Improving Acquisition of Teleoreactive Logic Programs through Representation Change

Nan Li,[1] **David J. Stracuzzi,**[2] **and Pat Langley**
Computer Science and Engineering
Arizona State University
Tempe, Arizona 85281 USA
nli1@cs.cmu.edu, david.stracuzzi@gmail.com, langley@asu.edu

Abstract

An important form of learning involves acquiring skills that let an agent achieve its goals. While there has been considerable work on learning in planning, most approaches have been sensitive to the representation of domain context, which hurts their generality. A learning mechanism that constructs skills effectively across different representations would suggest more robust behavior. In this paper, we present a novel approach to learning hierarchical task networks that acquires conceptual predicates as learning proceeds, making it less dependent on carefully crafted background knowledge. The representation acquisition procedure expands the system's knowledge about the world, and leads to more rapid learning. We show the effectiveness of the approach by comparing it with one that does not change domain representation.

Introduction

Hierarchical task networks (Nau et al. 1999; Wilkins and des Jardins 2001) have received increased attention for their efficient planning capabilities. However, generating hierarchical task networks by hand is often difficult and time consuming. For this reason, there has been a growing body of work on learning in this framework (Ilghami et al. 2002; Langley and Choi 2006; Nejati, Langley, and Konik 2006). Both Langley and Choi's system and Nejati et al's work represent the world with a hierarchical conceptual knowledge base, which provides a vocabulary for the system to describe state features with different levels of abstraction. Their skill learners then use this vocabulary to guide the precondition and subtask acquisition process for the procedural methods. A good representation of the world state reveals essential features of the world, and helps the system to decide which skill to apply in solving the problem.

However, such a representation is often not available. If the structure of the concept hierarchy does not reveal the critical features corresponding to the task/subtask hierarchy, the methods acquired may not guide the system to achieve its desired goal. In this paper, we refer to these concept hierarchies as *ill-structured concept hierarchies*. Manually coding *well-structured representations* often requires time and domain expertise. Hence, designing a learning mechanism that performs well across various domain representations, even when representations lack of essential features, is important. One possible way of achieving this is to build learners that improve their representations during skill acquisition.

In this paper, we consider the problem of learning methods with ill-structured concept hierarchies in a representational formalism, *teleoreactive logic programs*, which is a special case of hierarchical task networks. Careful inspection shows that skills acquired from an ill-structured concept hierarchy often have preconditions that do not accurately describe the applicable situations. To acquire preconditions with more accurate coverage, there are two primary issues. First, the preconditions should be sufficiently general so that methods will be used both in situations that are similar to those encountered during learning, and in situations that are unfamiliar. Second, preconditions should be specific enough so that procedural clauses that cannot achieve the goal in the current situation will not be selected for execution. If the ill-structured concept hierarchy lacks concepts that could effectively describe a method's preconditions, then the method learner may not acquire effective skills. One approach to addressing this issue is to automatically construct such concepts, and add them to the existing hierarchy.

We propose an algorithm that acquires both the conceptual clauses for preconditions and the procedural clauses for executing the solution by analyzing the structure of solved problems. Our approach differs from most of the other hierarchical task network learning algorithms in that the preconditions acquired are no longer simple conjunctions of existing predicates, but are instead more complicated concepts describing higher-level abstractions of the world. The new conceptual and procedural knowledge expands their respective knowledge bases, and can be used to guide learning on future tasks. We claim that our approach improves the performance of teleoreactive agents by guiding selection of procedural clauses.

A Review of Teleoreactive Logic Programs

Teleoreactive logic programs are a framework for encoding procedural knowledge that incorporates ideas from logic programming, reactive control, and hierarchical task networks (Nau et al. 1999; Wilkins and des Jardins 2001). They have a syntax similar to the first-order Horn clauses used in Prolog, and so may be viewed as logic programs. The term

[1]Now with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA.

[2]Now with the Cognitive Modeling Department, Sandia National Laboratories, Albuquerque, NM, USA.

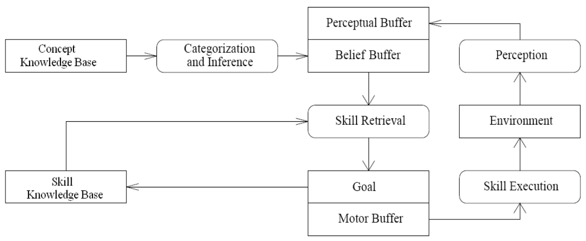


Figure 1: System diagram of the ICARUS architecture.

"teleoreactive" (Nilsson 1994) refers to the formalism's support for reactive execution of the goal-oriented methods over time. We have embedded teleoreactive logic programs into ICARUS, a physical agent architecture. ICARUS shares many features with agent architectures like Soar (Laird, Rosenbloom, and Newell 1986), and ACT-R (Anderson 1993), and Prodigy (Minton 1990). We will use ICARUS to refer to teleoreactive logic programs, although other implementations are possible.

To better illustrate teleoreactive logic programs, we provide an example from the card game of freecell solitaire (Bacchus 2001). Freecell begins with eight columns of stacked cards, initially random. There are also four empty home cells, one for each suit, and four empty free cells each of which can hold any single card. The objective is to move all cards from the eight columns to their respective home cells, stacked in ascending order. Only cards at the top of each column, and in the free cells may be moved. A player may move a card to one of four locations: 1) its corresponding home, 2) an empty free cell, 3) an empty column, or 4) the top of a column whose current top has rank one greater than the moved card and is of opposite color. In the application of teleoreactive logic programs to freecell described below, forty concepts and thirteen primitive skills are initially provided to the agent.

Knowledge Representation

ICARUS distinguishes conceptual and procedural knowledge, and has separate knowledge bases for each. The conceptual knowledge base stores a hierarchy of first-order Horn clauses with negation, and provides a vocabulary for the agent to describe its environment at different levels of abstraction. Each concept consists of a head, which states the predicate and the arguments of the concept, and a body that describes the conditions under which the concept is true. There are two forms of concepts. Primitive concepts have a test field in their bodies that refers only to low-level descriptions of the environment that can be observed directly by the agent. These low-level descriptions are referred as *percepts*. Non-primitive concepts have a relation field in their bodies that refers to other lower-level concepts. Sample concepts are demonstrated in Table 1. *(column-column-pair ?c1 ?c2)* is a primitive concept, whereas *(column-to-column-able ?c ?dc ?cb)* is a non-primitive concept. The non-primitive concept *(column-to-column-able ?c ?dc ?cb)* indicates that, if card *?c* and card *?dc* are the last cards in two columns, card *?c* is on card *?cb*, and card *?c* and card *?dc* forms a column-column pair, then card *?c* may be placed onto card *?dc*, which makes card *?cb* become the last card in the original column.

The procedural knowledge base stores skill knowledge

Table 1: Sample conceptual clauses from freecell solitaire.

```
;; cards ?c1 and ?c2 are of different color,
;; rank of ?c2 is 1 larger than ?c1
((column-column-pair ?c1 ?c2)
   :percepts    ((card ?c1 color ?co1 val ?v1)
                 (card ?c2 color ?co2 val ?v2))
   :tests       ((not (equal ?co1 ?co2))
                 (= ?v2 (+ 1 ?v1))))

;; card ?c may be placed onto card ?dc,
;; and ?cb is the card below ?c
((column-to-column-able ?c ?dc ?cb)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb))
   :relations   ((clear ?c)
                 (clear ?dc)
                 (on ?c ?cb)
                 (column-column-pair ?c ?dc)))

;; start condition corresponding to procedure clear
((precondition-of-clear ?dc)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb))
   :relations   ((column-to-column-able ?c ?dc ?cb)))
```

that can execute in the world in a form similar to hierarchical STRIPS operators (Fikes and Nilsson 1971). Each skill has a head, which is a defined concept that specifies the situation after the skill executes, a start condition that must be satisfied before the skill can execute, a body that describes how to achieve the desired condition, and an effects field that specifies the skill's effects. There are two types of skills, just as with concepts. Primitive skills (e.g. *(clear-and-put-on ?cb ?c ?dc)* in Table 2) have an action field that refers to actions that the agent can execute directly in the world. Non-primitive skills (e.g. *(put-on ?c ?dc)* in Table 2) have a subgoal field that specifies subgoals the agent should achieve. A skill (e.g. *column-to-column-able*) may refer to itself, either directly or through a subgoal, allowing the framework to support recursive programs.

Teleoreactive Execution and Problem Solving

Teleoreactive logic programs perform a series of distinct, internal operations to execute in a goal-directed but reactive manner. As mentioned earlier, ICARUS provides an interpreter for teleoreactive logic programs, which operates in discrete cognitive cycles, as shown in Figure 1. For each cycle, the interpreter first infers its beliefs about the environment. Then, based on its skill knowledge base and the inferred beliefs, the interpreter chooses an action towards achieving its goals, and executes it in the world.

To understand its environment, the interpreter first receives a set of percepts in the form of ground literals. It then infers beliefs about its environment by matching perceptual constants with concept arguments in a bottom-up fashion.[1] This inference process computes the deductive closure of beliefs implied by conceptual predicates and percepts. These concept instances are stored in belief memory for the agent

[1]Note the difference between this approach and that of Prolog, which evaluates predicates in a top-down, query-driven manner.

Table 2: Sample skill clauses from freecell solitaire.

```
;; Clear card ?cb by moving ?c from it to ?dc
((clear-and-put-on ?cb ?c ?dc)
  :percepts    ((card ?c)
                (card ?dc)
                (card ?cb))
  :start       ((column-to-column-able ?c ?dc ?cb))
  :actions     ((*sendtocol ?c ?dc))
  :effects     ((clear ?cb)
                (put-on ?c ?dc)
                (clear ?c)))

;; Move card ?c on to card ?dc
((put-on ?c ?dc)
  :percepts    ((card ?c)
                (card ?dc)
                (card ?cb))
  :start       ((precondition-of-put-on_s8 ?c ?dc))
  :subgoals    ((column-to-column-able ?c ?dc ?cb)
                (clear-and-put-on ?cb ?c ?dc))
  :effects     ((effect-of-non_s8 ?c ?dc)))
```

to make decisions later. Based on the inferred state of its surroundings, ICARUS uses its skill knowledge to take action towards achieving its goal. In order to decide the next move, the interpreter first retrieves an unsatisfied goal, and then chooses a skill that it expects to achieve the goal. The interpreter proceeds in a goal-oriented manner, while still remaining reactive to the environment.

When no skill applies (an impasse), the interpreter invokes a problem-solving mechanism which decomposes the current goal into subgoals. It uses both conceptual and procedural knowledge to chain backward from the goal until it finds a subgoal with an executable skill. The problem solver prefers chaining off of a skill over a concept.

When chaining off of a skill, the interpreter retrieves a skill that contains the goal in its head. Then the problem solver pushes the start condition of the selected skill onto a goal stack, and starts working on the start condition. For chaining off of a conceptual predicate, the system uses the concept definition to decompose the goal into multiple subgoals. If more than one subgoal is unsatisfied, the problem solver randomly selects one subgoal and pushes it onto the goal stack. Once a subgoal is achieved, the interpreter pops it out from the goal stack, and works on other unsatisfied subgoals until the goal is achieved. During this process, execution resumes immediately after an applicable skill is found, thereby interleaving problem-solving and execution.

Consider the problem solving example shown in Figure 2. Suppose the system is given the top-level goal of clearing the four-of-spades card, *(clear 4♠)*, and that no applicable skill is available in the current state. Given this impasse, the interpreter invokes the problem solver and chains off of the goal using the skill *(clear 4♠)*, which is shown in Table 2. This skill has the start condition *(column-to-column-able 2♡ 3♠ 4♠)* shown in Table 1 which describes the situation in which the 2♡ is the only card on the 4♠, and the player can move the 2♡ onto the 3♠. The interpreter places this start condition on the goal stack as a subgoal.

On the next cycle, the interpreter chains off of the concept *(column-to-column-able 2♡ 3♠ 4♠)* and decomposes it into *(clear 2♡)*, *(clear 3♠)*, *(on 2♡ 4♠)* and *(column-column-pair 2♡ 3♠)* through the concept definition. Of these, only

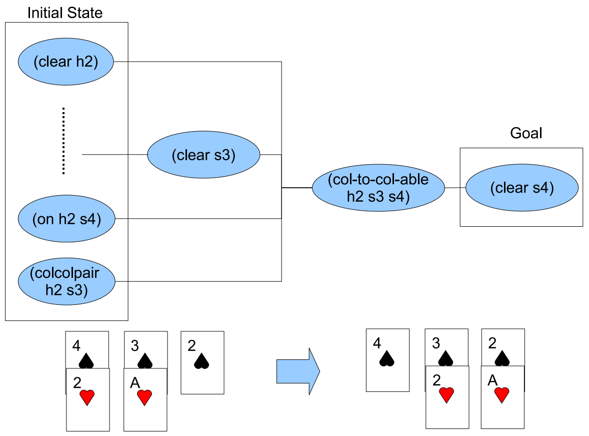


Figure 2: A freecell problem-solving example. Concepts are shown as circles, and procedural clauses are shown as rectangles.

(clear 3♠) is unsatisfied, so it gets added to the goal stack as the next subgoal. An applicable skill, *(clear 3♠)*, exists for this subgoal and the framework executes it immediately. This changes the state such that the A♡ is now on the 2♠, and makes *(column-to-column-able 2♡ 3♠ 4♠)* true. Now the skill *(clear 4♠)* becomes applicable. The system carries out the skill, and achieves its initial goal, *(clear 4♠)*. Note that although not shown, problem solving usually involves extensive search.

Skill Learning

When the problem solver achieves a goal or subgoal, the system constructs a new skill for this goal. The head of the skill is a generalized version of the goal in which constants are replaced by variables. Recall that the problem solver achieved the goal either by skill chaining or concept chaining, which shows implication for what the system learns.

If the goal was achieved by chaining off a skill, then the chaining process was accomplished by first making the start condition of the chained concept true, and then executing the chained skill. Therefore, the subgoals of the new skill should first include the start condition concept from the chained skill, followed by the subgoals of the chained skill in the order of execution. Since the first subgoal of the new skill ensures the applicability of the subsequent subgoals, the start condition of the new skill becomes the start condition of the first subgoal. In freecell, since *(clear 4♠)* was achieved through chaining off of skill *(clear-and-put-on 4♠ 2♡ 3♠)*, the skill learner constructs a new skill with two subgoals *(column-to-column-able 2♡ 3♠ 4♠)* and *(clear-and-put-on 4♠ 2♡ 3♠)*.

If the goal was achieved through concept chaining, then the problem solving process was completed by satisfying all of the subconcepts in the chained concept. Hence, the subgoals of the new skill are the unsatisfied subconcepts of the chained concept in the order of execution. The start condition of the new skill should be all the other subconcepts that were initially satisfied. In the freecell example, the subgoal *(column-to-column-able 2♡ 3♠ 4♠)* was achieved via backward chaining on the concept definition. The skill learner creates a new skill for this goal with one subgoal *(clear 3♠)*, since it was not satisfied in the initial state.

In subsequent runs, if the agent gets the same goal (maybe with different arguments), the interpreter will try to execute the new skill. In this way, if the agent meets a similar situation later, it will achieve the goal directly instead of repeating the problem-solving process again.

Learning and Using Conceptual Predicates

Having reviewed the framework and operational components of teleoreactive logic programs, we are ready to present our precondition learning mechanism. Although the skill-learning mechanism described by Langley and Choi (2006) has produced encouraging results, the start conditions acquired by their approach are sometimes overly general, and at other times overly specific. Moreover, our experience suggests that both issues can become worse when given ill-structured concept hierarchies. Our objective is to address these issues by constructing new conceptual clauses that better reflect the skill's applicability.

Learning Conceptual Predicates

Whenever a goal is achieved via problem-solving, a new skill is constructed based on the solution. The start conditions define abstractions of the world before the skill executes. To assist precondition construction, the concept leaner also acquires the effects of the new skill, which define abstractions of the world after the skill executes.

The concept learner acquires two kinds of predicates to strike a balance between specificity and generality: specialized predicates and generalized predicates. Specialized preconditions are used during execution to avoid skill overuse. Each specialized precondition is associated with one specific skill. It describes the situations in which the specific skill could apply. Generalized preconditions, on the other hand, are used during learning to avoid acquisition of over-specific preconditions. When skills are learned, the system cannot predict under what situations the skill will eventually be used to achieve the given goal. Hence, generalized preconditions describe the situations in which there exist at least one skill that could be applied to achieve the given goal. Each generalized precondition/effect is associated with one goal. The definition of a generalized precondition/effect is simply a disjunction over all specialized preconditions/effects from the skills that achieve the goal.

We now show how the two types of preconditions and effects are learned. Suppose a goal is achieved by chaining off of a concept. Then a specialized conceptual predicate C_{start} is created for the start condition of the new skill S_{new} by combining the generalized start conditions of the subgoals, $S_1, \ldots, S_n$ using a procedure similar to macro-operator composition (Mooney 1990). Specialized effects concepts, C_{effect}, are similarly created for a new skill by combining the generalized effects of subgoals of S_{new} using a similar method.

For example, as shown in Table 1, the subgoal *(column-to-column-able* 2♡ 3♠ 4♠*)* was achieved by chaining off of its concept definition. The subgoal of the new skill is *(clear* 3♠*)*. The effect field of the new skill is also initialized to *(clear* 2♡*)*, *(on* 2♡ 4♠*)* and *(column-column-pair* 2♡ 3♠*)*, which represents the initially satisfied subconcepts of the chained concept.

Table 3: Sample specialized concepts learned for freecell.

```
;; Start condition of skill column-to-column-able
((precond-column-to-column_c1 ?c ?dc ?cb)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb))
   :relations   ((clear ?c)
                 (on ?c ?cb)
                 (column-column-pair ?c ?dc)
                 (precondition-of-clear ?dc)))

;; Effect of skill column-to-column-able
((effect-column-to-column_c1 ?c ?dc ?cb)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb)
                 (card ?c2)
                 (card ?dc2))
   :relations   ((clear ?c)
                 (on ?c ?cb)
                 (column-column-pair ?c ?dc)
                 (effect-of-clear ?dc)))
```

Next the precondition learner computes the specialized start condition concept for the new skill. This contains *(clear* 2♡*)*, *(on* 2♡ 4♠*)* and *(column-column-pair* 2♡ 3♠*)*, which are the subconcepts that were true in the initial state. The generalized start condition of the only subgoal, *(clear* 3♠*)*, which is *(precondition-of-clear* 3♠*)*, also gets added to the new concept. Constants in the learned concept are replaced with variables, and the concept is added to the conceptual knowledge base. Finally, the precondition learner computes a specialized effect concept for the new skill. The subconcepts are initially set to the effects of the new skill. None of these effects get deleted upon execution of the skill's subgoal, *(clear* 3♠*)*. Several new conditions are made true by the execution, and must be added to the effect concept's relations field. This produces the new specialized effect predicate, *effect-of-column-to-column-able_c1* shown in Table 3.

If the goal was achieved by chaining off of a skill, the specialized precondition of the new skill is the generalized start condition of the first subgoal, S_1. Since the generalized start condition instead of the specialized one is used, the precondition of the new skill will generalize to all situations where there is at least one applicable skill for the first subgoal. The start condition of the chained skill is not included, because the applicability of the chained skill only depends on S_1. The specialized effect of the learned skill is set as the effect of the original skill. Since the specialized precondition and the effect are the same as some existing precondition and effect, the interpreter will not introduce new predicates for them.

For instance, since the goal *(clear* 4♠*)* was achieved through skill chaining, the skill learner constructs a new skill with *(column-to-column-able* 2♡ 3♠ 4♠*)* and *(clear-and-put-on* 4♠ 2♡ 3♠*)* as subgoals. New skills and preconditions are introduced as shown in Tables 3 and 4. The specialized start condition of the new skill is simply the generalized start condition of the first subgoal, *(column-to-column-able* 2♡ 3♠ 4♠*)*. The specialized effect of the new skill is the specialized effect of the chained skill. Constructed the specialized start condition and effect, the system needs to update the generalized start condition and effect accordingly.

Table 4: Selected skills learned for freecell solitaire.

```
;; Skill for column-to-column-able learned
;; via problem solving through concept chaining
((column-to-column-able ?c ?dc ?cb)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb))
   :start       ((precond-column-to-column_c1 ?c ?dc ?cb))
   :subgoals    ((clear ?dc))
   :effects     ((effect-column-to-column_c1
                 ?c ?dc ?cb)))

;; Skill for clear learned from problem solving
;; through skill chaining
((clear ?cb)
   :percepts    ((card ?c)
                (card ?dc)
                (card ?cb))
   :start       ((precond-column-to-column ?c ?dc ?cb))
   :subgoals    ((column-to-column-able ?c ?dc ?cb)
                (clear-and-non ?cb ?c ?dc))
   :effects     ((clear ?cb)
                (non ?c ?dc)
                (clear ?c)))
```

Notice the recursive structures that arise in the learned preconditions. This stems from the recursive nature of the problem solver, which repeatedly pushes goals onto a stack and restarts problem sotlving from the new subgoal. Also note that the learned concepts are eligible for chaining in the problem solver. This enables the framework to acquire skills based on the learned concepts.

Recall that generalized start condition/effect is a disjunction over all corresponding specialized start conditions/effects. After constructing the specialized start condition and the specialized effect, the algorithm updates the definitions of their corresponding generalized concepts by adding the just acquired definitions as disjunctive terms. Even if no predicate is introduced for skills learned with skill chaining, the generalized precondition/effect for the associated goal still gets updated. This informs the interpreter that the given goal can be achieved under a new situation, and enables the learned skills to achieve goals under unfamiliar situations. The generalized precondition and effect predicates in the freecell example are shown in Table 5.

Extending the Skill Learner

Given the newly learned predicates that represent preconditions and effects on skills, we next modify the basic skill learning algorithm to install the new concepts as the start and effect conditions of the skill. The specialized precondition concepts acquired by the concept learner are more specific than the start conditions assigned by the original skill learner. This helps to prevent the interpreter from retrieving inappropriate skills during execution, which degrades efficiency. We also expand the basic skill learning mechanism to include effects with skills by referencing the specialized effect predicates constructed in the new skills. Finally, to improve the interpreter's ability to learn recursive structures, we extend the basic skill learning mechanism to acquire new skills even when the goal was backward chained through a non-primitive skill. In this case, the subgoals of the learned skill must include the start condition of the chained skill. Sample new skills acquired in the freecell example are shown in Table 4.

Table 5: Sample generalized concepts learned for freecell.

```
;; Generalized start condition for skill column-to-column-able
((precond-column-to-column ?c ?dc ?cb)
   :percepts    ((card ?c)
                 (card ?dc)
                 (card ?cb))
   :relations   ((precond-column-to-column_c1 ?c ?dc ?cb)))

;; Generalized effect for skill column-to-column
((effect-column-to-column ?c ?dc ?cb)
   :percepts    ((card ?c)
                (card ?dc)
                (card ?cb)
                (card ?c2)
                (card ?dc2))
   :relations   ((effect-column-to-column_c1
                 ?c ?dc ?cb)))

;; Generalized start condition for skill clear
((precondition-of-clear ?dc)
   :percepts    ((card ?c)
                (card ?dc)
                (card ?cb))
   :relations   ((precond-column-to-column ?c ?cd ?cb)))

;; Generalized effect of skill clear
((effect-of-clear ?dc)
   :percepts    ((card ?c)
                (card ?dc)
                (card ?cb))
   :relations   ((clear ?cb)
                (put-on ?c ?dc)
                (clear ?c)))
```

Experimental Evaluation

To evaluate the robustness of the concept learner, we carried out experiments in two domains with rich conceptual knowledge. Logistics planning is a classic domain used in the planning literature that is naturally stated in terms of recursive concepts . Freecell also contains rich conceptual knowledge describing card patterns, which has been used in planning competitions (Bacchus 2001). Our goal is to show that with the concept learner, the extended system acquires procedural knowledge more efficiently, and the learning efficiency of the extended system is less sensitive to the quality of the given concept hierarchy.

Experimental Design

We compare our precondition learner with Langley and Choi's (2006) approach of learning from problem solving, which uses a similar skill learning mechanism but does not acquire conceptual knowledge for preconditions. Since our system constructs predicates incrementally interleaving with the process of problem solving, we did not set up the experiment with separate training and testing phases. Instead, we set up the experiment so that the system calls for the problem solver whenever it fails to achieve a goal based on known concepts and skills (Nejati, Langley, and Konik 2006). When a system achieves the top-level goal using exactly one top-level skill and without using the problem solver, we consider the problem solved successfully.

To evaluate the robustness of the learning system as well as the quality of the preconditions learned, we developed two concept hierarchies for each domain, a well-structured hierarchy and an ill-structured hierarchy. The well-structured concept hierarchy directly reflects the hierarchical structure of skill knowledge, which means that the preconditions of the target skills appear directly in the definition of the goal concepts as subconcepts. For the ill-structured hierarchy, we removed this property from some concepts in the well-structured hierarchy.

We evaluated the quality of the methods acquired with each concept hierarchy using four performance metrics. The first measurement tests the generality of the preconditions using recall, defined here as the percentage of problems for which the system can directly retrieve a skill, regardless of its correctness. A second measurement evaluates the specificity of the preconditions based on precision, defined analogously as the percentage of the problems that the system can solve using the first retrieved skill among all the problems for which the system can retrieve some skill. Thirdly, success rate measures the percentage of the problems that can be solved using only known skills and without applying an incorrect skill. This reveals how well the system strikes the balance between generality and specificity. Finally, the average number of cycles the needed to solve one problem provides insight into speedup associated with the improved learning methods. Other metrics such as CPU time are not reported since they are secondary to the goals of this work. All of the reported results are an average over three runs with one hundred randomly generated problems in each run.

In addition to the conditions described above, we also attempted to compare our approach with a system similar to Hogg et al.'s (2008) algorithm. The system constructs preconditions using the same combining method that we use in computing the specialized start condition, except without introducing new predicates. In practice, the rules learned without new predicates becomes very specific, which prevents this approach from finishing most of the tests. Since this result is not a direct evidence for the robustness of our approach, we did not report these results here.

We also wanted to do a comparison with Ilghami et al.'s (2002) and Nejati et al.'s (2006) work. However, their systems require annotated traces as input whereas our system does not. We could not find a reasonable way to compare our system with them experimentally. Therefore, no results are reported here. Besides, the proposed concept learner is integrated into a unified system, where the learning process interleaves with problem solving. Traditional planners, on the other hand, are stand-alone modules, which do not contain learning components. Hence, we did not compare our system to traditional planners.

Freecell Solitaire

The objective in testing the systems on freecell is to determine the robustness of both the concept learner and the skill learner in acquiring useful knowledge for fixed domain sizes. We provided the system with two initial knowledge bases. The well-structured (good) conceptual knowledge base contains 31 concepts, while the ill-structured (bad) concept hierarchy includes 33 concepts. The two hierarchies differ in seven concepts. Both knowledge bases contain the same 21 primitive skills. These are sufficient to solve only problems that are one step away from the goal. We tested the system on problems with eight, 12, 16, 20 and 24 cards. Figure 3 displays the results with both the no concept learning approach and the concept learning approach.

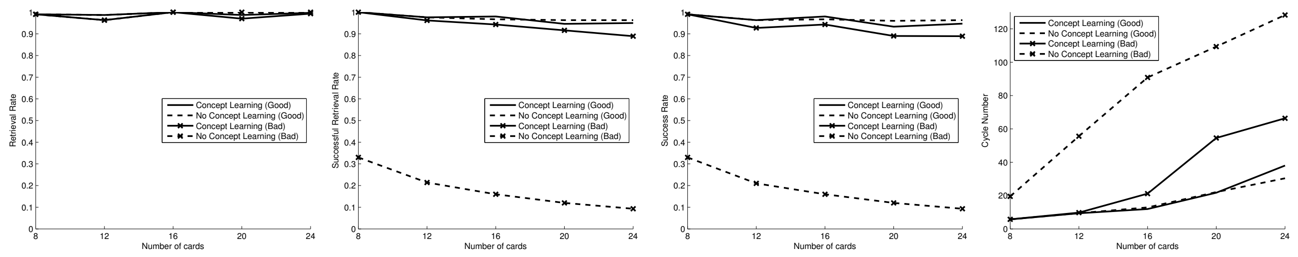
The results show that with the good concept hierarchy, both systems performed well. All of the final success rates with both learners are high. Even with 24 cards, the final success rates with both learners are still above 93%. With the ill-structured concept hierarchy, the performance of the no concept learning approach drops drastically to around 10% with 24 cards, while the concept learning approach is still able to maintain a success rate of 88%. Inspection reveals that although both approaches maintain high retrieval rates, the successful retrieval rates of the no concept learning approach become very low with the bad concept hierarchy, while the successful retrieval rates of the concept learning approach remain to be 90% with the bad hierarchy. This is an indication that the methods acquired by the no concept learning system have overly general preconditions. And by learning concepts for precondition, our approach mitigated this problem and acquired methods that have maintained the appropriate level of generality.

Lastly, the average number of cycles used to achieve goals is significantly smaller with the concept learning approach comparing with the no concept learning approach. Analysis of individual runs shows that since the methods acquired by the no concept learning approach have over general start conditions, the system sometimes picks a skill to achieve the goal at the very beginning, but fails to solve the problem later. Then the system has to switch to other applicable methods to achieve the goal. This causes the more steps. Given that these tests are performed with the same number of cards as used while learning, we can conclude that the preconditions acquired by the concept learner substantially decrease the chance of selecting irrelevant skills.

Logistics Planning

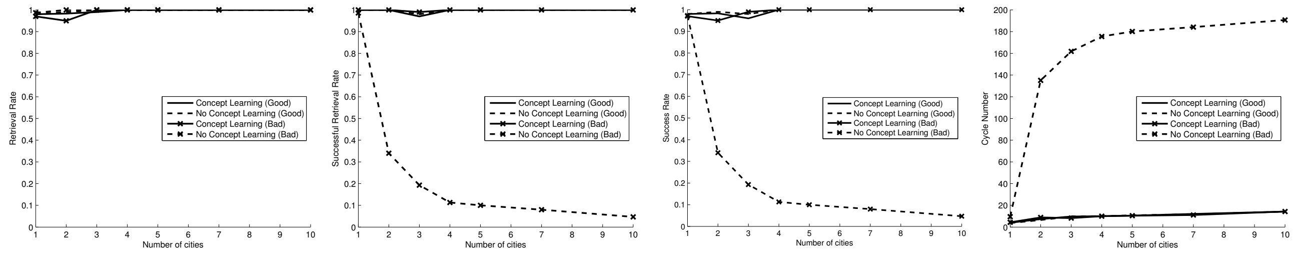
The goal of the logistics planning domain is to deliver packages to destinations using trucks and planes. Trucks can only move inside cities, while planes can fly among cities. Agents are provided with a map detailing the routes among cities to the agent. Our objective in using this domain is to test the ability of learned concepts and skills to generalize among problems of different sizes with ill-structured knowledge bases. We provided the system with two initial knowledge bases. The well-structured (good) conceptual knowledge base contains 17 concepts, while the ill-structured (bad) concept hierarchy includes nine concepts. Only seven concepts are the same in both hierarchies.

The skill knowledge bases are also different. The well-structured (good) skill knowledge base has seven primitive skills. The ill-structured (bad) skill knowledge base includes four primitive skills. These were sufficient to solve problems which only require one action. The system was given seven maps with one, two, three, four, five, seven and ten cities respectively. For each map, we randomly generated 100 problems. Then, the system was tested on the seven hundred problems with cities of increasing sizes. Thus, knowledge acquired in earlier test problems was carried over into later problems, even when the map and number of cities have changed. Figure 4 displays the results for both systems.



(a) (b) (c) (d)

Figure 3: Results from freecell with varying numbers of cards: (a) recall, (b) precision, (c) success rates, (d) cycle count.



(a) (b) (c) (d)

Figure 4: Results in Logistics with different numbers of cities: (a) recall, (b) precision, (c) success rates, (d) cycle count.

With both good and bad concept hierarchies, in the concept learning condition, the system rapidly acquired recursive structures in concepts and skills, thereby increasing the generality of the learned skills. The system solved around 99% of the problems in the three-city map. It then successfully solved all the problems in maps of larger cities using only skills acquired in the three-city map. This indicates the generality gained from the three-city case. The original skill learning algorithm also acquired applicable skills, but the start conditions were again too general with the bad concept hierarchy, causing inapplicable skills to be retrieved. The average number of cycles used in achieving the goals suggested that the concept learning approach returned plans with smaller number of steps than the no concept learning approach given the bad knowledge base.

To sum up, with well-structured concept hierarchies, both approaches were able to perform well, and generalized to problems of larger sizes. With ill-structured knowledge bases, our concept learning approach outperformed the no concept learning system by improving the execution performance and by specializing skills enough to reduce irrelevant retrievals, while maintaining sufficient generality. We conclude that our approach is more robust to different representations than the one without concept learning.

Related Research

The main claim of this paper is that hierarchical preconditions learned through successful problem solving lets an agent choose among its acquired skills more effectively. Although there has been considerable work on representation change (e.g. Muggleton and Buntine 1988, Martín and Geffner 2004) in machine learning, little has occurred in the context of problem solving and execution. One exception comes from Utgoff (1984), whose system introduces unary predicates, like even and odd numbers, to describe situations in which learned strategies solve problems. Similarly, Fawcett's (1996) work generates new relational predicates for heuristic state evaluation by analyzing operator preconditions and transforming the results. Our approach differs from both efforts by supporting incremental learning interleaved with problem solving.

Our framework also incorporates ideas from other research on analytical learning that does not address representation change, such as explanation-based learning (Segre 1987) and macro-operator formation (Iba 1989). One key difference is that our approach transforms the explanation structure that results from analysis into a concept hierarchy, rather than dispensing with the structure.

Ilghami et al.'s (2002) work also focuses on learning preconditions by analyzing successful traces, and by applying a well-known machine learning algorithm, *candidate elimination* to construct preconditions. However, the preconditions learned are not embedded hierarchically, which causes potential lose of flexibility. Shavlik's (1990) system achieves similar effects by learning recursive plans stated as decomposition rules. Work on relational reinforcement learning, such as by Dzeroski et al. (2001) uses first-order representations to provide effective abstraction in learning Q value. Our research also has much in common with other efforts on learning teleoreactive logic programs (Langley and Choi 2006; Nejati, Langley, and Konik 2006; Hogg, Muñoz-Avila, and Aha 2008). All of these systems differ from our approach in that they operate over a fixed set of predicates.

Concluding Remarks

The promising results shown above form a foundation for extensions of this work along several dimensions. In order to better demonstrate and understand the quality of the learned preconditions, one possible future step is to carry out more extensive experiments. Moreover, in order to measure the improvement gained from the precondition learner, allowing the problem solver to chain backward through learned skills and concepts is another interesting extension. Importantly, improving the problem solver is one of the essential extensions in further improving the proposed system, as the problem solver remains a critical bottleneck with respect to both learning and performance. We believe that our precondition learner extends naturally in these directions.

In summary, we motivated the need for building learners robust to representation change by acquiring concpetual predicates during learning. To demonstrate this idea, we introduced a precondition learning approach that defines new conceptual predicates to remove reliance on the quality of the given conceptual knowledge, and interleaves with skill learning. We showed how the mechanism learns hierarchically organized conceptual predicates, including disjunctive and recursive concepts, to construct preconditions that balance generality with specificity. We also demonstrated how these preconditions improve execution performance with both an example and experimental evaluations comparing with learners on a fixed representation. In particular, we showed how our precondition learner improved the ability of teleoreactive logic programs to represent domain knowledge, and how an agent could leverage this new knowledge to construct preconditions that improve learning performance and execution. From these experiments, we conclude that skill learners with appropriate acquisition of conceptual knowledge during the course of learning are more robust than learners operate on a fixed domain representation across various domain representations.

Acknowledgements

The authors would like to thank Dongkyu Choi and Tolga Konik for helpful discussion concerning this work. This material is based on research sponsored by DARPA under agreement FA8750-05-2-0283 and by ONR under agreement N00014-08-1-0069. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed on implied, of DARPA, ONR or the U. S. Government.

References

Anderson, J. R. 1993. *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Bacchus, F. 2001. AIPS '00 planning competition. *AI Magazine* 22:47–56.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43(1–2):7–52.

Fawcett, T. 1996. Knowledge-based feature discovery for evaluation functions. *Computational Intelligence* 12(1).

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Hogg, C.; Muñoz-Avila, H.; and Aha, D. W. 2008. Htn-maker: Learning htns with minimal additional knowledge engineering required. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*. Chicago: AAAI Press.

Iba, G. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3:285–317.

Ilghami, O.; Nau, D. S.; Muñoz-Avila, H.; and Aha, D. W. 2002. Camel: Learning method preconditions for HTN planning. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, 131–141. Toulouse, France: AAAI Press.

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

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

Martín, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20:9–19.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42(2-3):363–391.

Mooney, R. J. 1990. *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding*. San Mateo, CA: Morgan Kaufmann.

Muggleton, S., and Buntine, W. 1988. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, 339–352. Morgan Kaufmann.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. Shop: A simple hierarchical ordered planner. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 968–973.

Nejati, N.; Langley, P. W.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23nd International Conference on Machine Learning*. Pittsburgh, PA: ACM.

Nilsson, N. 1994. Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research* 1:139–158.

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

Shavlik, J. 1990. Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning* 5:39–70.

Utgoff, P. E. 1984. *Shift of Bias for Inductive Concept Learning*. Ph.D. Dissertation, Department of Computer Science, Rutgers University, New Brunswick, NJ.

Wilkins, D., and des Jardins, M. 2001. A call for knowledge-based planning. *AI Magazine* 99–115.