# The GoDeL Planning System: A More Perfect Union of Domain-Independent and Hierarchical Planning

**Vikas Shivashankar**[1] **Ron Alford**[1] **Ugur Kuter**[2] **Dana Nau**[1]
[1]Department of Computer Science, Institute of Systems Research, and
Institute for Advanced Computer Studies, University of Maryland at College Park
[2]Smart Information Flow Technologies LLC, Minneapolis
svikas@cs.umd.edu ronwalf@cs.umd.edu ukuter@sift.net nau@cs.umd.edu

## Abstract

One drawback of Hierarchical Task Network (HTN) planning is the difficulty of providing complete *domain knowledge*, i.e., a complete and correct set of HTN methods for every task. To provide a principled way to overcome this difficulty, we define a simple formalism that extends classical planning to include problem decomposition using methods, and a planning algorithm based on this formalism.

In our formalism, the methods specify ways to achieve goals (rather than tasks as in conventional HTN planning), and goals may be achieved even when no methods are available. Our planning algorithm, GoDeL (Goal Decomposition with Landmarks), is sound and complete irrespective of whether the domain knowledge (i.e., the set of methods given to the planner) is complete.

By comparing GoDeL's performance with varying amounts of domain knowledge across three benchmark planning domains, we show experimentally that (1) GoDeL works correctly with partial planning knowledge, (2) GoDeL's performance improves as more planning knowledge is given, and (3) when given full domain knowledge, GoDeL matches the performance of a state-of-the-art hierarchical planner.

## 1 Introduction

Although HTN planning has generally been acknowledged to be more useful for practical applications than other kinds of AI planning, its applicability is limited by the difficulty of providing a complete set of *domain knowledge*, i.e., a set of HTN methods capable of generating plans for every possible planning problem in a given domain. The purpose of our work is to provide a way of overcoming this difficulty. Our contributions include the following:

- We describe a simple extension to classical planning to include hierarchical decomposition. Like HTN planning, our formalism involves hierarchical decomposition using methods; but in our formalism the methods are for achieving goals rather than tasks, and goals may be achieved regardless of whether there are methods for achieving them.
- We present the GoDeL (**Go**al **De**composition with **L**andmarks) planning algorithm, which incorporates techniques for method decomposition, domain-independent heuristics, and planning landmarks. We present several theorems, including soundness and completeness irrespective of whether GoDeL is given a complete set of methods.
- We present experimental results for an implementation of GoDeL (for landmark generation in GoDeL, we used code from the well-known LAMA planner [Richter and Westphal, 2010]). The results show that GoDeL works correctly with partial planning knowledge (i.e., an incomplete set of methods), and its performance improves as more planning knowledge is given. When given complete domain knowledge, GoDeL performs as well or better than a state-of-the-art hierarchical planner, GDP-$h$ [Shivashankar *et al.*, 2012].

## 2 Preliminaries

Our definitions are based on those in [Ghallab *et al.*, 2004, Chap. 2], but generalized to include goal networks.

We define a *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 conjuncts 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 $\gamma(s, \pi)$ is the state produced by executing the entire plan. If $\pi$ and $\pi'$ are plans or actions, then their concatenation is $\pi \circ \pi'$.

A *goal network* is a way to represent the objective of satisfying a partially ordered sequence of goals (hence one can think of it as a particular kind of temporally extended goal). 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 $T$.

A *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.** *If action $a$ is applicable in $s_0$ and $\pi$ is a solution for $P'' = (D, \gamma(s_0, a), gn)$, then $a \circ \pi$ is a solution for $P$.*

In cases where $gn$ contains just a single node, the definitions of $P$ and its solutions reduce to the conventional definitions of a classical planning problem and its solutions. In such cases we will say that $P$ is *classical.*

A *method* $m$ has a head head$(m)$ and preconditions precond$(m)$ like those of a planning operator, and a sequence of *subgoals*, subgoals$(m) = \langle g_1, \ldots, g_k \rangle$, where each subgoal $g_i$ is a conjunct of literals from $L$. We define the *postcondition* of $m$ to be post$(m) = g_k$ if subgoals$(m)$ is nonempty; otherwise post$(m)$ = precond$(m)$.

**Example 2.** In the Logistics domain, a method to move a package $p$ from locations $l$ to $l'$ within a city might look like:

- head: move-package-within-city$(p, l, l')$
- precond: package-at$(p, l)$, truck$(t)$, same-city $(l, l')$
- subgoals: $\langle$truck-at$(t, l)$, in-truck$(p, t)$, truck-at$(t, l')$, package-at$(p, l')\rangle$

Notice that unlike conventional definitions of HTN planning problems (e.g., [Ghallab *et al.*, 2004, Chapter 11]), in which methods are essential to the definition of a solution, our definition of a planning problem and its solutions does not involve methods at all. In our planning algorithm (see the next section), the purpose of methods is to provide guidance for what parts of the search space to examine next. This guidance is based on *relevance*, as defined here:

If $a$ is an action or a ground instance of a method, then $a$ is *relevant* for a goal formula $g$ if effects$(a)$ (if $a$ is an action) or post$(a)$ (if $a$ is a method instance) entails at least one literal in $g$ and does not entail the negation of any literal in $g$.

## 3 Planning Algorithm

The **Go**al **De**composition with **L**andmarks (GoDeL) planning algorithm, when given a problem $(D, s_0, gn)$, works as follows: It chooses a $g$ from $gn$ that has no predecessors and (1) first attempts to decompose $g$ using one of the given methods, (2) if not, then it automatically infers subgoals to insert into $gn$ using landmark-based techniques, and (3) if neither of the above steps work, it falls back to traditional action-chaining.

Algorithm 1 describes the GoDeL planning algorithm. It takes as input a planning problem $P = (D, s, gn)$, a set of methods $M$, and the partial plan $\pi$ generated so far. Lines 2 – 5 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 6 – 9). If $u$ is an action, the state is progressed to $\gamma(s, u)$ (Line 11). If $u$ is a method, the subgoals of $u$ are added to $gn$, adding edges to preserve the total order imposed by subgoals$(u)$ (Line 15). In either case, GoDeL is invoked recursively on the new planning problem.

If GoDeL fails to find a plan in the previous step, it then uses the Infer-Subgoals procedure to infer lm, a network of subgoals that are to be achieved enroute to achieving $g$ (Line 18). The subgoals are added to $gn$, adding edges to preserve the partial order in lm. The algorithm is then recursively invoked on the new planning problem (Line 20). If this call also returns failure, then the algorithm falls back to action chaining (Lines 22 – 25), returning failure if no actions are applicable in $s$.

GoDeL maintains a global map used_methods that keeps track of the set of method instances already used in a given

**Algorithm 1**: A nondeterministic version of GoDeL. Initially, $(D, s, gn)$ is the planning problem, $M$ is a set of methods, and $\pi$ is $\langle\rangle$, the empty plan. used_methods, a global variable, is a mapping from states to methods applied in those states; it is initially set to the empty map. subgoals_inferred is a boolean initially set to false.

```
1  Procedure GoDeL (D, s, gn, M, π)
2  if gn is empty then return π
3  nondeterministically choose a goal formula g in gn
   without any predecessors
4  if s ⊨ g then
5  |  return GoDeL(D, s, gn − {g}, M, π)
6  U ← {operator and method instances applicable to s and
   relevant to g}
7  U ← U − used_methods[s]
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), gn, M, π ∘ u)
12 |  else
13 |  |  add u to used_methods[s] and set
   |  |  subgoals_inferred to false
14 |  |  res1 ← GoDeL(D, s, subgoals(u) ∘ gn, M, π)
15 |  if res1 ≠ failure then return res1
16 if subgoals_inferred ≠ true then
17 |  subgoals_inferred ← true
18 |  lm ← Infer-Subgoals (D, s, g)
19 |  if lm ≠ ∅ then
20 |  |  res2 ← GoDeL (D, s, lm ∘ gn, M, π)
21 |  |  if res2 ≠ failure then return res2
22 A ← {operator instances applicable to s}
23 if A = ∅ then return failure
24 nondeterministically choose an a ∈ A
25 return GoDeL (D, γ(s, a), gn, M, π ∘ a)
```

**Algorithm 2**: Procedure to deduce possible subgoals for GoDeL to use. It takes as input a planning problem $P = (D, s, g)$, and outputs a poset of subgoals. It uses LMGEN, an abstract landmark generation algorithm that takes $P$ and generates a DAG of landmarks for it.

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

state $s$. This is used to prune out used methods from the set of options $\mathcal{U}$ (Line 7) and is updated when a new method is applied (Line 13). Similarly, GoDeL also uses subgoals_inferred, a boolean variable that keeps track of whether the goal network has been modified since the last time the Infer-Subgoals procedure is invoked, ensuring that the latter is invoked only once between changes to the goal network. As we shall see in Section 4, these steps are critical in ensuring GoDeL's completeness.

**Subgoal Inference**. We now describe the subgoal inference technique used in GoDeL. This aspect of GoDeL is motivated by the fact that sometimes the planner may have methods that tell how to solve some subproblems, but not the top-level problem. For instance, the method in Example 2 would not 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 *et al.*, 2004; Richter and 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 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 *et al.*, 2004; Richter and Westphal, 2010]) to automatically infer subgoals (and orderings between them) for which the given methods are relevant.

Algorithm 2 is Infer-Subgoals, the subgoal inference procedure. It uses an abstract landmark generation procedure LMGEN that takes as input a classical planning problem $P$ and generates a DAG of landmarks.

Infer-Subgoals begins by computing the landmark graph $(V, E)$ for the input problem $P$. It then computes $L$, the subset of landmarks in $V$ that have relevant methods (Line 3). 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 $(V, E)$ (Line 5). Infer-Subgoals then returns the resulting network of landmarks $\text{lm} = (L, E_L)$.

## 4 Formal Properties

To show that GoDeL is sound and complete, it is necessary to show that GoDeL deals correctly with some "corner cases" involving methods and landmarks, e.g., that it can detect and recover from cases where the method recursion doesn't terminate. To keep the proof sketches simple, the following theorems all deal with the case where the planning problem $P = (D, s_0, gn)$ is classical, i.e., $gn$ contains one node, hence one goal $g$. It is straightforward to generalize the theorems to arbitrary goal networks.

**Theorem 3 (soundness).** *Let $P = (D, s_0, gn)$ be as described above, and $M$ be a set of methods. If a nondeterministic trace of* GoDeL$(D, s_0, gn, M, \langle\rangle)$ *returns a plan $\pi$, then $\pi$ is a solution for $P$.*

We omit the (simple) proof due to space constraints.

A planning problem $P'$ is *reachable* by GoDeL from $P$ if $P'$ is in the search space produced by invoking GoDeL on $P$, i.e., if one of GoDeL's nondeterministic traces includes $P'$.

**Theorem 4 (search space finiteness).** *If $P = (D, s_0, gn)$ is as described above, $M$ is a set of methods, and* LMGEN *is a sound landmark generation algorithm, then the set of planning problems reachable by* GoDeL *from $P$ is finite.*

*Proof Sketch.* Firstly, note that there are two ways by which goals get added to the goal network, method application and subgoal inference. Let us first consider the former. Note that GoDeL never uses the same method instance twice to decompose a goal network in the same state (see Lines 7 and 13 in Algorithm 1). Since there are finitely many methods and the size of the planning problem is finite, it follows that the size of the largest goal network reachable via method decomposition from $gn$'s goal $g$ in the state $s$ is bounded.

With respect to subgoal inference via landmarks, note that for any classical planning problem $P_c = (D, s, g)$, the set of landmarks $\mathcal{L}$ for $P_c$ is finite. Since LMGEN is sound, the subgoals it infers belongs to this set. Thus, it is easy to show that Infer-Subgoals can be run only a finite number of times, adding at most $|\mathcal{L}|$ nodes to the goal network each time. Therefore, we can place an upper bound on the goal network size in a given state $s$. The bound on the size of the largest goal network reached by GoDeL is thus at most $|S|$ times the previous bound, $S$ being the set of reachable states. Using this bound, we can subsequently bound the number of reachable goal networks. Since $S$ is finite, the number of reachable planning problems is also finite. □

**Theorem 5 (completeness).** *Let $P = (D, s_0, gn)$ be as described above, and $M$ be a set of methods. If $P$ is solvable and* LMGEN *is sound, a nondeterministic trace of* GoDeL$(D, s_0, \langle g\rangle, \langle\rangle)$ *will return a solution $\pi$ for $P$.*

*Proof Sketch.* From Theorem 4 and the fact that GoDeL can always fall back to action-chaining to generate a solution if method decomposition does not work, the theorem follows. □

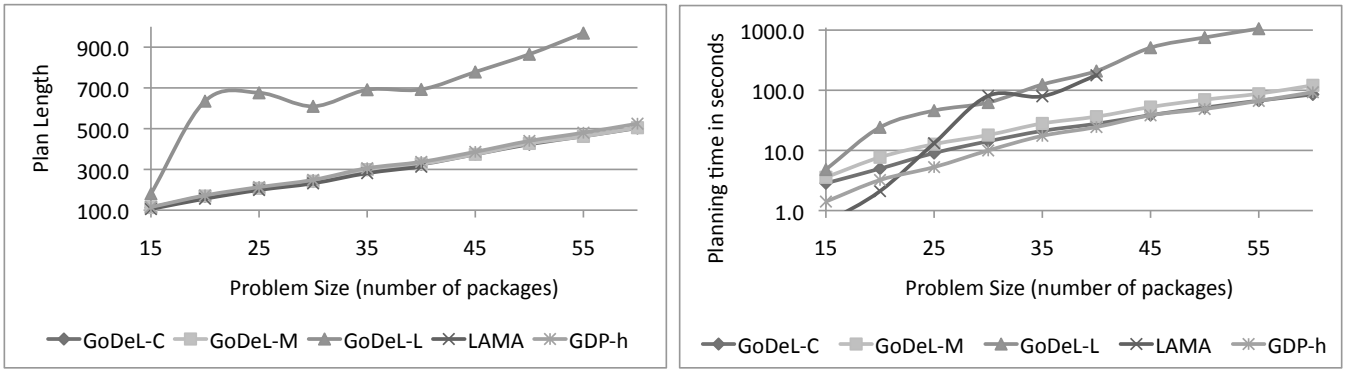


Figure 1: Average plan lengths and running times (in logscale) in the Logistics domain, as a function of the number of packages to be delivered. Each data point is the average across 10 problems. GoDeL-$L$ could not solve 60-package problems while LAMA could not solve problems greater than 40.

## 5 Implementation and Experiments

We implemented GoDeL in C++. The nondeterministic choice among the options in the set $\mathcal{U}$ (Lines 6–9 in Algorithm 1) is implemented using depth-first search. We sort $\mathcal{U}$ using a variant of the GDP heuristic suggested in [Shivashankar *et al.*, 2012]. For generating landmarks in Infer-Subgoals, we use LAMA's landmark generation code [Richter and Westphal, 2010], which generates sound, acyclic landmark graphs. GoDeL's action chaining (line 22) uses depth-first search with the various actions sorted by the FF heuristic [Hoffmann and Nebel, 2001] value.

### 5.1 Experimental Design

The following research questions motivated our experiments:

- *Does* GoDeL*'s performance improve as more planning knowledge is given to it?*
- *Does* GoDeL*'s technique of inferring subgoals help discover useful intermediate goals which the domain knowledge can help solve?*
- *How well does* GoDeL *perform when given a complete set of methods?*

Our benchmark planning domains, and our reasons for choosing them, were as follows:

- Logistics [Veloso, 1992]. Problem decomposition works well for these problems: they decompose neatly into (1) moving packages with the city, (2) moving packages between airports and (3) moving packages across cities.
- Blocks-World [Bacchus, 2001]. Blocks-World problems, unlike Logistics problems, do not decompose neatly. Instead, a good planning strategy needs to select actions to apply on a state-by-state basis [Ghallab *et al.*, 2004, Sect. 4.5].
- Depots [Fox and Long, 2002]. This domain combines aspects of the above two domains, hence is useful for evaluating the performance of planners with partial planning knowledge.

For each of these domains, we wrote three different method sets having complete (C), moderate (M) and low (L) amounts of domain knowledge respectively, as described below. We shall henceforth refer to GoDeL when using these various levels of knowledge as GoDeL-$C$, GoDeL-$M$ and GoDeL-$L$ respectively. We compared these variants of GoDeL against GDP-$h$ [Shivashankar *et al.*, 2012] and LAMA [Richter and Westphal, 2010], which are state-of-the-art hierarchical and domain-independent planners, respectively. We provided GDP-$h$ the same methods given to GoDeL-$C$ in each domain.

All experiments were run on 2GHz machines with 4GB RAM. We set a time limit of 30 minutes per problem, discarding data points not solved within this time.

### 5.2 Experimental Results

**Logistics**. For the Logistics domain, the complete method set required three methods: one each for (1) same-city delivery, (2) different-city delivery and (3) airport-to-airport delivery. We constructed the moderate (M) method set by removing method 2. The low (L) method set consisted of just method 1. We compared the planners across 10 $n$-package problems for each of $n = 15, 20, \ldots, 60$, from the 1998/2000 International Planning Competition distributions.

Figure 1 compares the plan lengths and running times of the various planners in this domain. We see that GoDeL-$C$ and GDP-$h$ produced plans of similar length. GoDeL-$M$ was also able to perform at par with these planners; this was because the subgoal inference algorithm automatically inserted the goals that method 2 (which is missing for GoDeL-$M$) would otherwise have inserted into the goal network. With regard to planning times for these planners, GoDeL-$C$, GoDeL-$M$ and GDP-$h$ all reported similar running times.

GoDeL-$L$, however, did not perform as well with its running times nearly an order of magnitude higher and its plan lengths nearly twice as compared to GoDeL-$C$. This is because it had to solve a larger portion of the problem via action-chaining due to the missing methods. Since this part of GoDeL is relatively simplistic (depth-first search with FF heuristic), GoDeL in a number of cases found it hard to recover from a bad action it chose to apply in the previous step, thus leading to longer plans and running times.

LAMA solved the smaller logistics problems (up to 40-

Table 1: Amount of action chaining as a function of the domain and the amount of domain-specific knowledge provided. The lower the fraction, the larger the fraction of the plan that is generated via method decomposition.

| Domain | GoDeL-$C$ | GoDeL-$M$ | GoDeL-$L$ |
|---|---|---|---|
| Logistics | 0 | 0 | 0.66 |
| Blocks World | 0.09 | 0.48 | 0.59 |
| Depots | 0.08 | 0.59 | 0.75 |

package problems) very quickly, producing plans of roughly the same length as GDP-$h$, GoDeL-$C$ and GoDeL-$M$; but it couldn't solve the larger problems within the time limit.

**Blocks World**. For the Blocks World domain, the complete set of methods included three methods: one each for (1) on$(X, Y)$, (2) clear$(X)$ and (3) on-table$(X)$. The moderate (M) method set consisted of methods 1 and 2 while the low (l) method set consisted of just method 1. In addition, all three method sets used a need-to-move$(X)$ derived predicate that determined whether $X$ was in its final position. We compared the planners across 25 randomly generated $n$-blocks problems for each of $n = 10, 20, \ldots, 100$.

As shown in Figure 2, GoDeL-$C$, GoDeL-$M$ and GDP-$h$ have almost identical planning times. GoDeL-$M$ is able to perform as well as the other two planners because on-table$(X)$ (which the missing method 3 solves) is easily solvable through action chaining. GoDeL-$L$ on the other hand takes significantly longer to produce the same plans. As with Logistics, this is because GoDeL-$L$ has to generate a larger fraction of the plan via action-chaining, thus slowing the planner down. LAMA solves the smaller problems very quickly, but slows down on the larger problems, solving problems larger than 60 blocks slower than GoDeL-$M$.

With respect to plan length, all the planners produce plans of similar length with the exception of LAMA, which generates much longer plans than the others.

**Depots**. The complete method set for Depots consisted of five methods and the same derived predicate used in Blocks World. The moderate (M) method set consisted of the same set of methods, but with all knowledge related to the Logistics subproblem removed. The low (L) method set is identical to the low (L) method set of Blocks World, consisting of just a single method achieving on$(X)$. We compared the planners on 25 $n$-package problems for each $n = 8, 16, \ldots, 80$.

As shown in Figure 3, GoDeL-$C$ has the best running times, 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.

### 5.3 Discussion

The main conclusions from our experimental study were:

- GoDeL *performed at par or even better than GDP-$h$ when given complete domain knowlege.*
- GoDeL*'s technique of using subgoal inference and action-chaining in tandem helped it to cope with incomplete domain knowledge.* Table 1 gives the average fraction of action-chaining (as opposed to method decomposition or relevant-action application) used in generating plans as a function of the domain and amount of domain knowledge given to GoDeL. We can see that as the amount of domain knowledge is reduced, the fractions gradually increase, indicating that GoDeL is able to use whatever little knowledge is given to it. The fraction for GoDeL-$M$ in Logistics is particularly striking; even with one of the methods removed, the plans were generated without resorting to any action chaining. This was because GoDeL was able to automatically infer the subgoals of the missing method and insert them into the goal network using its subgoal inference algorithm.
- GoDeL*'s performance improves as more planning knowledge is given to it.* This is evident from the fact that GoDeL-$L$ (GoDeL with low knowledge) was outperformed by GoDeL-$M$ (GoDeL with moderate knowledge), which in turn was outperformed by GoDeL-$C$ (GoDeL with complete knowledge) across all three domains under consideration.

## 6 Related Work

GoDeL's goal semantics differs from the *goal task* semantics in [Erol *et al.*, 1994]: for a task $t$ and goal $g$, Erol's semantics can't invoke arbitrary methods that achieve $g$ but aren't labeled with $t$. It also differs from the semantics for tasks in [Kambhampati *et al.*, 1998], which requires manually annotating tasks with preconditions and effects, and making special designations to indicate which conditions should be achieved via decomposition and how to achieve them.

[Castillo *et al.*, 2001] propose the use of *articulation functions* to translate literals in a higher level of abstraction to a lower level and use partial-order planning algorithms to resolve these open subgoals at each level. [Biundo and Schattenberg, 2001] also propose a similar framework; they annotate abstract tasks with preconditions and effects and then use similar partial-order planning algorithms to refine these tasks and generate primitive plans. Our approach differs from these both in the kinds of solutions we consider (extension of classical planning) as well as the techniques we use (forward state-space heuristic search; use of landmarks).

Duet [Gerevini *et al.*, 2008] combined the SHOP2 HTN planner with the domain-independent LPG planner [Gerevini *et al.*, 2003] to solve planning problems with partial knowledge. Analogously to Kambhampati *et al.*'s formalism, it required annotations on non-primitive tasks to indicate whether tasks should be solved by SHOP2 or by LPG.

[Alford *et al.*, 2009] provided a technique for automatically translating a restricted form of HTN methods into PDDL, so that classical planners can be used to solve HTN planning problems. This translation technique, can be used (subject to some restrictions) with partial sets of HTN methods,

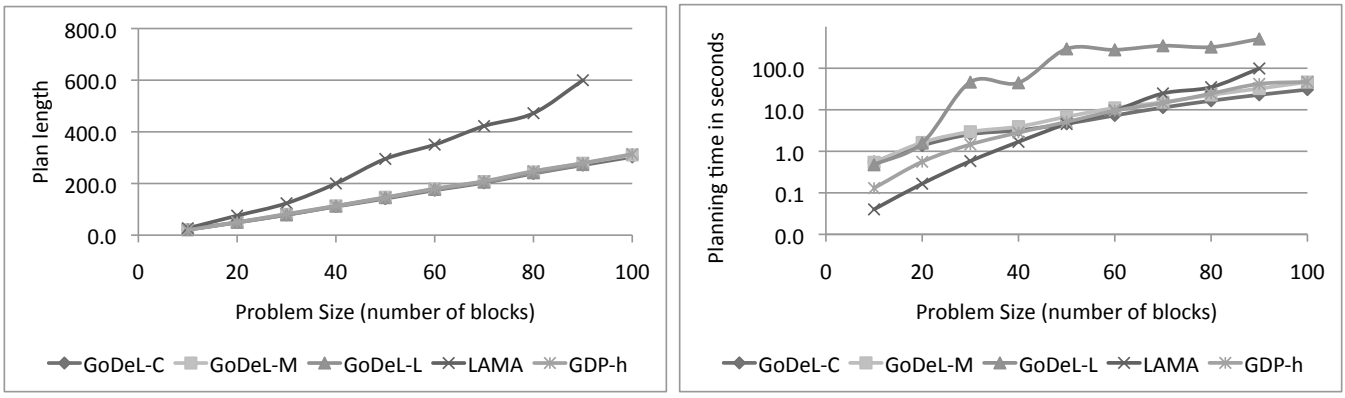


Figure 2: Average plan lengths and running times (in logscale) in the Blocks domain, as a function of the number of blocks. Each data point is the average of 25 randomly generated problems. GoDeL-$L$ and LAMA could not solve 100-block problems.

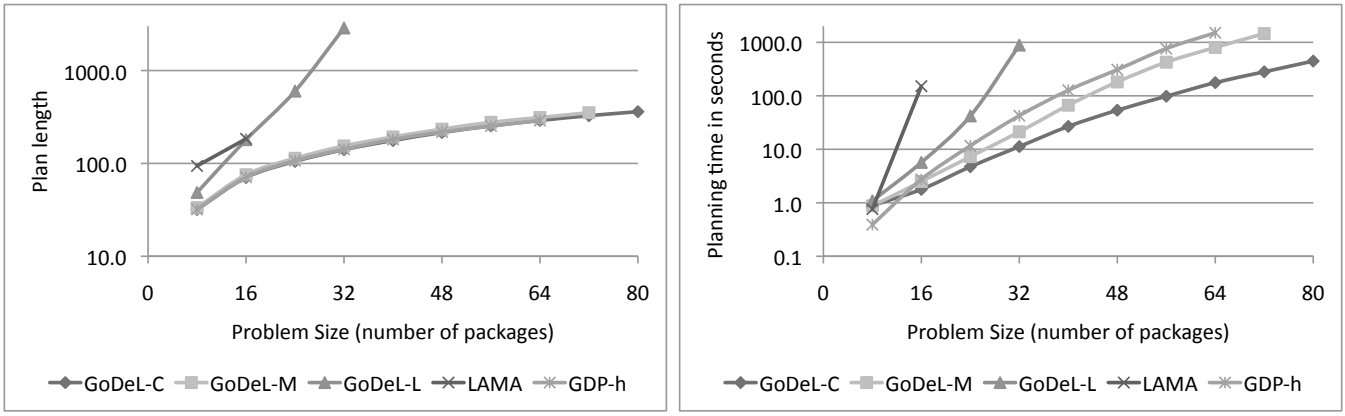


Figure 3: Average plan lengths and running times (both in logscale) in the Depots domain, as a function of the number of packages to be delivered. Each data point is an average of 25 randomly generated problems. GoDeL-$M$ could not solve 80-package problems; GDP-$h$, GoDeL-$L$ and LAMA could not solve problems of size $> 64$, 32 and 16 respectively.

but is correct only in domains where the recursion depth is bounded, and this bound needs to be supplied manually.

Another related work is that of [Elkawkagy *et al.*, 2012] who propose a landmark-based heuristic for HTN planning. This is in contrast to our work, in which we use landmarks to generate part of the hierarchy.

[Shivashankar *et al.*, 2012] described a hierarchical goal-based planning formalism that uses methods to decompose goals into sequences of subgoals, and also provided the GDP-$h$ planner used in our experiments. Unlike GoDeL, GDP-$h$'s definition of a solution requires method decomposition; hence GDP-$h$ can't guarantee solutions to solvable classical planning problems unless given a complete set of methods.

## 7 Conclusion and Future Work

In our formalism, methods provide suggestions for ways to accomplish classical goals (rather than impose requirements for how to accomplish tasks, as in HTN planning formalism in [Ghallab *et al.*, 2004, Chap. 11]). Thus, GoDeL can work correctly with complete domain knowledge (a full set of methods for every task), no domain knowledge (other than the basic actions, of course), or anything in between. It will terminate with a correct solution if one exists; the amount of knowledge only determines how fast that solution is found.

If GoDeL is given methods only for hard subproblems, its subgoal inference algorithm can automatically figure out subgoals for which those methods are relevant. When given complete domain knowledge, GoDeL performs at par or better than GDP-$h$ [Shivashankar *et al.*, 2012]. Thus, we believe that GoDeL naturally supports 'incremental' domain authoring; users can build partial domain models, evaluate the planner's performance with that, and then incrementally improve the model if required.

For future work, we will further investigate relations between landmarks and problem decomposition, e.g., ways to automatically learn methods by generalizing landmarks.

**Acknowledgments** This work was supported in part by ARO grant W911NF1210471, ONR contract N0001412C0239 and grant N000141210430, DARPA contract FA865011C7191 and 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.

## References


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

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

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

[Castillo *et al.*, 2001] L Castillo, J Fdez-Olivares, and A González. On the adequacy of hierarchical planning characteristics for real-world problem solving. *European conference on planning (ECP-2001)*, pages 169–180, 2001.

[Elkawkagy *et al.*, 2012] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo. Improving hierarchical planning performance by the use of landmarks. In *AAAI*, pages 1763–1769, 2012.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana 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.

[Fox and Long, 2002] Maria Fox and Derek Long. International planning competition, 2002. http://planning.cis.strath.ac.uk/competition.

[Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *JAIR*, 20:239–290, 2003.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, A. Saetti, and N. Waisbrot. Combining domain-independent planning and HTN planning. In *ECAI*, pages 573–577, July 2008.

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

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

[Hoffmann *et al.*, 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

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

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

[Shivashankar *et al.*, 2012] Vikas Shivashankar, Ugur Kuter, Dana S. Nau, and Ronald Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *AAMAS*, pages 981–988, 2012.

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