Learning Plan Knowledge in Tasks with Information Gathering

Tolga Könik and Negin Nejati

Computational Learning Laboratory
CSLI, Stanford University
Stanford, California 94305
{negin, konik}@stanford.edu

Abstract

In this paper we describe a framework for learning plan knowledge using expert solution traces in domains that include information-gathering tasks. We describe an extension to a special class of hierarchical task networks (HTNs) that can naturally represent information-gathering tasks and partial plans. We show how an existing analytical learning algorithm designed to learn a special form of HTNs can be improved to address the issues raised in information gathering. We also describe how our learning algorithm can use facts the expert explicitly asserts during task execution. Finally we report the preliminary evaluation of our system on a web form based scheduling and information-gathering domain.

Introduction

Classical planning systems often assume complete information about the state of the world. However, in many real-world problems, information necessary to complete the tasks are acquired during the execution of a task. In fact, some actions may have the sole purpose of gathering new information. Knowledge-based planning methods may be flexible enough to represent such information-gathering behavior, but encoding the required knowledge manually is difficult and time consuming. In this paper, we demonstrate a class of hierarchical task-networks (HTNs) (Nau et al., 2001) that are capable of representing planning knowledge even in domains that need information gathering. We also present a learning algorithm that constructs valid HTNs in these domains, thus addressing the difficulty of manually encoding background knowledge.

Our learning algorithm, LIGHT (**L**earning **I**ncremental **G**oal-driven **H**ierarchical **T**ask-networks), uses sequences of operators taken from expert solutions as the primary input, interprets them in the context of background knowledge, and constructs a set of HTN methods. If learned correctly, these methods can be used to solve similar problems.

Unlike traditional HTNs, the methods that LIGHT generates are indexed by the goals they achieve. LIGHT is similar to explanation-based learning (EBL) algorithms (Ellman, 1989) in generating an explanation structure using background knowledge, but is different in that it acquires hierarchical structures and learns more general preconditions. In our earlier work (Nejati et al., 2006), we described an early version of LIGHT and presented experimental evidence indicating that it outperforms an explanation-based macro-operator learning system. However, we conducted these experiments on classical planning domains where the state of the environment is completely observable.

In this paper, we propose a formalism for representing information-gathering tasks in a goal-driven HTN framework and we claim that a modified version of LIGHT can support information gathering. We also argue that a key characteristic of information-gathering domains is the importance of information flow between tasks, and that learned methods can be improved by analyzing constraints on information flow. Finally, we claim that LIGHT can use and recreate explicitly stated assertions that the expert provides during problem solving. This kind of expert input is particularly useful in information-gathering domains where much of expert activity may involve inferences that the learner cannot observe.

In the following sections, we first describe how goal-driven HTNs can be learned and used during performance. Next, we describe our changes to LIGHT to address challenges of information gathering. Finally, we conclude after evaluating our approach and discussing related work.

Learning and Using Goal-Driven HTNs

We will use a medical evacuation domain for motivating examples. This domain includes tasks like handling travel and hospital arrangements of patients, including:

- *Information-gathering actions*, such as lookup-flights(?From ?To ?Flight), which increase the planner's knowledge about the state of the world by retrieving information about available flights;
- *physical actions*, such as reserve(?Patient ?Flight), which are similar to classical planning actions in that they cause changes in the external world; and
- *assertion actions,* which enable the expert to keep an explicit record of the beliefs he/she has formed. For example if the expert believes that the patient-2 will depart from airport-5, he/she can selects the assert-

departure-airport(patient-2, airport-5) to explicitly state that belief.

Next, we review the representational and performance assumptions of LIGHT, followed by a description of LIGHT's learning process in more detail.

Table 1. LIGHT Concepts

Primitive concept instances

```
airport(SFO)
distance(SFO, San Francisco, 13miles)
```

Nonprimitive concepts definitions

```
close-airport(?location ?airport) ←
relations:
  airport(?airport)
  distance(?airport ?location ?distance)
tests:
  ?distance < 100
```

Representation

LIGHT uses two distinct structures to represent task knowledge. *Concepts* describe the environment whereas *skills* are methods that can be executed to achieve the agent's goal. Both concepts and skills are organized in hierarchies, which consist of primitive structures in the terminal nodes and nonprimitive ones in the internal nodes.

Table 2. Primitive and Nonprimitive LIGHT Skills

```
time-at-departure-airport(?patient ?time2) ←
  start:    origin-location(?patient ?loc)
            location-at(?patient ?loc ?time1)
  subgoals: departure-airport(?patient ?airport)
            arrival-time(?patient ?loc ?airport
                         ?time1 ?time2)

departure-airport(?patient ?airport) ←
  start:    origin-location(?patient ?loc)
  subgoals: close-airport(?loc ?airport)

arrival-time(?patient ?from ?to ?time1 ?time2) ←
  effects:  location-at(?patient ?to ?time2)
            ?time1 < ?time2
  start:    location-at(?patient ?from ?time1)
:actions    getArrivalTime(?patient ?time1
                           ?from ?to)
```

LIGHT's concepts constitute the basic vocabulary for representing the state of the environment. The perceived state is represented with a set of primitive concept instances, which are encoded as ground literals. The nonprimitive concepts are Horn clauses that LIGHT can use to form new inferences based on primitive concepts and other nonprimitive concepts. Table 1 depicts examples of concepts LIGHT uses in the medical evacuation domain.

Skills describe methods that are used to execute tasks. A set of LIGHT skills compose a specialized form of *hierarchical task network* (Nau et al., 2001) in which each skill is indexed by the goal it is expected to achieve.

Each skill has a goal and a set of start conditions, both of which are represented with concept predicates (Table 2). The goal of a skill denotes a partial representation of the environmental state that should be achieved when the skill is successfully executed. Start conditions encode the conditions under which a skill is considered for execution. Primitive skills are simple methods which invoke actions that can be executed, whereas nonprimitive skills are complex methods that decompose a task into a sequence of subtasks.

Performance Mechanism

Our performance engine (Langley & Choi, 2006a) executes HTNs in a goal-driven but reactive manner. Given a top-level goal, the agent finds a path in the skill hierarchy such that the path terminates with a primitive skill, and the start conditions of all selected skills on the path are satisfied when they are first selected. The selection of a skill path and execution of a selected action corresponds to one execution cycle. As soon as the execution engine finds a skill path, it executes the action that terminates the path. For each selected skill, the subgoals are accomplished in order over multiple execution cycles. Subgoals that are already achieved are skipped and at any level of the skill hierarchy, if a previously achieved subgoal is undone, the execution returns to that previous subgoal.

This approach is goal-driven because finding an applicable skill path corresponds to an abstract partial plan towards the goal in terms of intermediate goals. The execution engine is also reactive because the skill path is reevaluated in each execution cycle. Moreover, some persistence is achieved since the execution engine prefers the skill path selected in the previous cycle as long as it is still applicable.

HTNs can also be used for planning (Nau et al., 2001). In this context, the skills represent possible ways of decomposing tasks into subtasks, and the planner limits its search to the space of action sequences that are consistent with some decomposition of the skills.

Learning Mechanism

Generating the domain specific nonprimitive skills manually is usually expensive and prune to error. In response, in Nejati et al. (2006) we introduced a learning algorithm which reduces this effort by expanding upon conceptual background knowledge and action models about a domain to learn nonprimitive skills which achieve complicated tasks.

LIGHT learns from successful problem-solution pairs provided by the expert. A problem consists of a concept instance representing a desired goal and the initial state of

the environment. The solution includes a sequence of primitive skill instances that the expert chose to achieve the goal as well as the sequence of the observed states of the environment.

LIGHT interprets the expert's steps in the context of the achieved goal and background knowledge, and generates goal-indexed HTNs that can solve tasks with similar goals. Like other analytical methods, LIGHT generates an explanation of how the goal is achieved by using background knowledge. It does so by recursively regressing from the goals, either decomposing them into subgoals using conceptual knowledge, or explaining them in terms of the effects of the primitive skills.

For a given goal G, the explanation procedure starts by focusing on the final primitive skill S performed by the expert. If G is one of the effects of S, the algorithm explains G using *skill chaining*. In skill chaining, if P_S, the precondition of S, is not present in the initial state, S is not directly executable and LIGHT learns a skill that achieves Ps. In this case, LIGHT conjectures that the earlier steps of the expert were taken to achieve P_S and tags P_S as the new goal and explains it using those previous steps.

When the goal G is not one of the effects of the final primitive skill, LIGHT explains it using *concept chaining*. LIGHT determines the subgoals SG_i using the concept definition of G. Each subgoal SG_i is explained by considering all primitive skills executed prior to SG_i 's addition to the state as potential contributors. This forms new problem-solution pairs that are explained the same way G was explained. This routine is performed recursively until all the goals introduced along the way are explained.

After the explanation is completed, the resulting explanation structure is used to construct an HTN, by generalizing ground terms in the explanation to variables. Each learned skill is indexed after the goal it achieves and contains an ordered list of subgoals. If the new skill is constructed with skill chaining using the primitive skill S, the subgoals are selected as the precondition of S and the primitive skill S. On the other hand, if the explanation uses concept chaining on a concept G, the learned skill has a generalization of G as the goal and the subgoals of G are used to construct the subgoals of the learned skill. These skill subgoals are ordered in the order they are added to the expert solution trace. If a subgoal of a skill cannot be explained by the expert's actions, it is converted to a condition that is tested during the execution of that skill and no skill is learned for that subgoal. This might happen due to changes in the environment that cannot be explained with expert actions.

Learning in Information-Gathering Task

In our earlier work, we presented experimental evidence showing that LIGHT outperforms an explanation-based macro-operator learning system. However, those experiments were conducted in classical domains that did not require information gathering. In this section, we describe our extensions to LIGHT for supporting information gathering.

Representing Information-Gathering Skills

Our learner was originally developed to learn skills in domains that only involved physical actions and our execution engine supported only goals that are fully instantiated ground literals. A key idea in representing information-gathering tasks in LIGHT's framework is the use of existential variables in goals and effects.

Information-gathering actions have effects with uninstantiated variables. These variables are filled only after the execution of an action and they encode the knowledge gained from that action. The information-gathering actions represent a special form of actions with partially known effects. For example in Table 2, although the structure of the effects of get-arrival-time is known to the learner, the effects can be determined completely only after the action is executed. If the action model of the agent is correct, the grounded versions of the effects will be observed and added to the belief state of the agent after the action is executed.

As with information-gathering actions, we can represent *information-gathering tasks* with partially instantiated goals. Consequently, for a given goal, the purpose of a skill is not only to achieve a state where its goal holds, but also to ground the uninstantiated goal variables that encode the information acquired by the task. For example, in Table 2, when the goal time-at-departure-airport(?patient ?time2) is called with ?patient instantiated and ?time2 uninstantiated, first the subgoal departure-airport finds an appropriate airport, by querying airports that are close to the departure location. Next, arrival-time queries a Web service that estimates the travel time between the current location and the selected airport and returns an instantiation for ?time2.

The proposed HTNs with existential goals are similar to the HTNs described in Kuter et al. (2005), with the difference that the heads of our HTNs are goals that can be tested on the belief state of the agent. As a result, the query variables can be instantiated with previously acquired knowledge without skill execution. Moreover, during the execution of a skill, if the agent collects sufficient knowledge to infer an instantiation for its goal query, unlike traditional HTNs, the skill is successfully completed even it has remaining subgoals that are not executed yet.

Allowing goals with existential variables requires some adjustment in execution and learning. During execution, after a skill with a partially instantiated goal is selected, if the goal of the skill matches the belief state of the agent, the existential variables in the goal are instantiated with values that satisfy the goal predicate on that state. For example, while executing the task time-at-departure-airport

(Table 2), if the departure airport of the patient is already known, the subgoal departure-airport retrieves this information from the belief state of the agent without requiring the execution of that skill.

Another important aspect of information-gathering tasks is how the performance agent accesses dynamically collected information and uses that information in consecutive decisions. The goal-driven HTN representation provides two mechanisms for this purpose. New information can be transmitted through variables shared between the subgoals, and between a parent goal and subgoal. For example in departure-airport skill (Table 2), the variable ?airport is determined by close-airport and returned to departure-airport. Moreover, conditions that are tested during the execution of a task retrieve information from the belief state of the agent. For example, suppose that the performance agent is executing the departure-airport skill in Table 2. After the agent determines the initial location of the patient, if it has already queried the airports close to that location in previous tasks and therefore that information is in the belief state, the agent will not repeat the same query but simply reuse the previously acquired information.

Considering Information Flow in Learning

Broadening LIGHT's representation with existential variables in the goals and in action effects creates new challenges for the learner. To demonstrate the problem, consider an expert whose goal is to determine the time patient42 arrives at airport1. This goal is represented as location(patient42 airport1 ?time). To achieve this goal, the expert first assigns the patient a flight that takes him to airport1 and then queries the arrival time of that flight. However, the expert does not repeat the query, if he/she has already a belief about the result of this query. In this problem, a naive version of LIGHT learns the incorrect skill in Table 3. Here, LIGHT learns an incorrect order for the subgoals because the expert knew the arrival time of the particular flight he/she selects later even before that assignment decision is made.

This is an undesired skill because at the time the goal arrival-time is called, both of its arguments are uninstantiated. Consequently, the execution engine randomly selects among known flights to the agent (regardless of their destination), and then fails if the assigned flight is determined to be inconsistent with the destination airport (the body of the assigned skill contains a condition that tests the destination airport information).

Table 3. An incorrectly learned LIGHT skill

```
location(?patient ?dest-airport ?time)←
subgoals:
  arrival-time(?flight ?time) ;incorrect order
  dest-airport(?patient ?dest-airport)
  assigned(?patient ?flight)
```

In the core of the problem lies LIGHT's lack of knowledge about the input/output structure of goals that best model expert's behavior. For example, suppose LIGHT is explaining how an intermediate goal instance $p(a, b)$ is achieved on a solution trace. LIGHT does not know whether the expert's goal can be best modeled as a query of finding a $?y$ instantiation that satisfies $p(x_0, ?y)$ for a previously instantiated value x_0, or as a more general $p(?x, ?y)$ query of finding $?x$ and $?y$ instantiations. Consequently, LIGHT may fail to determine when the goal is accomplished. Note that this problem did not occur in our earlier work because our previous representational assumptions prohibited goals with existential variables.

If the learner explicitly reasons about information flow among variables in a learned skill, it may make better predictions about the input-output variable structures of the goals and construct skills that minimize execution time failure. To improve LIGHT in that direction, we devised a postprocessor which reorders the subgoals of skills that conflict with known constraints about information flow. This postprocessor is guided by two constraints, the input-output structure of the main goal, and the input arguments of the actions that are required to be instantiated. Based on these constraints, the learner reorders the subgoals if necessary to ensure a consistent information flow. For example, in Table 3, the postprocessor first analyzes the substructures of arrival-time and concludes that arrival-time cannot be executed with two uninstantiated arguments; therefore it must be moved to a later location. This process continues incrementally, until arrival-time becomes executable after the assigned subgoal.

Learning and Using Expert Assertions

One of the most important bottlenecks of learning from expert traces is that the internal reasoning of the expert is not always available to the learner. Some earlier learning by observation systems such as behavior cloning (e.g., Sammut et al., 1992) are applied in reactive tasks where this problem is minimized since most of the critical components of the behavior are directly observable. Other systems use background theory to predict mental reasoning of the expert using induction (e.g., Könik & Laird, 2006) or explanatory analysis (Nejati et al., 2006). In domains where most of the expert's activity is about gathering information and forming new inferences using collected information, the observed changes in the environment may remain limited as the majority of the changes occur in expert's mind. In such domains, explicitly stated assertions might help the learner to predict mental reasoning of the expert.

The assertion actions are not appropriate for all domains and sometimes they may interfere with task-performance by putting additional burden on the expert. Nevertheless, in some information-gathering domains, these assertions may even help the expert to keep track of acquired information. Moreover, recording important inferences

may be a required part of the task the expert is conducting and the learner may be required to replicate that behavior.

We use an expert behavior generation interface, which enables the expert to explicitly assert pieces of acquired information by inserting them into tables. A slightly updated version of LIGHT can use these explicitly stated assertions and learns to acquire, state, and use similar facts.

LIGHT achieves this by using a special encoding for assertion actions. For an action that asserts the fact *P*, the effect of the action represents a concept, "*P* is asserted", which basically means that *P* is recorded on a table. The precondition of this action contains only *P*. If *P* is considered as a goal that needs to be explained during learning, and if there is an action that assert *P*, LIGHT learns a new skill, which first achieves *P*, and then asserts it.

Evaluation

Our evaluation of LIGHT on information-gathering tasks is in its preliminary stages. We have tested our learning system in a medical evacuation domain where the expert uses a web form based interface[1] to select actions. Physical actions such as reserve call web services to make (simulated) changes in the environment, and return the result of the action (ie. "reservation accepted" or "the flight is full"). The query actions such as lookup-flights return a list of answers for each the expert query, for example a list of airports that are close to a given city. The expert is also provided with a list of tables that he/she can use to enter acquired or inferred information. The expert is given the goal of reserving hospital beds for a set of patients and arranging their travel plans.

A symbolic representation of the expert's interaction with the web interface is recorded to the solution traces, which contain an ordered list of actions, with the selected arguments, as well as their results. The results of the actions symbolically summarize the information the expert receives from the queries using ground literals such as close-airport("San Francisco",SFO) or reserved(patient-1, hospital-2).

In our evaluation, we used an expert solution trace where the expert made queries to find a travel route for multiple patients, made a tentative travel plan by recording information such as local and destination airports, used web services to estimate ground transportation time to the airport, and reserved flights.

LIGHT learned a HTN using the goal and the solution traces. Since our performance component is not fully connected to the web interface yet, we haven't tested the learned task networks with systematic experiments. Instead, we inspected the learned HTNs, and manually traced their predicted behavior.

[1] medical evacuation domain and its web based interface is developed by BBN Tech. http://www.bbn.com/

The learned HTN seems to replicate the expert behavior under similar conditions. Most of the learned methods contain goals that are called with partially instantiated. Our major hypothesis of representing information-gathering tasks with existential goal and effect variables works on these examples. The existential goal variables are either filled by the variables of the subgoals, or the activities these methods conduct and make the goals inferable from the belief state of the agent. The extension of LIGHT that reasons about information flow can learn correct skills that would have been learned incorrectly with its earlier version. Finally, the learned HTNs are able to replicate the behavior of the expert in filling tables with acquired and inferred information.

Related Work

Despite a substantial literature on learning for planning (Zimmerman, 2003), there have been remarkably few efforts toward acquiring hierarchical plan structures. Most work in this area does not consider information-gathering tasks and they are usually focused on learning from the results of search (Langley & Choi, 2006b).

An alternative approach is to learn from traces of an expert's behavior. Research on behavioral cloning (e.g., Sammut, 1992) incorporates this idea, but typically learns rules for reactive control. It transforms the observed traces into supervised training cases and induces reactive controllers that reproduce the behavior in similar situations. This approach typically casts learned knowledge as decision trees that determine which actions to take based on sensor input (e.g., Urbancic & Bratko, 1994).

Systems that learn from expert solution traces rarely learn hierarchical structures. Könik and Laird (2006) report an approach to behavioral cloning that uses a relational representation and learns hierarchical tasks, but it requires the expert to annotate the trace with information about the start and end of tasks. Ilghami et al. (2005) describe a method for constructing HTNs, but they assume the hierarchical structure is given. Tecuci's (1998) Disciple acquires hierarchical rules, but it requires user information about how to decompose problems and the reasons for decisions.

Other research on learning skills by observing others' behavior has, like our own, utilized domain knowledge to explain traces. Some work on explanation-based learning (e.g., Segre, 1987; Mooney 1990) took this approach, as did the paradigms of learning apprentices (e.g., Mitchell et al., 1985) and programming by demonstration (e.g., Cypher, 1993; Lau et al., 2003). Those approaches haven't focused on the acquisition of hierarchical skills.

Conclusion

In this paper we described an analytical learning algorithm that constructs plan-knowledge for domains with

information-gathering tasks. Our approach builds on our earlier work of learning goal-driven HTNs (Nejati et. al, 2006) but expands it to address the issues raised in information-gathering domains.

We proposed an extension to our representational framework to support information gathering. A key extension to our earlier representation is the inclusion of existential variables in goals and action effects. We showed that this change poses a new challenge for the learner because the prediction of correct goal structures becomes more difficult problem. We claimed that the learner can address this issue by reasoning about the information flow among goal variables. We have also discussed how the learner can use explicitly stated expert assertions. We have reported and discussed the preliminary performance of LIGHT on a web form based scheduling domain.

Although the learned knowledge looks promising, a more systematic evaluation remains to be done. One interesting direction we like to explore is to learn concept definitions by inductively generalizing explicitly stated expert assertions. LIGHT will later use those concepts in its analysis thus improving its ability to learn with incomplete background knowledge.

Another direction is to learn preferences between valid goal selections a learned HTN predicts. For example if there are multiple valid flight for a patient, the expert might always prefer the latest departing one to keep the earlier flights less populated for an emergency situation. When this distinction cannot be explained with background knowledge an inductive method can be employed to improve the quality of the learned knowledge.

Our work on learning HTNs in information-gathering tasks is in its early stages and we are in the process of systematic evaluation of our system. Nevertheless, we observed some promising results and we expect that our work will contribute to the planning field in addressing real world problems.

Acknowledgments

We thank Pat Langley, Paul O'Rorke, Ugur Kuter and Mark Burstein for insightful discussions. This paper reports research sponsored by DARPA under agreement FA8650-06-C-7606. The U.S. Government may reproduce and distribute reprints for Governmental purposes notwithstanding any copyrights. The authors' views and conclusions should not be interpreted as representing official policies or endorsements, expressed or implied, of DARPA or the government.

References

Cypher, A. ed. 1993. *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.

Ellman, T. 1989. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys, 21*, 163-221.

Ilghami, O., Nau, D. S., Muñoz-Avila, H., and Aha, D. 2005. Learning Preconditions for Planning from Plan Traces and HTN Structure. *Computational Intelligence, 21* (4), 388-413.

Könik, T. and Laird, J. E. 2006. Learning Goal Hierarchies From Structured Observations and Expert Annotations. *Machine Learning, 64*, 263-287.

Kuter, U., Sirin, E., Nau, D., Parsia, B., and Hendler, J. 2005. Information gathering during planning for web service composition. *Journal of Web Semantics, 3* (2-3), 183-205.

Langley, P., & Choi, D. 2006a. A unified cognitive architecture for physical agents. Proceedings of the Twenty-First *National Conference on Artificial Intelligence*. Boston: AAAI Press.

Langley, P., & Choi, D. 2006b. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, *7*, 493-518.

Lau, T. A., Domingos, P., & Weld, D. S. 2003. Learning programs from traces using version space algebra. *Proceedings of the Second International Conference on Knowledge Capture*, 36-43. Sanibel Island, FL: ACM.

Mitchell, T. M., Mahadevan, S., and Steinberg, L. I. 1985. LEAP: A learning apprentice for VLSI design. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 573-580. Los Angeles, CA: Morgan Kaufmann.

Mooney, R. J. 1990. *A general explanation-based learning mechanism and its application to narrative understanding*. San Mateo, CA: Morgan Kaufmann.

Nau, D. Muñoz-Avila, H. Cao, Y. Lotem, A. and Mitchell, S. 2001. Total-Order Planning with Partially Ordered Subtasks. *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence*, 425-430. Seattle, CA: Morgan Kaufmann.

Nejati, N., Langley, P., & Könik, T. 2006. Learning hierarchical task networks by observation. *Proceedings of the Twenty-Third International Conference on Machine Learning*, 665-672. New York, NJ: ACM.

Segre, A. 1987. A learning apprentice system for mechanical assembly. *Proceedings of the Third IEEE Conference on AI for Applications*, 112-117.

Tecuci, G. 1998. *Building intelligent agents: An apprenticeship multistrategy learning theory, methodology, tool and case studies*. London: Academic Press.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. 1992. Learning to fly. In Sleeman, D., Edwards, P. eds. *Proceedings of the 9th International Conference on Machine Learning*. Morgan Kaufmann, 385-393.

Urbancic, T., & Bratko, I. 1994. Reconstructing human skill with machine learning. *Proceedings of the Eleventh European Conference on Artificial Intelligence* , 498-502.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, *24* , 73-96.