & Srivastava, 1998; Mccluskey, 2000; Biundo & Schattenberg, 2001; Alford, Kuter, & Nau, 2009; Gerevini et al., 2008) to work with incomplete HTN method sets, the lack of correspondence between tasks and goals has interfered with these efforts necessitating several *ad hoc* modifications and restrictions.

The disconnect between tasks and goals has also interfered with efforts to augment HTN planning with plan repair capabilities (Ayan et al., 2007; Warfield et al., 2007). (Hawes, 2001; Hawes, 2002) provide HTN planning algorithms that work in an anytime manner. These algorithms however, instead of generating a concrete solution and then iteratively improving it in the remaining time, return a plan at varying levels of abstraction based on when the planner is interrupted; this limits its usefulness since applications typically require a concrete plan that is executable.

3. Hierarchical Goal Networks

As Sections 2.1 and 2.2 indicate, domain-independent planning and HTN planning formalisms have different (and in fact, complementary) advantages and disadvantages. Therefore, we attempted to come up with a hybrid formalism that is hierarchical, but is based on goals rather than tasks. This idea is not new; SIPE (Wilkins, 1984) and PRS (Georgeff & Lansky, 1987; Ingrand & Georgeff, 1992) both use goal decomposition techniques in order to build planning systems. Instead, our contributions lie in (1) building a formal theory and analyzing properites of a hierarchical goal-based planning framework, and (2) exploiting connections between HGNs and techniques in domain-independent planning such as state-space heuristics and landmark reasoning. Below we formalize HGN planning.

Classical planning. Following Ghallab, Nau and Traverso (2004), 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 $o = (\text{head}(o), \text{precond}(o), \text{effects}(o))$, where $\text{precond}(o)$ and $\text{effects}(o)$ 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 $\text{precond}(o)$ and $\text{effects}(o)$).

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 $\text{effects}(a)$. 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 we let $\gamma(s, \pi)$ be the state produced by executing the entire plan.

1. For example, SHOP (Nau et al., 2001) was disqualified from the 2000 International Planning Competition because of an error made by SHOP's authors when they wrote their HTN version of one of the planning domains.

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 2. HGN methods for transporting a package to its goal location in the Logistics domain.

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 $\text{precond}(m)$ like those of a planning operator, and a sequence of subgoals $\text{subgoals}(m) = \langle g_1, \ldots, g_k \rangle$, 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 $\text{subgoals}(m)$ is nonempty; otherwise $\text{post}(m) = \text{precond}(m)$. See Figure 2 for an example set of HGN methods for the Logistics domain.

An action a (or grounded method m) is *relevant* for a goal formula g if $\text{effects}(a)$ (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.

A *goal network* is a way to represent the objective of satisfying a partially ordered sequence 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.

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

Definition 1. *The set of* solutions *for P is defined as follows:*

Case 1. *If T is empty, the empty plan is a solution for P.*

Case 2. *Let t be a node in T that has no predecessors. If $s_0 \models g_t$, then any solution for $P' = (D, s_0, (T', \prec'))$ is also a solution for P, where $T' = T - \{t\}$, and $\prec'$ is the restriction of $\prec$ to T'.*

Case 3. *Let a be any action that is relevant for g_t and executable in s_0. Let π be any solution to the HGN planning problem $(D, \gamma(s_0, a), (T', \prec'))$. Then $a \circ \pi$ is a solution to P.*

Case 4. *Let m be a method instance that is applicable to s_0 and relevant for g_t 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 $(D, \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_k)), (T', \prec'))$. 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.

4. Goal Decomposition Planner

Algorithm 1 is GDP, our HGN planning algorithm. It works as follows:

In Line 3, if gn is empty then the goal has been achieved, so GDP returns π. Otherwise, GDP selects a goal g in gn without any predecessors (Line 4). If g is already satisfied, GDP removes g from gn and calls itself recursively on the resulting goal network.

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.

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 gn.

Algorithm 1: A high-level description of GDP. Initially, D is an HGN planning domain, s is the initial state, gn is the goal network, and π is $\langle\rangle$, the empty plan.

Procedure GDP(D, s, gn, π)
begin
  if gn is empty **then return** π
  $g \leftarrow$ goal formula in gn with no predecessors
  if $s \models g$ **then**
    remove g from gn and **return** GDP(D, s, gn, π)
  $U \leftarrow \{$actions and method instances that are relevant for g and applicable to $s\}$
  if $U = \emptyset$ **then return** failure
  nondeterministically choose $u \in U$
  if u is an action **then** append u to π and set $s \leftarrow \gamma(s, u)$
  else insert subgoals(u) as predecessors of g in gn
  return GDP(D, s, gn, π)
end

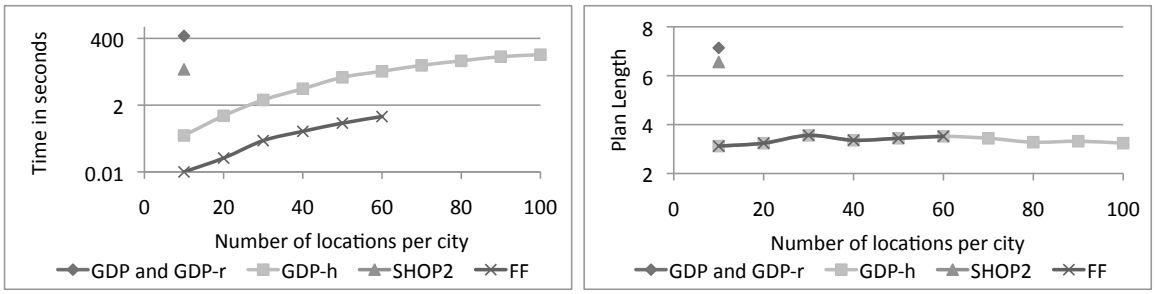

Figure 3. Average running times (in logscale) and plan lengths in the 3-City Routing domain, as a function of the number of locations per city. Each data point is an average of 25 randomly generated problems. FF couldn't solve problems involving more than 60 locations while GDP and SHOP2 could not solve problems with more than 10 locations. SHOP2-r could not solve a single data point, because of which it was excluded from the figure.

Experimental Results. To evaluate the performance of GDP, we compared it with the state-of-the-art HTN planner SHOP2 (Nau et al., 2003) and the domain-independent planner FF (Hoffmann & Nebel, 2001) across five domains. GDP-h represents GDP enhanced with a variant of the relaxed planning graph heuristic used in FF. GDP-r and SHOP2-r are identical to GDP and SHOP2 respectively, but with the input method sets randomly reordered. We did this to investigate the effect of methods being provided in the wrong order on the planner's performance. Here we present results from only one domain: *One-City Routing*; the complete experimental study is detailed in (Shivashankar et al., 2012).

We constructed the *3-City Routing* domain in order to examine the performance of the planners in a domain with a weak domain model. 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 3 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.

GDP-r performed at par with GDP since there was only one method, and hence reordering did not affect it. SHOP2-r, on the other hand, could not solve a single problem, thus illustrating the high sensitivity to the correctness of not only the methods, but also the *order* in which they are provided. We believe that GDP is also similarly sensitive to the method order; on the other hand, GDP-h, due its domain-independent heuristic-based method ordering technique, is agnostic to the method ordering provided by the domain author.

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

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 4 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. This is the case because the HGN task and method semantics obviates the need for a plethora of extra methods and bookkeeping operations needed in HTN domain models (see (Shivashankar et al., 2012) for a more detailed explanation).

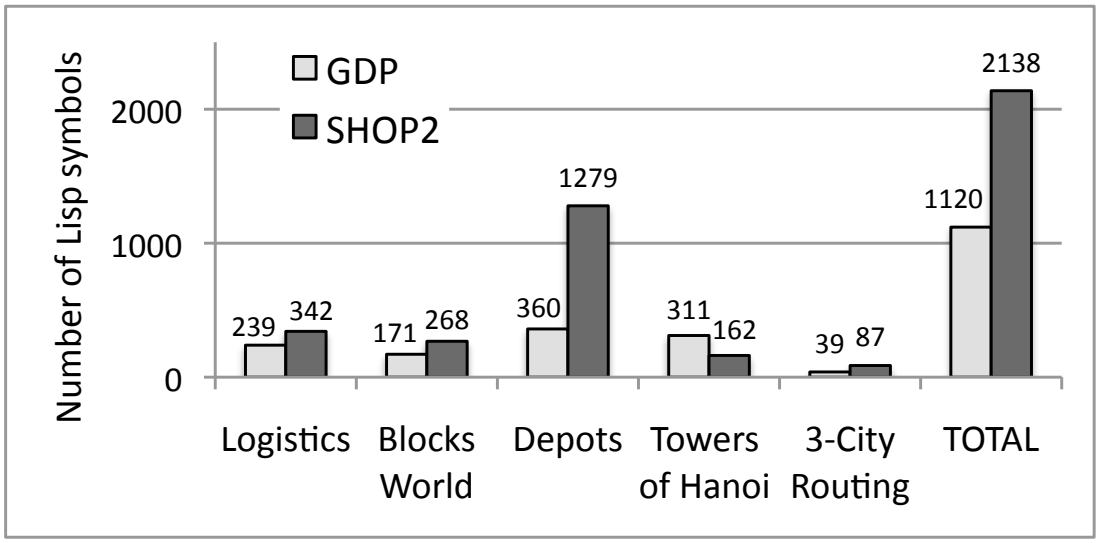

Figure 4. Sizes (number of Lisp symbols) of the GDP and SHOP2 domain models.

5. The GoDeL Planning Formalism and Algorithm

GDP is similar to SHOP/SHOP2 in that it is sound and complete only with respect to the methods provided to it. It therefore provides no completeness guarantees if the HGN methods are incomplete. GoDeL is an extension of GDP that does provide completeness guarantees even if given incomplete or empty method sets. It accomplishes this by interleaving method decomposition with subgoal inference using landmarks and action chaining.

Explanation of pseudocode. Algorithm 2 is the GoDeL planning algorithm. It takes as input a planning problem $P = (D, s, G)$, a set of methods M, and the partial plan π generated so far.

Lines 3 – 6 specify the base cases of GoDeL. If these are not satisfied, the algorithm nondeterministically chooses a goal g with no predecessors and generates $\mathcal{U}$, all method and operator instances applicable in s and relevant to g. It then nondeterministically chooses a $u \in \mathcal{U}$ to progress the search (Lines 7 – 9). If u is an action, the state is progressed to $\gamma(s, u)$ (Line 10). Else, the subgoals of u are added to G, adding edges to preserve the total order imposed on subgoals(u) (Line 12). In either case, GoDeL is invoked recursively on the new planning problem (Lines 12 and 14).

If GoDeL fails to find a plan in the previous step, it then attempts to infer a partially ordered set of subgoals lm that are to be achieved enroute to achieving g using the Infer-Subgoals procedure (Line 15). The subgoals are added to G, adding edges to preserve the partial order in lm. The algorithm is then recursively invoked on the new planning problem (Line 17). If this call also returns failure, then the algorithm falls back to action chaining (Lines 19 – 22), returning failure if no actions are applicable in s.

Below, we shall motivate and describe the Infer-Subgoals procedure in greater detail.

Subgoal inference. Sometimes, the planner may have methods that tell how to solve some subproblems, but not the top-level problem. For instance, the first method in Figure 2 would not

Algorithm 2: A nondeterministic version of GoDeL. Initially, (D, s, G) is the planning problem, M is a set of methods, and π is $\langle\rangle$, the empty plan.

```
1  Procedure GoDeL(D, s, G, M, π)
2  begin
3      if G is empty then return π
4      nondeterministically choose a goal formula g in G without any predecessors
5      if s ⊨ g then
6          return GoDeL(D, s, G − {g}, M, π)
7      U ← {operator and method instances applicable to s and relevant to g}
8      while U is not empty do
9          nondeterministically remove a u from U
10         if u is an action then
11             res1 ← GoDeL(D, γ(s, u), G, M, π ∘ u)
12         else
13             res1 ← GoDeL(D, s, subgoals(u) ∘ G, M, π)
14         if res1 ≠ failure then return res1
15     lm ← Infer-Subgoals(D, s, g)
16     if lm ≠ ∅ then
17         res2 ← GoDeL(D, s, lm ∘ G, M, π)
18         if res2 ≠ failure then return res2
19     A ← {operator instances applicable to s}
20     if A = ∅ then return failure
21     nondeterministically choose an a ∈ A
22     return GoDeL(D, γ(s, a), G, M, π ∘ a)
23 end
```

be applicable to problems involving transporting packages across cities, but it *is* applicable to the subproblems of moving the package between the start and goal locations and the corresponding airports. A natural question that then arises is the following: *How can we automatically infer these subproblems for which the given methods are relevant?*

To answer the above question, we use landmarks. A *landmark* for a planning problem P (Hoffmann, Porteous, & Sebastia, 2004; Richter & Westphal, 2010) is a fact that is true at some point in every plan that solves P. A *landmark graph* is a directed graph whose nodes are *landmarks* and edges denote orderings between these landmarks. Therefore, if there is an edge between two landmarks l_i and l_j, this implies that l_i is true before l_j in every solution to P.

Therefore, a landmark for a planning problem P can be thought of as a subgoal that every solution to P must satisfy at some point. We can, as a result, use any landmark generation algorithm (for example, (Hoffmann, Porteous, & Sebastia, 2004; Richter & Westphal, 2010)) to automatically infer subgoals (and orderings between them) for which the given methods are relevant.

Algorithm 3: Procedure to deduce possible subgoals for GoDeL to use. It takes as input a planning problem $P = (D, s, g)$, and outputs a DAG of subgoals. It uses LMGEN, a landmark generation algorithm that takes P and generates a DAG (V, E) of landmarks for it.

1 **Procedure** Infer-Subgoals (D, s, g)
2 **begin**
3 $(V, E) \leftarrow \mathsf{LMGEN}(D, s, g)$
4 $L \leftarrow \{v \in V : s \not\models \mathsf{fact}(v), g \not\models \mathsf{fact}(v)$ and $\exists$ a method $m \in D$ s.t. goal(m) is relevant to $\mathsf{fact}(v)\}$
5 **if** $L = \emptyset$ **then return** $\emptyset$
6 $E_L \leftarrow \{(u, v) : u, v \in L$ and v is reachable from u in $(V, E)\}$
7 **return** (L, E_L)
8 **end**

Algorithm 3 is Infer-Subgoals, the subgoal inference procedure. It uses a landmark generation procedure LMGEN (such as ones in (Richter & Westphal, 2010; Hoffmann, Porteous, & Sebastia, 2004)) that takes as input a classical planning problem P and generates a DAG of landmarks.

Infer-Subgoals begins by computing LM_graph, the landmark graph for the input problem P. It then computes L, the set of nodes in LM_graph that have relevant methods (Line 4). It does not consider trivial landmarks such as literals true in the state s or part of the goal g. E_L is the set of all edges between landmarks $l_i, i_j \in L$ such that there exists a path from l_i to l_j in LM_graph (Line 6). Infer-Subgoals returns the resulting partial order lm $= (L, E_L)$.

Experimental Results. The main purpose of this experimental study was to investigate the performance of GoDeL with varying amounts of domain knowledge. In particular, we hypothesized that GoDeL would (1) perform at par with state-of-the-art domain-independent planners when given low/no knowledge, (2) improve performance as more knowledge is given, and finally (3) perform at par with state-of-the-art hierarchical planners when given complete knowledge.

To verify this hypothesis, we compared (1) GDP-h, the heuristic enhanced version of GDP from Section 4, (2) the state of the art domain-independent planner LAMA (Richter & Westphal, 2010) and (3) GoDeL run with three different amounts of domain knowledge: GoDeL with complete knowledge (GoDeL-C), moderate knowledge (GoDeL-M), and low knowledge (GoDeL-L). We compared these planners across three domains; we present the results only for Depots. The full experimental study is reported in (Shivashankar et al., 2013).

The experimental results on the Depots domain are consistent with our hypotheses. As shown in Figure 5, GoDeL-C has the best running times in Depots, and is the only planner to solve all of the problems. Even GoDeL-M significantly outperformed GDP-h, which had access to the full method set. GoDeL-L, which was given only one Blocks-World method, solved significantly fewer problems. LAMA solved the fewest problems, having not solved any problems containing 24 blocks or more. With respect to plan lengths, GoDeL-C, GoDeL-M and GDP-h produced plans of similar lengths for the problems they could solve. GoDeL-L and LAMA however generate significantly longer plans for the problems they could solve.

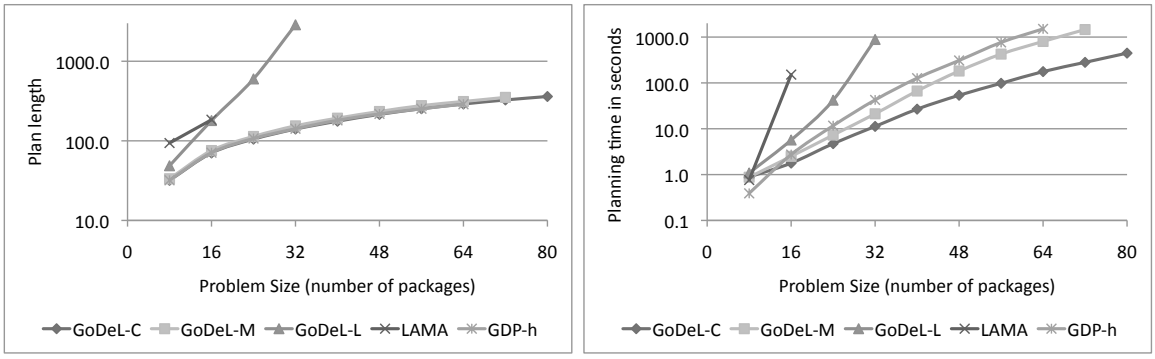

Figure 5. Average plan lengths and running times (both in logscale) in the Depots domain, as a function of the number of packages to be delivered. Each data point is an average of 25 randomly generated problems. GoDeL-M could not solve 80-package problems; GDP-h, GoDeL-L and LAMA could not solve problems of size > 64, 32 and 16 respectively.

6. Role of HGNs in GDA

We presented the HGN planning formalism in Section 3 and two HGN planning algorithms in Sections 4 and 5. However, the question that still remains to be answered is: *Why use HGNs in GDA?* In this section, we address this by discussing the main features of HGN planning algorithms and how they help address some of the planning challenges posed in GDA. We shall also discuss how HGN planning is in a unique position to solve some of the more advanced planning problems in goal reasoning.

6.1 Modeling Domain-Specific Knowledge using HGN Methods

One of the strengths of using HTNs is that one can model complicated pieces of knowledge as HTN methods, thus not only speeding up search but also enabling fine-tuning of the types of solutions you want the planner to return. HGNs also provide the same amount of control, but with the added benefit of retaining the semantics of goals in the methods (see (Shivashankar et al., 2012) for a discussion about incompatibilities between task and goal semantics). We show in (Shivashankar et al., 2012) that in fact HGNs are equal in expressivity to total-order HTN planning, thus indicating that one will not lose any expressivity when moving from HGNs to HTNs.

6.2 Managing Agenda of Goals using HGNs

In the GDA model, managing the agenda of goals and deciding which goals to achieve next is outside the purview of the planner. As we mentioned previously, sometimes generating plans for the candidate goals and comparing them is often the best way to decide which goal should be pursued next. Moreover, certain goal orderings work much better than others; determining a feasible order in which goals should be achieved is a problem the planner can help solve. HGNs provide a framework to do this reasoning: since problems are modeled as $(state, goalnetwork)$ pairs, the agenda of goals can be managed within the planner as a goal network and the planner could use its

heuristics in combination with any other domain-specific goal choice strategy to choose next goals. In particular, the definition of a goal network could be extended to include *utilities* over the goals, which could even change over time (and thus potentially trigger replanning). (Wilson, Molineaux, & Aha, 2013) adopt a similar approach of letting the planner evaluate candidate next goals in a domain-independent fashion in the context of HTNs. They do not, however, factor in implications that achieving a particular goal might have on the (un)solvability of another goal downstream.

6.3 Seamless Use of Incomplete HGN Domain Models

As mentioned previously, one of the main criticisms of HTN planning is the brittleness of the planners due to the over-reliance on the correctness and completeness of HTN methods. GoDeL on the other hand can work with any arbitrary sets of HGN methods. This is possible mainly due to goals in HGN methods which makes it immediately compatible with domain-independent planning. Therefore, domain engineers can provide domain knowledge only for the most complicated parts of the domain and rely on GoDeL to figure out the rest, thus easing the burden of domain authoring. This becomes even more crucial in complex real-world domains where its unrealistic to assume that a complete suite of domain knowledge can be made available to the planner.

6.4 Plan Repair in Dynamic Environments

The domain authoring issue is further exacerbated in dynamic environments since other agents, exogenous events, and erroneous sensor and action models can bring about unanticipated situations which the domain author cannot anticipate offline. Therefore, it is even more critical in such cases that the planner has a fallback mechanism in case the provided knowledge is insufficient. Therefore, GoDeL augmented with plan repair capabilities will be perfectly suited for such use-cases.

6.5 Temporal Planning and Cost-Aware Planning

One of the advantages in moving from task networks to goal networks is that we can adapt existing planning heuristics from the domain-independent planning literature to work with HGNs. We provided a proof-of-concept of this by augmenting GDP, our first HGN planner, with a variant of the *relaxed planning graph heuristic* used in the FastForward planning system (Hoffmann & Nebel, 2001). This helps the planner in cases when multiple methods are applicable at a certain point in the search and the planner needs a way to decide which methods to try out first.

Along the same lines, we can adapt *admissible* heuristics (Karpas & Domshlak, 2009) that, if used with an algorithm like A*, can assure cost-optimal plans. We can also aim for a principled compromise via anytime planning techniques (Richter & Westphal, 2010), where we do not assure cost optimality, but instead try to improve the current best solution by exploring promising parts of the search space yet unexplored in an anytime fashion and asymptotically reach the optimal solution.

We can also augment the HGN formalism to reason about temporal aspects of planning via durative actions and deadlines on goals. These extensions, we think, will be very useful from a GDA standpoint since agents often have to reason about resource constraints, deadlines, and action costs; therefore it will be helpful to have planners that can actively optimize with respect to these metrics in a scalable manner.

7. Final Remarks and Future Work

We have described our planning formalism, called *Hierarchical Goal Networks (HGNs)*, and our planning algorithm based on this formalism. We argued that HGNs are particularly relevant and useful for GDA because they provide flexibility and reasoning capabilities in goal reasoning, during both planning and execution.

We're currently developing a generalization of HGNs for temporal goal reasoning. This work involves extending the HGN formalism described here with durative tasks and deadlines on goal achievement. The GoDeL algorithm for HGN planning must be generalized for reasoning with such temporal constructs. One of the challenges in this work is to model, correctly, the temporal semantics between planning and execution that will yield sound behavior in the generated plans.

Furthermore, to facilitate GDA, we want to extend HGN planning to include ways of reasoning and planning with goals under execution failures. Execution failures typically stem from the discrepancies from the planning and goal model that a system has and the ground-truth discovered during execution. We're interested in investigating how a planner can help reorder goal preferences during execution when such discrepancies occur, and repair the existing plans accordingly.

Finally, humans can reason about goals in very abstract levels and semantically, whereas automated planning and learning algorithms are computationally effective but has to stay in the realm of logical formulations. This comparison is perhaps similar to the comparisons between humans and AI in image processing tasks. For example, during their planning and acting, humans can identify a goal that may not be on their initial agenda but is a "low-hanging fruit." We're currently investigating how to model this concept in an AI planning and execution algorithm.

Acknowledgements.. This work was supported in part by ARO grant W911NF1210471 and ONR grants N000141210430 and N000141310597. The information in this paper does not necessarily reflect the position or policy of the funders; no official endorsement should be inferred.

References

Alford, R., Kuter, U., & Nau, D. S. (2009). Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. *IJCAI*.

Ayan, N. F., Kuter, U., Yaman, F., & Goldman, R. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution.*.

Biundo, S., & Schattenberg, B. (2001). From abstract crisis to concrete relief–a preliminary report on combining state abstraction and HTN planning. *Proc. of the 6th European Conference on Planning* (pp. 157–168).

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *AAAI*.

Fox, M., Gerevini, A., Long, D., & Serina, I. (2006). Plan stability: Replanning versus plan repair. *ICAPS* (pp. 212–221).

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. *AAAI* (pp. 677–682).

Gerevini, A., Kuter, U., Nau, D. S., Saetti, A., & Waisbrot, N. (2008). Combining domain-independent planning and HTN planning. *ECAI* (pp. 573–577).

Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated planning: Theory and practice*.

Hammond, K. J. (1990). Explaining and repairing plans that fails. *Artif. Intell.*, *45*, 173–228.

Hawes, N. (2001). Anytime planning for agent behaviour. *Twelth Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 157–166).

Hawes, N. (2002). An anytime planning agent for computer game worlds. *Proceedings of the Workshop on Agents in Computer Games at The 3rd International Conference on Computers and Games*, 1–14.

Hoffmann, J., & Nebel, B. (2001). The FF planning system. *JAIR*, *14*, 253–302.

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *JAIR*, *22*, 215–278.

Ingrand, F., & Georgeff, M. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *6*, 33–44.

Kambhampati, S., Mali, A., & Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. *AAAI* (pp. 882–888).

Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. *IJCAI* (pp. 1728–1733).

Mccluskey, T. L. (2000). Object transition sequences: A new form of abstraction for HTN planners. *AIPS* (pp. 216–225).

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-driven autonomy in a navy strategy simulation. *AAAI*. AAAI Press.

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

Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (2001). The SHOP planning system. *AI Mag.*

Richter, S., & 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. S., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. *AAMAS* (pp. 981–988).

Shivashankar, V., Kuter, U., Nau, D. S., & Alford, R. (2013). The GoDeL planning system: A more perfect union of domain-independent planning and hierarchical planning. *IJCAI*.

Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P. W., & Scheutz, M. (2010). Planning for human-robot teaming in open worlds. *ACM TIST*, *1*, 14.

Warfield, I., Hogg, C., Lee-Urban, S., & Muñoz-Avila, H. (2007). Adaptation of hierarchical task network plans. *FLAIRS* (pp. 429–434). AAAI Press.

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

Wilson, M. A., Molineaux, M., & Aha, D. W. (2013). Domain-independent heuristics for goal formulation. *FLAIRS Conference*. AAAI Press.