Advances in Cognitive Systems 1 (2012) 109–126 Submitted 11/2011; published 12/2012

Improving Acquisition of Teleoreactive Logic Programs through Representation Extension

Nan Li[1] NLI1@CS.CMU.EDU
David J. Stracuzzi[2] DAVID.STRACUZZI@GMAIL.COM
Pat Langley[3] LANGLEY@ASU.EDU
Computing Science and Engineering, Arizona State University, Tempe, AZ 85281 USA

Abstract

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

1. Introduction

Hierarchical task networks (Nau et al., 1999; Wilkins & 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 research on learning such structures, including work by Ilghami et al. (2002), Langley and Choi (2006), and Nejati et al. (2006). The latter two represent the environment with a conceptual hierarchy that provides a vocabulary for describing situations at different levels of abstraction. Their learners then use this vocabulary to guide the acquisition of precondition and subtasks for new hierarchical skills. A good representation of the world reveals its essential features and helps the system decide which skills to apply in solving problems.

However, such a representation is often unavailable. If the organization of the concept hierarchy does not reveal critical features that correspond to the task/subtask hierarchy, then the methods or skills acquired may not guide the system to achieve its desired goals. We refer to such concept hierarchies as *ill structured*. However, manually coding *well-structured* representations often requires time and domain expertise. Ideally, a skill learning method should automatically revise and extend its conceptual knowledge to better reflect the target task/subtask hierarchy.

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.
3. Now with the Silicon Valley Campus, Carnegie Mellon University, CA, and University of Auckland, New Zealand.

In this paper, we consider the problem of learning *teleoreactive logic programs*, a special case of hierarchical task networks, in the presence of ill-structured concept hierarchies. In our experience, skills acquired from an ill-structured hierarchy often have inaccurate preconditions for their application. Two factors govern the accuracy of skills' preconditions. First, they must be sufficiently general to let the skills both reproduce the original solutions and carry over to unfamiliar but relevant contexts. Second, the preconditions must be sufficiently specific to prevent the selection and application of learned skills that cannot achieve the current goal. In the absence of concepts that describe preconditions effectively, a system may acquire skills that are overly general or overly specific, which in turn reduce its ability to solve new problems.

We present an approach that responds to this challenge by constructing appropriate concepts and adding them to the hierarchy. Our mechansim analyzes the structure of solved problems, which it then uses to define both skills for reproducing the solutions and conceptual predicates that serve as these skills' preconditions and effects. The approach differs from most other algorithms for learning hierarchical task networks in that the preconditions acquired are no longer simple conjunctions of primitive predicates, but are instead more complex concepts that describe high-level abstractions about the world. The new structures expand the conceptual and procedural knowledge base, and they can be used to guide learning on future tasks. Experimental results show that, although the mechanism increases run time for learning, it aids teleoreactive agents in acquiring correct skill knowledge. More specifically, a teleoreactive system with concept learning can acquire accurate and useful skill knowledge even in the presence of bad representations, while a system that lacks this ability acquires inaccurate skills.

2. 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 & 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 "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 cognitive architecture for physical agents. This framework shares many features with architectures like Soar (Laird et al., 1986), and ACT-R (Anderson, 1993), and PRODIGY (Minton, 1990). Our studies of teleoreactive logic programs have utilized ICARUS, 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, 40 concepts and 13 primitive skills are initially provided to the agent.

Table 1. Sample conceptual clauses from FreeCell solitaire.

```
;; cards ?c1 and ?c2 are of different color, and 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)))
```

2.1 Knowledge Representation

ICARUS distinguishes between conceptual and procedural knowledge, and it has separate knowledge bases for each. The conceptual knowledge base stores a hierarchy of first-order Horn clauses with negation that provides a vocabulary for describing the 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, which describes the conditions under which the concept is true. Concepts may refer to either lower-level concepts or to *percepts* that the agent observes in the environment. Table 1 presents some sample concepts. For instance, *(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*.

The procedural knowledge base stores skills that can execute in the world in a form similar to hierarchical STRIPS operators (Fikes & 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. Skills have either an action field that refers to actions the agent can execute directly in the world, such as *(clear-and-put-on ?cb ?c ?dc)* in Table 2, or a subgoals field that specifies subgoals the agent should achieve, such as *(put-on ?c ?dc)* in Table 2. A skill (e.g., *column-to-column-able*) may refer to itself, either directly or through a subgoal, letting the framework support recursive programs.

2.2 Teleoreactive Execution and Problem Solving

Teleoreactive logic programs perform a sequence of operations to execute their skills in a goal-directed but reactive manner. ICARUS provides one such interpreter for teleoreactive logic programs that operates in discrete cognitive cycles, as Figure 1 depicts. On each cycle, the interpreter infers beliefs about its environment, after which its selects a primitive, executable skill that appears relevant to achieving its current goals, which it then executes in the world.

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-not-on_s8 ?c ?dc)))
```

To understand its environment, the interpreter first receives a set of percepts in the form of ground literals. The architecture then infers beliefs about its situation 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 a belief memory for use in later decision making. Based on the inferred state of its surroundings, ICARUS uses its skill knowledge to take action towards achieving its goals. To determine its next move, the interpreter first retrieves an unsatisfied goal and then chooses a skill that it expects to achieve the goal. By selecting skills according to both the current state and the current 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 that decomposes the current goal into subgoals. ICARUS uses both conceptual and procedural knowledge to chain backward until it finds a subgoal associated with an executable skill, preferring to chain through skills over concepts. In the former case, the problem solver retrieves a skill that contains the goal in its head, pushes its start condition onto a goal stack, and begins working on this subgoal. In the latter case, the system uses the concept definition to decompose the goal into multiple subgoals. If more than one subgoal is unsatisfied, the problem solver selects one subgoal at random to push onto the goal stack. Once it achieves the subgoal, the interpreter pops it from the goal stack, and works on other subgoals until it achieves the parent goal. During this process, execution resumes upon finding an applicable skill, thereby interleaving problem solving and control.

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 instantiated start condition on the stack as a subgoal.

1. Note that this differs from Prolog, which evaluates predicates in a top-down, query-driven manner.

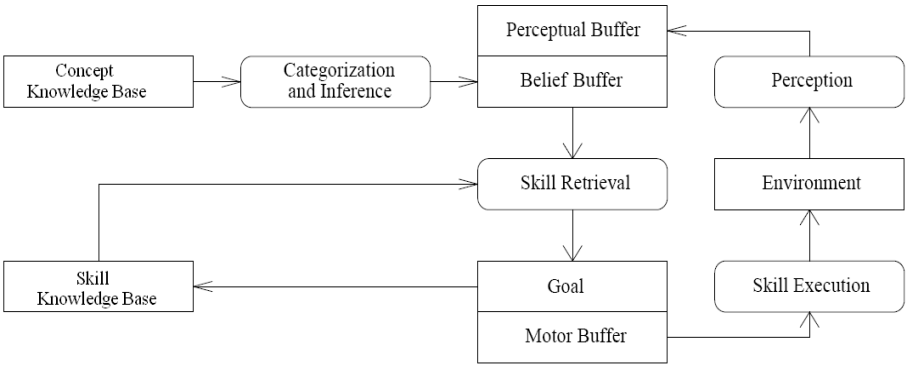

Figure 1. System diagram of the ICARUS architecture.

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's definition. Of these, only *(clear* 3♠*)* is unsatisfied, so ICARUS adds it to the stack as the next subgoal. An applicable skill, *(clear* 3♠*)*, exists for this subgoal and the architecture executes it immediately. This changes the state such that A♡ is now on 2♠, which makes *(column-to-column-able* 2♡ 3♠ 4♠*)* true. Now the skill *(clear* 4♠*)* becomes applicable, so the system carries out the skill and achieves its initial goal, *(clear* 4♠*)*. Note that problem solving typically involves extensive search, although the example does not illustrate it.

2.3 Skill Learning

As Langley and Choi (2006) describe, when the ICARUS problem solver achieves a goal or subgoal, it constructs a skill for this goal. The head is a generalized version of the goal in which constants have been replaced by variables. Details of the new skill's body depend on whether the problem solver achieved the goal by chaining through a skill or through a defined concept.

If the problem solver achieved the goal by chaining through a skill, then it must first have satisfied that skill's start condition and executed it. Therefore, the first subgoal of the new skill should be the instantiated start condition for the chained skill, followed by the skill's subgoals in their order of achievement. Because the first subgoal of the new skill ensures applicability of the subsequent subgoals, the start condition of the skill that achieved the first subgoal becomes the start condition of the new skill. For example, in our FreeCell example, since *(clear* 4♠*)* was achieved by chaining through skill *(clear-and-put-on* 4♠ 2♡ 3♠*)*, the ICARUS learning module constructs a new skill with two subgoals, *(column-to-column-able* 2♡ 3♠ 4♠*)* and *(clear-and-put-on* 4♠ 2♡ 3♠*)*.

If the problem solver instead achieved the goal through concept chaining, then it must have satisfied all subconcepts of the chained concept. As a result, the subgoals of the new skill should be the unsatisfied subconcepts of this concept in their order of achievement. The start condition of the new skill should be all 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.

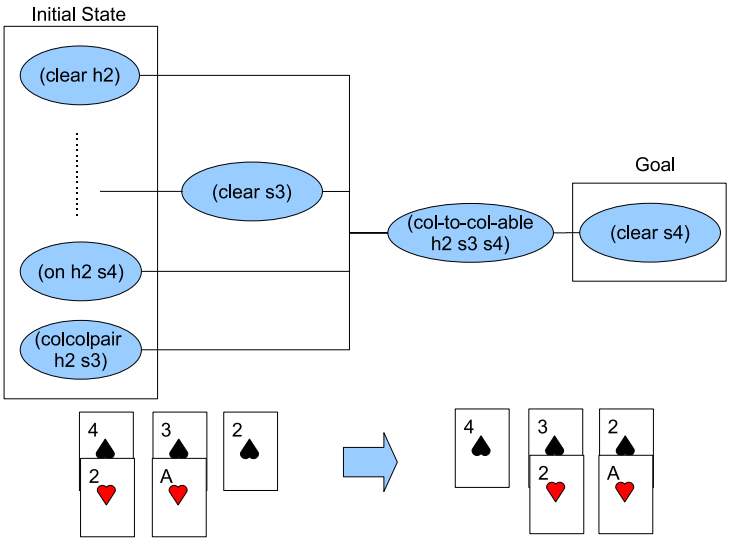

Figure 2. A FreeCell problem-solving example. Concepts are shown as circles, and procedural clauses are shown as rectangles.

In subsequent runs, if the ICARUS agent receives the same goal (potentially 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.

3. Representing, Using, and Learning Conceptual Predicates

Now that we have reviewed one approach to learning teleoreactive logic programs from traces of problem solving, we can examine drawbacks of this method and whether we can improve upon it. Although Langley and Choi reported encouraging results with their mechanism, we have found that the start conditions it acquires are sometimes overly general, while other times they are overly specific. Moreover, our experience suggests that both problems can become substantially worse in the presence of ill-structured concept hierarchies.

In particular, if the problem solver achieved the goal through skill chaining, the learned start condition is determined by the specific skills used to achieve the first subgoal. Other skills that might have achieved the subgoal are not considered, thus limiting the condition's generality. Suppose there were ten skills available in achieving the first subgoal. In order to learn the start condition that covers them all, the method must chain backward through the skill ten times under ten different situations, which would slow learning substantially. Moreover, the start condition of the new skill may not be a simple conjunction of primitive predicates, but rather a hierarchical concept that results from a sequence of problem-solving experiences. Since the mechanism operates over a fixed representation, this can cause it to acquire ineffective preconditions, especially when given ill-structured background knowledge. An improved approach would automatically revise the conceptual hierarchy to produce more appropriate preconditions.

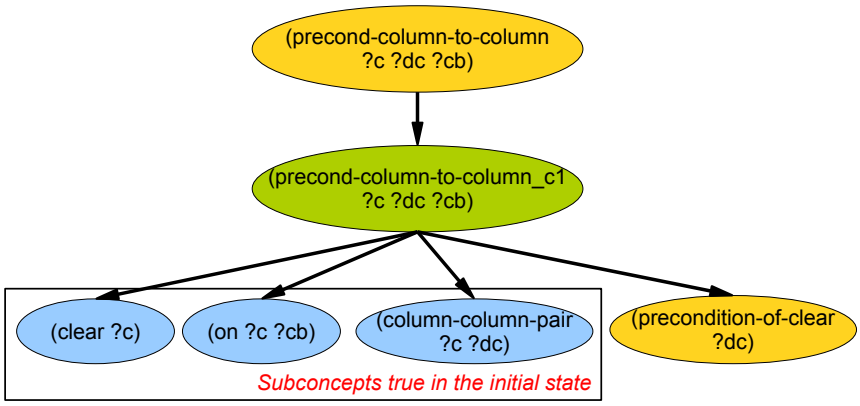

Figure 3. An example concept hierarchy acquired by the concept learner. Given concepts are shown as blue circles, generalized concepts are shown as yellow circles, and specialized concept are shown as green circles.

Similarly, if the problem solver achieved the goal through concept chaining, Langley and Choi's method can create skills with overly general start conditions, since they do not contain information about start conditions of skills that achieved the subgoals. At one extreme, given an ill-structured concept hierarchy and no subconcepts that were initially satisfied during chaining, the learned skill would have an empty start condition, which implies that it is always applicable. This overly general condition could cause the skill to execute under conditions in which it cannot achieve the goal because some essential aspect of the state is missing.

3.1 Representing and Using Conceptual Predicates

Our new approach to skill acquisition attempts to overcome these issues by altering the conceptual hierarchy. As before, the system constructs a new skill whenever it achieves a goal during problem solving. However, it also defines new predicates that it uses to denote the skill's preconditions and effects. Introducing new concepts for these abstractions expands the system's knowledge about situations that can arise in the world. They also let the system acquire additional skills that achieve subgoals stated in terms of these concepts and thus provide broader coverage.

As explained above, specifying learned skills in terms of the original predicates can lead to preconditions that are overly general or overly specific. The new system utilizes two kinds of conceptual predicates that strike a balance between specificity and generality. The first are *specialized predicates* for preconditions and effects, which it uses during the execution process to select appropriate skills. One specialized predicate describes the class of situations in which a skill should be applied, whereas another specialized predicate describes its effects. These predicates can be highly specific, since they are meant to ensure the selected skill will achieve its associated goal.

However, the system also encodes *generalized predicates*, which it uses to offset the specificity of skill preconditions and effects. During learning, the system cannot predict the situations in which a new skill will be used to achieve the given goal. In response, it uses generalized predicates to describe situations in which at least one skill can be applied to achieve the goal and the situations that result. Each generalized condition predicate is defined as a disjunction over all specialized condi-

Table 3. Sample specialized concepts learned for FreeCell solitaire.

```
;; 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)))
```

tions for skills that achieve this goal; each generalized effect predicate is defined as a disjunction of their specialized effects. Since generalized predicates are not associated with any particular skill for achieving a desired goal *G*, the learner uses them when determining the preconditions and effects of new higher-level skills that incorporate *G* as a subgoal.

3.2 Learning Conceptual Predicates

We can now consider how the two types of precondition and effect predicates are learned. Suppose a goal is achieved by chaining through a concept definition. Then the system creates a specialized conceptual predicate for the start condition of the new skill S_{new} by combining the generalized condition and effect predicates associated with the subgoals, $S_1, \ldots, S_n$, using a procedure similar to that for finding conditions during macro-operator composition (Mooney, 1990). This produces a conjunction of the conditions associated with each subgoal minus those made true by the effects of subgoals earlier in the sequence. The system uses the same procedure to generate a specialized concept for S_{new}'s effects, which is a conjunction of the effects associated with each subgoal minus those made untrue by effects later in the sequence.

For example, as shown in Table 1, the subgoal *(column-to-column-able* $2\heartsuit$ $3\spadesuit$ $4\spadesuit$*)* is achieved by chaining through its concept definition. The subgoal of the new skill is *(clear* $3\spadesuit$*)*. The effect field of the new skill is initialized to *(clear* $2\heartsuit$*)*, *(on* $2\heartsuit$ $4\spadesuit$*)*, and *(column-column-pair* $2\heartsuit$ $3\spadesuit$*)*, which represents the initially satisfied subconcepts of the chained concept. Next, as shown in Figure 3, the precondition learner computes a specialized concept for the new skill's start condition. This consists of *(clear* $2\heartsuit$*)*, *(on* $2\heartsuit$ $4\spadesuit$*)*, and *(column-column-pair* $2\heartsuit$ $3\spadesuit$*)*, which are the subconcepts that held in the initial state. The system also adds to the concept *(precondition-of-clear* $3\spadesuit$*)*, the generalized predicate for the only subgoal, *(clear* $3\spadesuit$*)*. After replacing constants with variables, the learner adds the new concept to its knowledge base. In addition, the system creates a specialized concept defined initially as the new skill's effects. In this case, none of the effects involve deletion, but several new literals become true and are added to the effect concept's definition. The result is the specialized effect predicate, *effect-of-column-to-column-able_c1*, shown in Table 3.

Note that although the only subgoal, *(clear* $3\spadesuit$*)*, was achieved by executing a specific skill for it, the system still uses the generalized start condition *(precondition-of-clear* $3\spadesuit$*)* in the new skill, rather than adding the specialized predicate associated with the executed skill. By using the

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-not-on ?cb ?c ?dc))
  :effects    ((clear ?cb)  (not-on ?c ?dc)  (clear ?c)))
```

generalized predicate in the new precondition, the learned skill will be applicable both when the specialized one is executable and when other skills that can achieve the same subgoal are applicable. After creating the specialized concept for the goal, the system updates the corresponding generalized condition predicate accordingly.

In contrast, if the system achieved the goal by chaining through a skill, the specialized precondition for the new skill is the generalized predicate for the skill that achieved the first subgoal, S_1. Because the system uses the generalized predicate rather than the specialized one, the new skill's precondition will generalize to all situations in which at least one applicable skill will achieve the first subgoal. The start condition of the chained skill is not included because its applicability only depends on S_1. The specialized effect for the learned skill is set to the original skill's effect. Because the specialized precondition and effect are the same as some existing precondition and effect, the interpreter does not introduce new predicates for them.

For instance, since the system achieved the goal *(clear* $4\spadesuit$*)* through skill chaining, it constructs a new skill with *(column-to-column-able* $2\heartsuit$ $3\spadesuit$ $4\spadesuit$*)* and *(clear-and-put-on* $4\spadesuit$ $2\heartsuit$ $3\spadesuit$*)* as subgoals. Table 3 presents the definitions for concepts associated with these skills, while Table 4 shows the skill itself. The specialized start condition of this skill is simply the generalized predicate associated with the first subgoal, *(column-to-column-able* $2\heartsuit$ $3\spadesuit$ $4\spadesuit$*)*. The specialized effect for the new skill is the specialized effect of the skill through which chaining occurred. Having constructed the specialized conditions and effects, the system updates the generalized condition and effect concepts accordingly. Notice that recursive structures occur in the learned preconditions; this stems from the recursive nature of the problem solver, which repeatedly pushes goals onto a stack and attempts to solve them. Also note that learned concepts are eligible for chaining during problem solving, which lets the system incorporate them into skills learned later.

Recall that a generalized condition or effect predicate is a disjunction over all corresponding specialized conditions or effects. After determining a skill's specialized conditions and effects, the system updates definitions for their corresponding generalized concepts by adding new rules with

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 ?dc2)  (card ?dc))
  :relations   ((precond-column-to-column ?c ?dc2 ?dc)))

;; 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)))
```

the same predicate P in their heads, which serve as disjunctive clauses in P's definition. Even if the system introduces no new predicate for skills learned through skill chaining, it still updates the generalized condition or effect concept for the associated goal. This encodes knowledge that it can achieve the goal in a new situation, letting the learned skills reach this objective in novel contexts. For example, consider a skill that contains two subgoals, S_1 and S_2. The skill's start condition combines the generalized condition predicates, C_1 and C_2, associated with the two subgoals. When the system learns a skill related to one of these subgoals, the generalized predicate associated with that subgoal will become more general, as will the start condition for the skill. Similarly, adding a disjunctive rule for a generalized effect predicate increases the coverage of any skill that refers to it. Table 5 shows some generalized condition and effect predicates for our FreeCell example.

3.3 Extending the Skill Learner

The system uses the newly defined predicates to specify the learned skill. The specialized precondition concepts acquired in this manner are more specific than the start conditions produced by Langley and Choi's approach. This helps to prevent the system from retrieving inappropriate skills during execution, which would take it down fruitless paths. The extended learning mechanism also uses the specialized effect predicates to specify new skills' effects. In addition, the earlier system only learned through skill chaining when this process occurred through primitive skills that referred to executable actions. The augmented system can also learn from successful chaining through non-primitive skills that it has acquired from earlier problem-solving experiences, although we turned this ability off in our experimental studies. To this end, the learned skill includes the start condition of the chained skill as its first subgoal. Table 4 shows some skills acquired for the FreeCell example.

4. Experimental Evaluation

To evaluate the robustness of our concept learner, we carried out experiments in two domains that benefit from rich conceptual knowledge. Logistics planning is a classic task domain used in the literature that is described naturally in terms of recursive concepts. FreeCell solitaire, which also supports rich knowledge about card patterns, has been used in planning competitions (Bacchus, 2001). Our aim was to show that an extended system incorporating the concept learner can acquire procedural knowledge efficiently and that its behavior is relatively insensitive to the quality of concept hierarchy provided as background knowledge.

4.1 Experimental Design

We compared our new approach to learning precondition on skills with the method reported by Langley and Choi (2006). As noted earlier, this earlier technique also acquires hierarchical skills from problem solving, but it does not construct new conceptual knowledge for use in their preconditions. Since both methods learn incrementally during the process of problem solving, we designed our experiments so that the performance system invokes the problem solver whenever it fails to achieve a goal, in the same manner as Nejati et al. (2006). When it achieves the top-level goal using exactly one top-level skill (i.e., all skills selected achieve their goals) and without calling the problem solver, we consider the problem solved. For each problem, the system reports whether it has solved the task successfully within 200 cycles or it has failed.

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, one well structured and the other ill structured. The former directly reflected the hierarchical structure of skill knowledge, which means that the preconditions of the target skills appeared directly in the definition of the goal concepts as subconcepts. For the ill-structured hierarchy, we retained some concepts in the well-structured version but removed some others.

We evaluated the quality of the methods acquired with each concept hierarchy using four performance metrics. The first measure involves the generality of conditions using *retrieval rate*, defined here as the percentage of problems for which the system directly retrieves a skill, regardless of its correctness. A second metric concerns the specificity of conditions based on *successful retrieval rate*, defined as the percentage of problems the system solves using the first retrieved skill among all the problems for which it retrieves some skill. *Success rate* measures the percentage of problems solved using only known skills and without applying an incorrect skill, reflecting a balance between generality and specificity. Finally, the *number of cycles* needed to solve a problem provides insight into the benefits of learning. Although not our focus here, we also recorded CPU time.

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. Their approach constructs preconditions using an analytical method similar to the one described by Mooney (1990). In practice, rules learned in this manner become very specific and complex. For example, in the FreeCell domain, acquired start conditions describe conditions on cards relevant to the concept. In a 16-card FreeCell test, one rule referred to eight different cards, which leaves little room for generalization. Moreover, the number of arguments associated with these rules creates a utility problem in the matching process.

Suppose that the environment contains 16 cards and that a rule refers to eight of them. There are approximately 16^8 ways to match this condition, which prevented the system from finishing most tests. Although this supports our arguments for acquiring new concepts for use in preconditions, we cannot draw any stronger conclusions from these failures.

We also wanted to compare our method with those reported by Ilghami et al.'s (2002) and Nejati et al.'s (2006). However, both systems learned from worked out solutions to training problems, whereas our system learns from traces of its own solutions, and we could not find a reasonable way to compare them experimentally. Moreover, since our goal is to develop ways to improve the acquisition of hierarchical skills for use in knowledge-based planning and execution, our studies did not include any systems that generate plans from primitive operators. There already exists ample evidence that knowledge about task structure can produce more efficient planning; we are concerned with how to acquire that content effectively.

4.2 FreeCell Solitaire

Our objective in testing the two methods on FreeCell is to determine the robustness of skill acquisition with and without concept learning in constructing useful knowledge for fixed domain sizes. To this end, we provided the systems with two distinct knowledge bases. The well-structured (good) conceptual knowledge base contained 31 concepts, while the ill-structured (bad) concept hierarchy included 33 concepts. The two hierarchies differed in seven of their concepts, but the knowledge bases shared 21 basic skills that were sufficient to solve problems only one step away from the goal. We tested both systems on problems with eight, 12, 16, 20, and 24 cards, in each case running them on three different sequences of 100 randomly generated tasks and averaging the results across these sequences. Figure 4 displays the results with and without concept learning.

The graphs show that, with the good concept hierarchy, both systems performed well, with each of the final success rates being high. Even with 24 cards, the final success rates with both learners are still above 93%. With the ill-structured concept hierarchy, the performance without concept learning dropped drastically to around 10% with 24 cards, while the version with concept learning maintained a success rate of 88%. Inspection revealed that, although both approaches maintain high retrieval rates, the successful retrieval rates for the bad concept hierarchy without concept learning became very low, while they remained about 90% with concept learning. This suggests that the skills acquired without concept learning had overly general preconditions. In contrast, acquisition of new conceptual predicates mitigated this problem, letting the system learn skills with appropriate levels of generality that translate into much higher overall success rates. This result was consistent across FreeCell tasks that involved different numbers of cards.

Finally, the average number of cycles used to achieve goals was substantially lower with concept learning than without it. Analysis of individual runs suggested that the skills acquired without concept learning have overly general start conditions, causing the system select skills that, although they appear relevant, later fail to solve the problem. In response, the system must switch to other applicable skills in its efforts to achieve the goal, which leads to more steps. Given that these tests involve the same number of cards as used during learning, we can conclude that the preconditions acquired with the concept learner substantially decrease the chances of selecting inappropriate skills, which in turn increases the overall success rate greatly.

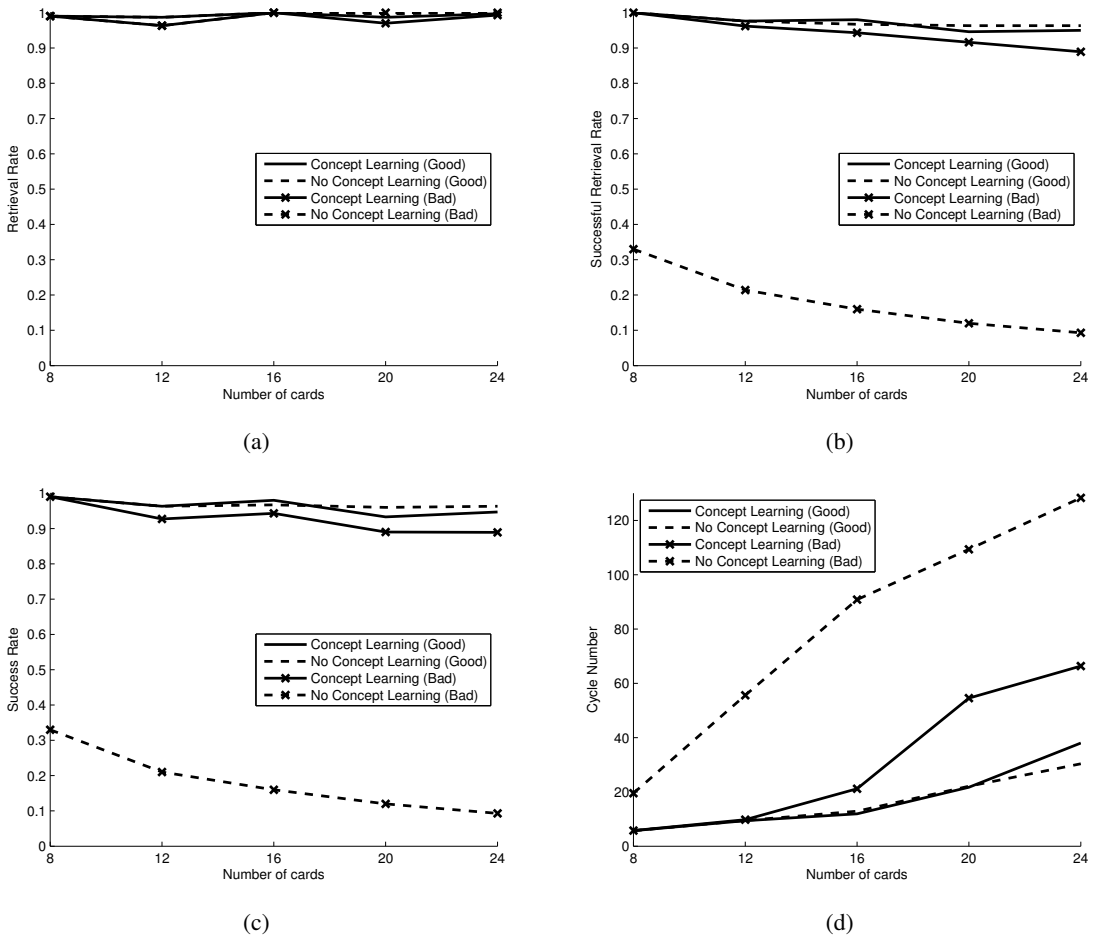

Figure 4. Results for FreeCell solitaire with varying numbers of cards: (a) retrieval rate, (b) successful retrieval rate, (c) success rate, and (d) number of cycles.

4.3 Logistics Planning

Tasks in the logistics planning domain involve delivery of 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 was 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) knowledge base contained 17 concepts, while the ill-structured (bad) hierarchy included nine concepts. Only seven concepts were the same in both hierarchies.

The initial knowledge about skills was also different. The well-structured knowledge had seven primitive skills, whereas the ill-structured knowledge included only four. These were sufficient to solve problems that required only one action. We generated 100 problems each for seven maps with one, two, three, four, five, seven, and ten cities, respectively. We presented both systems with all

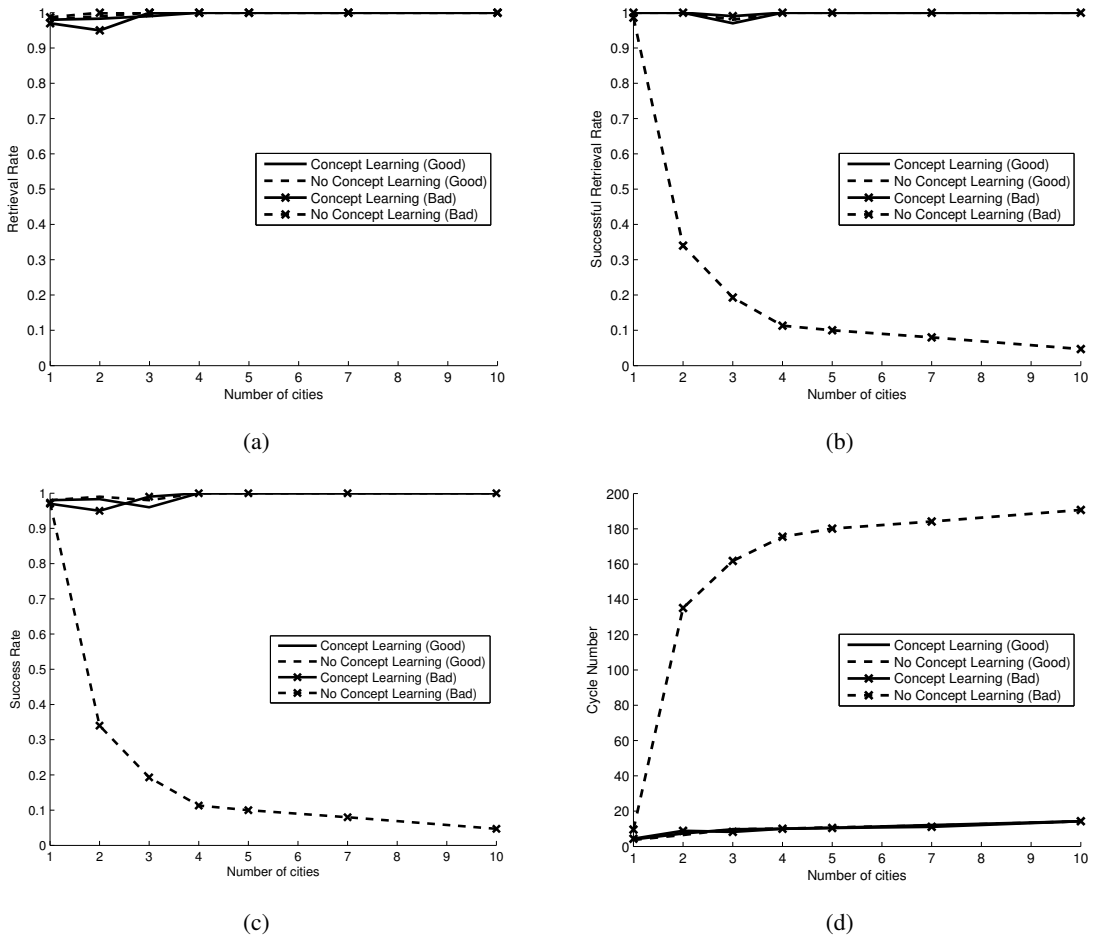

Figure 5. Results for logistics planning with different numbers of cities: (a) retrieval rate, (b) successful retrieval rate, (c) success rate, and (d) number of cycles.

700 problems, ordered by maps of increasing size. Knowledge acquired in earlier problems carried over into later problems, even when the number of cities changed. Figure 5 displays the results for both systems, averaged over three different runs with random ordering within each size group.

For both good and bad concept hierarchies, the version with concept learning rapidly acquired recursive concepts, thereby increasing the generality of the learned skills. The system solved around 99% of the problems in the three-city map and then solved all problems in larger cities using only skills acquired in the simpler map. This indicates that, with concept learning, the system generalizes well to cities of larger size. Without concept learning, the system still acquired relevant skills but, given a bad concept hierarchy, in many cases the start conditions were too general, which caused inappropriate retrievals. The average number of cycles needed to solve problems shows that, in the presences of bad background knowledge, concept learning let the performance system achieve its goals with many fewer steps than when this mechanism was absent.

Table 6. Average time (in seconds) spent in solving one problem.

FreeCell (24 cards)	Ill-Structured Hierarchy	Well-Structured Hierarchy	Logistics Planning	Ill-Structured Hierarchy	Well-Structured Hierarchy
No Concept Learning	102,427	426	No Concept Learning	42,005	8,213
Concept Learning	1,927	2,443	Concept Learning	86,449	16,567

4.4 Analysis of CPU Time

We also examined the average time spent in solving problems in the largest environments for both domains. As shown in Table 6, when given well-structured hierarchies, the system that incorporates concept learning spends much more time than the version without it, although both variants had high success rates. Hence, there is clearly some overhead associated with utilizing learned concepts that are unnecessary. Future research should explore ways to detect such redundant concepts and thus to avoid defining them.

Given ill-structured hierarchies for the logistics domain, the system that employs concept learning still takes longer (86,449s vs. 42,005s) than the one without, but, interestingly, this effect is both reversed and much stronger in FreeCell (1,927s vs. 102,427s). This is due to the fact that, without concept learning, the system acquires overly general skills, almost none of them accurate, that apply in many different ways. Furthermore, this system solves many fewer problems than does the one with concept learning, so that measuring CPU time is less meaningful in this situation. Consider a learner that does not acquire any skills and, as a result, always fails rapidly. Such a system would be faster than others but it would not be useful for solving problems.

To sum up, with well-structured concept hierarchies, both approaches performed well and generalized to larger size problems. With ill-structured knowledge, the version with concept learning produced structures that solved substantially more problems without search. This occurred because it acquired skills that were specialized enough to reduce inappropriate retrievals while still general enough to ensure broad applicability. We conclude that augmenting skill acquisition with concept learning enables more robust responses to variations in representations of background knowledge.

5. Discussion

In the previous sections we presented a mechanism for concept learning and demonstrated that it supports robust skill acquisition across different representations. In this section, we discuss some important properties of the learning mechanism. One characteristic is its acquisition of disjunctive concepts that encode generalized preconditions and effects of skills. These include recursive concepts, which result from the recursive nature of the problem solver. However, all created concepts are based on either provided conceptual knowledge or previously learned concepts. As a result, the system can only learn predicates expressable as combinations of concepts available in the ini-

tial knowledge base. At one extreme, if we give the concept learner an empty initial hierarchy, it will not acquire any new conceptual predicates. Another feature of our concept learner is that it only considers new concepts that explain the problem-solving process. Searching the full space is intractable, so this feature of the learning mechanism trades off completeness for efficiency.

As with related methods for skill learning (Langley & Choi, 2006; Nejati et al., 2006), ours simply replaces constants with variables when determining arguments. This design works with domains such as the blocks world and logistics, but will cause problem when specific objects are essential to the problem solution. For example, if, in the logistic domain, only one plane can land at the airport of a city, then simply replacing the constant with a variable in the precondition will overgeneralize: any plane can land at the airport. Future research should consider the automatic detection of such special objects. For instance, the system might acquire an overly general concept first and then specialize it when the skill fails to solve a future problem.

We should also consider the complexity of acquired concepts. The number of specialized predicates is bounded above by the number of learned skills. Likewise, the number of generalized concepts is bounded by the number of top-level and intermediate goals needed to solve the problems. For each concept, let n be the number of objects in the world and let a be the number of arguments in a concept. The number of possible instantiations is $\mathcal{O}(n^a)$, whereas the number of objects in the world is fixed. Hence, increasing the number of arguments in a concept can greatly increase the complexity of matching it, which explains why our version of Hogg et al.'s (2008) approach, which learns overly specific preconditions, can run out of time in experiments. In contrast, concepts acquired by our approach have the same number of arguments as the goal of the skill, which keeps their complexity within reasonable bounds.

6. 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 in machine learning (e.g., Muggleton & Buntine, 1988), 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 that is interleaved with problem solving.

Our framework also incorporates ideas from other research on analytical learning that does not address representation extension, such as explanation-based learning (Segre, 1987) and macro-operator formation (Iba, 1989). One key difference is that our approach transforms the explanation structure produced by the analysis into a concept hierarchy, rather than dispensing with the structure. Shavlik's (1990) BAGGER system also retains some explanatory structure when learning recursive plans stated as decomposition rules, but it does not introduce new conceptual predicates.

Ilghami et al. (2002) report another approach that learns preconditions from successful traces. However, it instead uses the candidate elimination algorithm to learn from multiple training cases, and the acquired preconditions are not organized in a hierarchy, which can restrict their flexibility.

Our research also has much in common with other efforts on learning teleoreactive logic programs (Langley & Choi, 2006; Nejati et al., 2006) and related structures (Hogg et al., 2008). All of these systems differ from our approach in that they operate over a fixed set of predicates.

7. Concluding Remarks

In this paper, we motivated the need for building learners that are robust to differences in representations of background knowledge. In response, we developed an approach that defines new conceptual predicates, some of them disjunctive and recursive, during the process of acquiring teleoreactive logic programs in order to reduce reliance on the original concept hierarchy. For each skill, this method creates two types of predicates that, together, support skills that balance generality with specificity. Specialized concepts are associated with skills and ensure their retrieval in appropriate situations, whereas generalized concepts are associated with goals and ensure access to candidate skills that may achieve them.

We also demonstrated that specifying learned skills in terms of these defined predicates improves their effectiveness at solving novel tasks without resorting to search. More specifically, experiments in two domains revealed that this approach is substantially less dependent on well-structured representations of background knowledge than a more basic method that does not extend its own conceptual predicates. Detailed analyses suggested that the combination of specialized and generalized predicates had the intended effect, guarding against application of skills in inappropriate settings while supporting generalization to new situations.

Although our experiments produced encouraging results, there remain a number of directions in which we can extend our approach to skill acquisition. More comprehensive studies would let us better demonstrate and understand the conditions under which our mechanisms learn effective procedures. We should also evaluate the system's ability, disabled in the experiments reported here, to chain through learned skills and concepts, and thus to learn over them. Intuitively, this should further increase the rate of learning, but whether it has this effect is an empirical question.

In addition, we should explore ways to mitigate the increases in CPU time observed in the experiments. One response is to alter the inference process to generate beliefs selectively, say only ones that are relevant to the goal, rather than operating in its current exhaustive manner. We can also extend the mechanism for concept learning so that it gradually removes unhelpful predicates and thus reduces run time overhead. These augmentations should be straightforward to implement and, taken together, they should lead to even more robust acquisition of conceptual and skill knowledge.

Acknowledgements

The authors would like to thank Dongkyu Choi and Tolga Könik for helpful discussions concerning this research, which was sponsored by DARPA under agreement FA8750-05-2-0283 and by ONR under agreements N00014-08-1-0069 and N00014-10-1-0487. 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.

Fawcett, T. (1996). Knowledge-based feature discovery for evaluation functions. *Computational Intelligence*, *12*, 42–64.

Fikes, R., & 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., & Aha, D. W. (2008). HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*. Chicago: AAAI Press.

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

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

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

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

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 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., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 339–352). Morgan Kaufmann.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: A simple hierarchical ordered planner. *Proceedings of the International Joint Conferences on Artificial Intelligence* (pp. 968–973). Stockholm.

Nejati, N., Langley, P., & Könik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the Twenty-Third International Conference on Machine Learning*. Pittsburgh.

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. *Proceedings of the Third IEEE Conference on AI for Applications* (pp. 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*. Doctoral dissertation, Department of Computer Science, Rutgers University, New Brunswick, NJ.

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