

Transfer Learning of Hierarchical Task-Network Planning Methods in a Real-Time Strategy Game

Stephen Lee-Urban Héctor Muñoz-Avila

Department of Computer Science
Lehigh University
Lehigh, PA
{sml3,hem4}@lehigh.edu

Austin Parker Ugur Kuter Dana Nau

Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, Maryland, 20742
{austinjp,ukuter,nau}@cs.umd.edu

Abstract

We describe a new integrated and automated AI planning and learning architecture, called Learn2SHOP. Learn2SHOP departs significantly from the previous works on AI planning and learning in that its modular architecture integrates *Hierarchical Task Network (HTN)* planning, concept learning, and computer simulations. Using simulations during the planning and learning process enables the system to get information about the outcomes of the actions. We have implemented Learn2SHOP and tested it on a transfer-learning task. The objective of transfer learning is transferring knowledge and skills learned from a wide variety of previous situations to the current, and likely different, previously unencountered problems(s). The experiments with Learn2SHOP have demonstrated the advantages of integrating planning, learning, and simulation in a real-time strategy game engine.

Introduction

Learning in the context of automated planning has been a frequently studied research topic (Zimmerman & Kambhampati 2003), where the objective is to develop and use automated techniques to learn some knowledge that is used to improve the performance of a planner. Many different techniques have been developed with this objective, including *learning macro-operators*, e.g., (Mooney 1988; Botea, Müller, & Schaeffer 2005), *learning search control knowledge* (Mitchell 1977; Minton 1988; Fern, Yoon, & Givan 2004), *learning of task hierarchies* (Choi & Langley 2005; Reddy & Tadepalli 1997; Ruby & Kibler 1991) and *learning plan abstraction* (Knoblock 1993; Bergmann & Wilke 1996).

This paper focuses on how to take knowledge that was acquired under one model and harness it in the learning within another model (e.g., taking lessons that were learned in one planning scenario and using them in other (similar) planning scenarios). We address this issue with a modular framework called Learn2SHOP based on *Hierarchical Task Network* (hereafter HTN) planning. Unlike other systems, Learn2SHOP learns using simulation in an actual gaming environment to validate the expectations of a provided model. Further, Learn2SHOP uses a general

HTN framework which contains *methods* describing standard techniques. Since these techniques can be applicable to large classes of games, Learn2SHOP will be able to handle changing environments and be able to transfer knowledge from one game to the other.

Learn2SHOP has been tested in a transfer learning task. The objective of transfer learning is transferring knowledge and skills learned from a variety of previous situations, called *source* problems, to the current, previously unencountered problems(s), called the *target(s)* (where significant differences may exist between these two problem types). For this, a transfer learner needs to have some knowledge about the underlying characteristics of both source and target problems. Transfer can be especially effective when such knowledge can be represented suitably structured, e.g., in a relational fashion as in reinforcement learning and/or in a hierarchical fashion as in HTNs.

The overall algorithm employed by Learn2SHOP takes as input example solutions to given problems. It acquires data on the performance of these examples through simulation in the game's actual environment. This data is then used in a concept learning algorithm to determine the applicability of the various HTN methods to the given game. As learning progresses, Learn2SHOP becomes more and more certain of which HTN methods are best in which situations and performs better at the provided game. If the game is switched, Learn2SHOP will not be confused but will proceed as expected: applying lessons already learned that work, and re-learning those lessons which worked before but do not apply to the new environment.

Hierarchical Task Networks in AI Planning

For modeling structured knowledge about a problem domain, one of the best-known approaches is *Hierarchical Task Network (HTN)* planning. An HTN planner formulates a plan by decomposing tasks (i.e., symbolic representations of activities to be performed) into smaller and smaller subtasks until tasks are reached that can be performed directly. The basic idea was developed in the mid-70s (Sacardoti 1975; Tate 1977), and the formal underpinnings were developed in the mid-90s (Erol, Hendler, & Nau 1996).

An *HTN planning problem* description consists of the following: the initial state (a symbolic representation of the state of the world at the time that the plan executor will begin

executing its plan) and the goal task network (a set of tasks to be performed, along with some constraints over those tasks that must be satisfied). A solution to an HTN planning problem is a *plan*; i.e., a sequence of actions that, when executed in the initial state, perform the desired tasks.

In order to generate solutions for the planning problems, an HTN planner uses an *HTN domain description* that contains two kinds of knowledge artifacts: *methods* and *operators*. HTN planners may have other kinds of knowledge artifacts as well. For example, the SHOP planner (Nau *et al.* 1999) also has axioms that can be used to infer conditions about the current state.

The operators are like the planning operators used in any classical planner. The names of these operators are designated as *primitive tasks* (i.e., tasks that we know how to perform directly). Any task that does not correspond to an operator name is a *nonprimitive* task.

Each method is a prescription for how to accomplish a non-primitive task by decomposing it into subtasks (which may be either primitive or non-primitive tasks). A method consists of three elements: (1) the task that the method can be used to accomplish, (2) the set of preconditions which must be satisfied for the method to be applicable, and (3) the subtasks to accomplish.

For example, consider the task of moving a collection of boxes from one location to another. One method might be to move them by car. For such a method, the preconditions might be that the car is in working order and is present at the first location. The subtasks might be to open the door, put the boxes into the car, drive the car to the other location, and unload the boxes.

We define the notion of an *HTN trace* as follows. Given a task t , an *HTN trace* for t consists of a plan p that accomplishes t and a subset of the methods from the input domain description that, when successively applied to t and its subtasks, generates the plan p . Given an HTN planning problem P , a *solution HTN trace* is an HTN trace that generates a solution plan for the goals of that planning problem.

We assume that operators do not have any preconditions, and we focus on learning the preconditions of methods. An *HTN learning problem* description consists of an HTN planning problem and a solution HTN trace for that planning problem, and an *incomplete* HTN domain description which includes the set of operators and possibly the axioms and a list of *skeletal methods*. A *skeletal method* is an HTN method as describe above, but it only contains the head (i.e., the task it accomplishes) and the subtasks. A skeletal version of a method provides the learners with inferences derived and decisions made while this plan was generated (e.g., by a human expert), but it does not provide any information about the applicability conditions of those methods.

Concept Learning via Candidate Elimination

Candidate elimination is an incremental concept learning algorithm that uses training examples to construct a lattice of possible hypotheses for the concept to be learned (Mitchell 1997a; Hirsh 1994; Hirsh, Mishra, & Pitt 2004). Each hypothesis is a boolean formula which returns true or false,

and each training sample gives an example that either fits or does not fit the target concept. A concept, in our case, corresponds to an applicability condition of an HTN method.

The lattice is defined by the *generality* operator, whereby hypothesis h_1 is *more general than* hypothesis h_2 iff h_1 returns true whenever hypothesis h_2 returns true. Notice that by this definition h_1 returns true at least as often as h_2 .

We can imagine the set of all hypotheses which correctly label a set of given examples. These valid hypotheses are called a *version space*. We can compactly represent a version space with two borders, G and S , in the generality lattice. G is the set of the most general hypotheses that are consistent with the examples, and S is the set of the most specific hypotheses that are consistent with the examples. It is well known that anything which is more general than a member of S and less general than a member of G will be in the version space (Mitchell 1997b). As more training examples are given, G and S can easily be refined such that they still contain the entire version space. The algorithm stops when G and S converge to the same hypothesis.

To limit the sizes and to increase the effectiveness of G and S , it is necessary to restrict the set of hypotheses the algorithm may return. This restriction is called the *inductive bias*. It limits the concepts which may be learned.

Candidate elimination is the ideal choice for Learn2SHOP because it admits a useful intermediate representation. That is, before the algorithm finishes via convergence, we can still extract a provably correct partial hypothesis from the lattice. This is accomplished with the general and specific borders. If all members of the general border tell us a given sample is false, then every hypothesis in the version space will also return false, so we can safely return false. If all members of the specific border tell us a given sample is true, then again, every member of the version space must return true for the same sample and we can safely return true. If the borders disagree, then the learning algorithm knows it cannot yet correctly classify the given sample.

Consider now a version space which has been learning a concept for a given game. The intermediate representation allows us to use the current information from the game before the learner is finished learning.

Another property of the candidate elimination algorithm is the collapse when presented with inconsistent data. That is, sometimes the version space can contain zero hypotheses, because there are zero hypotheses which match the input data. So, for instance, if in the Learn2SHOP system we begin learning a concept in one game, and then switch to another game where the concept is fundamentally different, the version space will collapse because the samples from one game conflict with the samples from the other game. This naturally informs the system that it needs to restart the learning process for that concept in the new domain – we were not able to transfer knowledge in these cases.

The Learn2SHOP Architecture

Figure 1 shows the Learn2SHOP architecture. The learning component's inputs (label 1 in Figure 1) include the

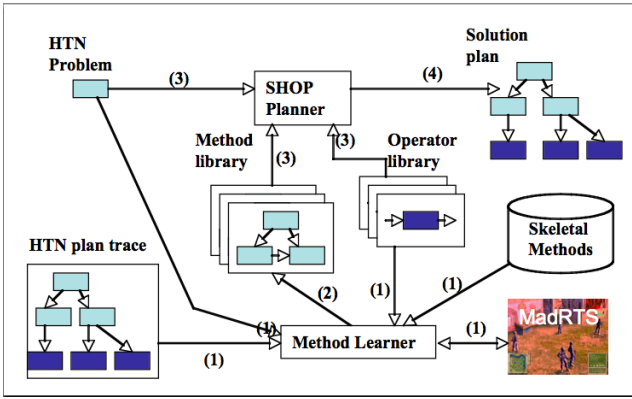


Figure 1: The Learn2SHOP architecture includes the following integrated modules: SHOP as the planning engine, a method-learning module that can accommodate any of several different learning algorithms, and MadRTS as the world simulation engine.

HTN planning problem, the HTN plan trace obtained while solving the problem, the list of skeletal methods, and the list of operators. The learning component uses a running instance of MadRTS, a real-time strategy game engine, to test via simulation if the plan resulting from the HTN plan trace solves the problem. The learning component outputs methods, which are added to the list of methods (label 2 in the figure). Given a new planning problem, the HTN planner uses the methods learned so far and the operators to produce a solution plan (labels 3 and 4). We now describe each of these components in detail.

SHOP (Nau *et al.* 1999) is an HTN planner for a special case of HTN planning in which the subtasks of each method are totally ordered, (i.e., they need to be accomplished in a strict linear order). SHOP proceeds by decomposing tasks in a left-to-right depth-first fashion, backtracking whenever it reaches a point where it cannot proceed further. As a consequence, SHOP generates the steps of each plan in the same order that the plan executor will execute those steps. Because of this task-decomposition strategy, SHOP knows the current state at each step of the planning process. Since it is easier to reason about what is true than what might be true, this makes it easy to incorporate substantial expressive power into the planning system, such as the auxiliary functions and axioms mentioned earlier.

In Learn2SHOP, training examples for the learning algorithms are extracted from annotated HTN traces. An annotated HTN trace contains information about specific instances in which methods were applicable. These instances serve as the positive examples for the version space. The annotated HTN may also contain information on inapplicable methods; i.e., information that indicates specific instances in which methods were inapplicable. These instances serve as the negative examples for the learner. Although there is no requirement that negative examples be provided, the more such examples are provided, the faster the version spaces will converge to the applicability conditions of methods.

To create annotated plans for use as training samples, Learn2SHOP uses a modified version of the SHOP planner described above. The modifications vary the order in which SHOP attempts the applicable methods so that all applicable methods are equi-likely to be returned. A further modification causes the planner to record the negative methods in the annotated plan trace as necessary. This modified SHOP can then take a correct set of HTN methods and use them to create training samples for the system to learn from.

We use a system called CaMeL (Ilghami *et al.* 2005) to learn HTN method preconditions from the annotated HTN traces. CaMeL uses a variant of candidate elimination to learn the preconditions for the input skeletal methods. The algorithm maintains for each skeletal method a version space, which represents the set of possible hypotheses for the applicability conditions of that skeletal method. Each such version space is refined as positive and negative examples are provided until it converges to a single hypothesis, i.e., an applicability condition. At this point, the method, with the learned condition, is added to the method library.

The applicability conditions of a method may contain free variables, yet a version space is always restricted to only ground formulas. CaMeL resolves this conflict by manipulating the version space input. This manipulation comes in the form of inverse substitution, or the reintroduction of variables into the state. For instance, the inversion of the state variables $\{fact(5, 3), fact(2, 5)\}$ with the substitution $\{?x \rightarrow 5, ?y \rightarrow 2\}$ would yield to $\{fact(?x, 3), fact(?y, ?x)\}$, where $?x$ and $?y$ are two variable symbols. To the version space, the variables in the inverted state appear as syntactic objects of the domain language (i.e., as constant symbols), while outside the version space we know they are not. Because the annotated HTN traces contain ground methods, CaMeL can calculate the substitution needed to unify the skeletal method with that in the provided state. This is used for positive samples which are applicable when unified with the provided state. For negative samples, the system constructs all possible substitutions, and uses each to construct an independent negative example from the given state.

We note that this inverse substitution method is not correct and sometimes results in errors when the same constant appears in both a substituted predicate and an unrelated state predicate. In cases where the errors are significant, they cause version space collapse and the system recovers via version space restart.

The overall CaMeL algorithm thus works as follows. For each skeletal method m , the algorithm creates a version space. The version space uses the inductive bias containing conjunctions of literals. Any annotated HTN trace that includes m is used as training data for this version space. If m is part of an HTN trace, then the state at which it is used is inverted to contain the appropriate variables and given to the version space as a positive example. If m is mentioned as inapplicable, then the state at which it was inapplicable is taken, and, for every possible variable assignment, inverted and given to the version space as a negative example.

When extracting an answer from the version space we bypassed the usual method of considering only the consensus

of all hypotheses in the version space, and instead choose a random member of the most general border. The HTN preconditions were therefore too general and admitted false positives; better too many plans than too few plans.

As the simulation framework in our architecture, we used MadRTS (TM), a real-time strategy game developed by MadDoc (R) Software, LLC. Real-time strategy (RTS) games describe a class of games involving real time competition against other human or AI opponents. These games are popular in the consumer market, and as such are not contrived for AI researchers and developers. Consequently, RTS games are characterized by their complexity, often having large decision and state spaces (Aha, Molineaux, & Ponsen 2005). Tasks such as economic/resource management, technology tree exploration, unit production, combat tactic decisions and diplomacy are common in these games.

MadRTS was created for non-commercial military and academic AI research purposes, and is funded by the Navy Research Laboratory. In addition to having a good coverage of the common RTS tasks already described, there is a networking API for the game whereby state information may be queried, and game action messages sent over a TCP/IP connection. In particular, a program may start MadRTS, load a scenario, and start “playing” the game by executing actions and examining the resulting state. This simulation ability ended up being crucial to the transfer learning task. Because the HTN domain modeling game play was approximate and inaccurate there were sometimes errors. However, we could take plans from the approximate HTN domain and simulate them within the actual scenario. By using only those plans which accomplished the goal task, we were able to eliminate errors implicit in the fact that we learn using only an approximate model of the domain.

The Integrated System

The integrated system starts by being given a problem to solve and a set of training samples showing the problem being solved. Further input provide skeletal HTN methods with no preconditions. The training samples are annotated HTN traces (i.e., plans and HTN skeletons annotated with one applicable and inapplicable method for each task that appears in the hierarchy), which can be fed to the CaMeL learning system, however, to determine the veracity of the provided samples, we first run their associated plans in MadRTS. Those HTN traces generated by SHOP which do not solve the problem in MadRTS (i.e., any annotated HTN trace generated by SHOP which turned out to fail due the simulation due to the noise) are thrown out. The remaining samples are actually given to the CaMeL framework producing preconditions for the skeletal HTN methods. Now complete, the skeletal HTN methods with their preconditions are then output to SHOP which then can solve any problem similar enough to the provided problem.

Transfer Learning

The objective of transfer learning is transferring knowledge and skills learned from a wide variety of previous situations, called *source* problems, to the current, previously un-

encountered problems(s), called the *target(s)* (where significant differences may exist between these problem types). To achieve this objective, a transfer learner needs to have some knowledge about the underlying characteristics of both source and target problems. Transfer can be especially effective when such knowledge can be represented suitably structured, e.g., in a relational fashion as in reinforcement learning and/or in a hierarchical fashion as in HTNs.

Different levels of knowledge transfer may occur between the interrelated source and target problems. For example, education research classifies the transfer as *near* or *far*, given the representational differences between the source and target problems (Barnett & Ceci 2002). In machine learning and AI research, on the other hand, knowledge transfer is usually characterized in broad and detailed dimensions and in terms of the knowledge-acquisition and problem-solving capabilities of the learners. In a recent DARPA Program on transfer learning (Oblinger 2005), a taxonomy has been outlined that classifies knowledge-transfer techniques based on differences in the specific knowledge needed to solve different problems, the representation of such knowledge, and the objectives of planning problems.

In this paper, we considered a particular class of *restructuring problems*, where new problem instances involve the same sets of components but in different configurations from those previously encountered during training. For example, in an RTS game such as MadRTS, restructuring is exemplified by problems with different city maps, which could require different instantiation and prioritization of transportation goals (e.g., delivering a resource to a building address versus delivering it to a specified public location in the city).

The next section gives an experimental evaluation of Learn2SHOP in these types of restructuring problems, demonstrating the importance of the modular design in knowledge transfer between two situations.

Experimental Evaluation

The objective of the experiments was to learn to apply knowledge which correctly solves one task to a slightly different task. Namely, we gave our system the knowledge necessary to solve the task of capturing a building in a MadRTS scenario and the system is expected to learn the knowledge necessary to capture an area. These tasks are related – if the area contains a building, capturing the area means capturing the building – but they are not identical. If the area contains no building, it is sufficient to occupy the area with a unit. The transfer-learning related question was, if we are given HTN traces for solving the first task and we learn certain applicability-conditions for the input HTN traces, can we use those HTN traces and the applicability conditions to more quickly and effectively learn to solve the second task?

To provide objectivity, the experiments were performed by an independent third party, namely the Naval Research Laboratory. There were two experiment conditions: the first ran the system without any training on a set of *source* problems (without training on source set - NS), and the second was with training on a set of 5 *source* problems (with training on source set - WS). In the WS setup, the system was trained with all 5 problems in the source set. Under both

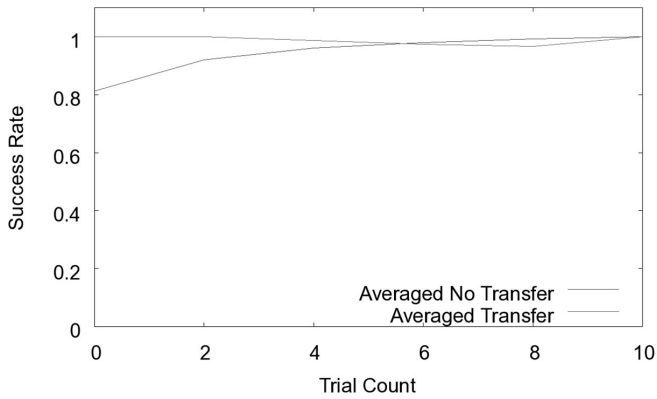


Figure 2: TL Level 3 - Average Curves

experiment conditions, the system was iteratively trained on a *target train* set of problems of size T , where T was varied from 1 to 10 ($T=1, T=2, \dots, T=10$). Under these conditions, the system was tested on a *target test* set of 15 problems. We have 5 problems in the source set, 10 problems in the target training set, and 15 problems in the target testing set.

Four performance measures in the experiments were:

- The Success Rate $[0,1]$: This indicates the percentage of time that the target test problems were correctly solved. A rate of zero means the problem was never solved, and a value of one means it was perfectly solved.
- The Jump Start ($x < 0, x = 0, x > 0$): This indicates the advantage of transferred knowledge in the first trial of the target testing set (when the size of the target train set is $T=1$). A jump start less than zero means that the transferred knowledge is hurting the performance. A jump start of zero means the transferred knowledge was neither helpful nor hurtful. A jump start greater than one means the transferred knowledge was an advantage. The value is computed by calculating the difference between the success rate in the same problem, one with transferred knowledge, and one without.
- The Transfer Ratio ($x < 0, x = 0, x > 0$): The overall advantage of transferred knowledge across the whole testing set (average over all T). The range of values are defined the same as for “Jump Start”.
- The Asymptotic Advantage ($x < 0, x = 0, x > 0$): Indicates the advantage of transferred knowledge in the last trial of the testing set ($T=10$). The range of values are defined the same as for “Jump Start”.

Figure 2 shows the average success rate over all trials ($T=1, \dots, T=10$). The y-axis is the success rate (as a percentage), and the x-axis is the trial count. The difference without training on the target training set ($T=0$) and with training on the source problem set is the “Jump Start”, shown in Table 1, row I. As the size of T increases, around $T=6$, the advantage of training on the source problem set is obviated; therefore there is no asymptotic advantage (row III). However, on average there is an advantage to training on the source problem set, as indicated by the transfer ratio (row II).

Table 1: TL Level 3 - Statistics

TL Metrics		Success Rate	
Name	Type	Score	P Value
I	Jump Start	0.19	0.0000
II	Transfer ratio	3.30	0.0206
	Ratio (of area under the curves)	1.04	0.0150
	Transfer difference	0.33	0.0154
	Transfer difference (scaled)	0.33	0.0142
III	Asymptotic advantage	0.00	0.0000

Related Work

Several architectures have been proposed to learn HTNs from a collection of plan traces and from a given action model. (Reddy & Tadepalli 1997)’s X-Learn, for example, uses inductive generalization to learn task decomposition constructs, which relate subgoals and conditions for decomposing those goals into their subgoals. By grouping goals in this way, task models are learned that lead to significant speed-ups in problem-solving. As another example, (Choi & Langley 2005) achieves the same objective via problem-solving techniques similar to explanation-based learning.

A crucial characteristic of Learn2SHOP that differentiates it from the works above is that Learn2SHOP provides the capability to use a real-time simulation engine in order to test the acceptability of input plans/planning traces. In this paper, we used Mad Doc Software’s MadRTS real-time strategy game as our simulation engine and the HTN planner SHOP (Nau *et al.* 1999) as our planning engine. In addition to MadRTS and SHOP, Learn2SHOP includes a modular learning component for learning applicability conditions of the HTN methods used by SHOP.

We now discuss related research on AI and gaming simulations. Computer games are considered by some to be the “killer app” for human-level AI, and coordinating behavior is one of many research problems they present (Laird 2001). Real-time strategy games are particularly interesting because they present real-time requirements and provide a rich, complex environment.

Reinforcement learning has been used with success in certain classes of computer games. One of the most well-known examples is Gerry Tesauro’s implementation of a Reinforcement learning agent, called TD-Gammon that plays backgammon at a skill-level equal to the world’s best human players. In addition to proving the power of the Reinforcement learning approach, TD-Gammon revolutionized backgammon when it discovered strategies previously unexplored by grandmasters, yet more effective (Tesauro 1994).

Hierarchical planning has been shown to be a promising means to build computer opponents. For example, Bridge Baron 8 won the 1997 world-championship competition for computer programs using HTN planning techniques to plan its declarer play (Smith, Nau, & Throop 1998). Another use of HTN planning in games is HTNBots which applies HTNs to team strategies in the shooter game Unreal Tournament (Hoang, Lee-Urban, & Muñoz Avila 2005).

Conclusions

In this paper, we have described a new, modular architecture for integrating planning and learning of Hierarchical Task Network methods. Our architecture, called Learn2SHOP, is able to perform simulations in a game environment in order to gather information about the state of the environment.

We have tested Learn2SHOP in a transfer-learning task, where the objective was to take knowledge that was acquired under one model and harness it in the learning within another model (e.g., taking lessons that were learned in one game scenario and using them in other game scenarios). The experiments performed by an objective third-party, namely Naval Research Laboratories, demonstrated the effectiveness of our integrated system in a suite of performance measures of knowledge transfer.

We are currently working on more recent and robust techniques from inductive learning and reinforcement learning research in order to learn HTNs in our framework with and without simulations. Our preliminary ideas include learning both HTN method skeletons and preconditions (e.g., see (Hogg & Munoz-Avila 2007) for a first attempt towards these objectives).

Acknowledgments. We thank our anonymous reviewers for their thoughtful and helpful comments that improved the quality of this paper. This work was supported in part by ISLE (#0508268818) and Naval Research Laboratory sub-contracts to DARPA's Transfer Learning program, and in part by NSF grants IIS0412812 and IIS0642882.

References

- Aha, D.; Molineaux, M.; and Ponsen, M. 2005. Learning to win: Case-based plan selection in a real-time strategy game. In *ICCB*, 5–20.
- Allen, J. F.; Hendler, J.; and Tate, A., eds. 1990. *Readings in Planning*. Morgan Kaufmann.
- Barnett, S. M., and Ceci, S. J. 2002. When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological Bulletin* 128(4):612–637.
- Bergmann, R., and Wilke, W. 1996. On the role of abstraction in case-based reasoning. In *EWCB*, 28–43.
- Botea, A.; Müller, M.; and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *ICAPS*, 231–240.
- Choi, D., and Langley, P. 2005. Learning teleoreactive logic programs from problem solving. In *International Conf. Inductive Logic Programming*, 51–68.
- Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity results for hierarchical task-network planning. *AMAI* 18:69–93.
- Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*.
- Hirsh, H.; Mishra, N.; and Pitt, L. 2004. Version spaces and the consistency problem. *Artificial Intelligence* 156(2):115–138.
- Hirsh, H. 1994. Generalizing version spaces. *Machine Learning* 17(1):5–45.
- Hoang, H.; Lee-Urban, S.; and Muñoz Avila, H. 2005. Hierarchical plan representations for encoding strategic game ai. In *Proc. Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)*. AAAI Press.
- Hogg, C., and Munoz-Avila, H. 2007. Learning Hierarchical Task Networks from Plan Traces. In *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning*.
- Ilgami, O.; Nau, D. S.; Muñoz-Avila, H.; and Aha, D. W. 2005. Learning preconditions for planning from plan traces and HTN structure. *Computational Intelligence* 21(4):388–413.
- Knoblock, C. A. 1993. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Kluwer.
- Laird, J. 2001. Using a computer game to develop advanced AI. *Computer* 34(7):70–75.
- Minton, S. 1988. Learning effective search control knowledge: An explanation-based approach. Technical Report TR CMU-CS-88-133, School of Computer Science, Carnegie Mellon University.
- Mitchell, T. M. 1977. Version spaces: A candidate elimination approach to rule learning. In *IJCAI*, 305–310. Cambridge, MA: AAAI Press.
- Mitchell, S. 1997a. A hybrid architecture for real-time mixed-initiative planning and control. In *AAAI/IAAI Proceedings*, 1032–1037.
- Mitchell, T. M. 1997b. *Machine Learning*. McGraw-Hill.
- Mooney, R. J. 1988. Generalizing the order of operators in macro-operators. In *ML*, 270–283.
- Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In Dean, T., ed., *IJCAI*, 968–973. Morgan Kaufmann Publishers.
- Oblinger, D. 2005. Darpa transfer learning program: Proposer information pamphlet. http://www.darpa.mil/ipto/solicitations/closed/05-29_PIP.htm.
- Reddy, C., and Tadepalli, P. 1997. Learning goal-decomposition rules using exercises. In *ICML*.
- Ruby, D., and Kibler, D. F. 1991. SteppingStone: An empirical and analytic evaluation. In *AAAI*, 527–531. Morgan Kaufmann.
- Sacerdoti, E. 1975. The nonlinear nature of plans. In *IJCAI*, 206–214. Reprinted in (Allen, Hendler, & Tate 1990), pp. 162–170.
- Smith, S. J. J.; Nau, D. S.; and Throop, T. 1998. Computer bridge: A big win for AI planning. *AI Magazine* 19(2):93–105.
- Tate, A. 1977. Generating project networks. In *IJCAI*, 888–893.
- Tesauro, G. 1994. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6(2):215219.
- Zimmerman, T., and Kambhampati, S. 2003. Using available memory to transform graphplan's search. In *IJCAI*.