Learning Game Strategies by Experimentation

Thomas R. Hinrichs and Kenneth D. Forbus
Qualitative Reasoning Group, Department of EECS
Northwestern University
2133 Sheridan Rd, Evanston, IL 60208
{t-hinrichs, forbus}@northwestern.edu

Abstract

Deliberative experimental learning is an approach for learning explicit game strategies in a small number of trials by posting and experimentally satisfying learning goals. Learning explicit strategies is important for producing knowledge that can easily be transferred via analogy to new games, as well as for rapid learning. In our approach, experiments, or plans for learning, serve to drive both exploration and credit assignment, by helping to explain the execution trace. We describe a system that learns strategic plans for a subset of games in the General Game Playing (GGP) framework and present experimental results showing that it learns to win most of these games in fewer than 10 trials.

1 Introduction

To learn to play a new game in a reasonable number of trials, how much knowledge must be brought to bear? At one extreme, an agent might play randomly for an enormous number of trials, applying induction to extract winning strategies, or perhaps not generalizing at all. At the other extreme, an agent might start with rich strategic schemas and operationalize them for each new game. We have pursued a middle ground that we call *deliberative experimental learning* (DEL). DEL is an empirical approach to game learning that could be characterized as "Try things that seem promising; explain things in ways that seem plausible." In practice, this means an agent reflects on what it does and doesn't know and possesses heuristic learning strategies for performing *experiments* to find out. These experiments also help to explain the game outcome in terms of higher-level strategic tasks. DEL builds in very little knowledge about game playing, but automatically analyzes game definitions to heuristically extract spatial or other perceptual information about the game that would be immediately accessible to any human sitting down to play, such as the fact that there are pieces that have distinct locations and can be moved.

In this paper, we describe the game-playing task environment, the strategy learning problem, and the experimentation approach. We then describe how this process is implemented in an agent that learns strategies for winning a subset of games in the GGP framework, followed by an experimental evaluation of the strategy learning mechanism and a discussion of its effectiveness and applicability.

1.1 General Game Playing

General Game Playing (GGP) was chosen as the task framework because it can support many different games with relatively little implementation effort. In GGP, games are encoded in a Game Definition Language (GDL), which is similar to a restricted form of Prolog [Love *et al,* 2006]. The GDL encodes a relational net, which concisely represents a finite state machine, i.e., games are restricted to finite, discrete, synchronous deterministic simulations. We focused in particular on piece-moving games in which there is either no opponent, or simple deterministic agents in a two-dimensional spatial grid.

GGP presents some challenges to an essentially knowledge-directed learning mechanism. Although the complete rules of a game are accessible to the learning agent, the semantics of predicates can be arbitrarily obscure. GDL makes explicit the current state of the game, the legal actions, the goal and terminal state descriptions, but it does not provide STRIPS-like action models. One challenge is simply to extract enough information about actions to be able to use the goals of the game to direct search and experimentation, rather than working forward blindly. A key research goal for us therefore is to determine what sorts of game-playing knowledge the agent must have in order to extract useful information and learn winning strategies.

Another challenge with the GGP framework is that each trial of a game starts with exactly the same initial state. For a game like chess, this isn't an issue, but in learning strategies for maneuvering in a maze-like game, this lack of variation makes it hard to generalize and separate coincidences from salient differences. To measure the robustness of learning, we tested our learning agent on 60 variants of three GGP game domains. A variant is a minor modification of initial state, goal definition, resources, or actions. Although we assisted in the development of one of these domains, the others were developed by other researchers.

Game 1: Escape. This is a 2-dimensional board game in which the objective is for the "explorer" to escape across some number of obstacles to reach the exit (see Figure 1).

Figure 1: Escape, Wargame, and Rogue

There are simple resources, such as a hammer, nails, and logs, and the explorer must collect the resources, combine them if necessary, and apply them to the obstacle in order to traverse it. Variants of this game include lashing together barrels with rope to make a raft, destroying a wall with the hammer, and crossing multiple obstacles. One peculiarity of this game is that obstacles are lethal unless destroyed or compromised. Rather than simply preventing movement, if the explorer wanders onto a water square, the game is over.

Game 2: Wargame. This game involves a simple maze-like room in which the "soldier" must kill some number of "terrorists" in order to exit the room (see Figure 1). A number of resources are stashed in the room, such as guns, grenades, and different kinds of distracters. This game has some peculiar simplifications. First, the terrorists are more like zombies. They always move directly towards the soldier and do no path planning. Second, the gun only has one bullet, making it much harder to learn to shoot. Third, the bullets go right through the walls and have infinite range. Lastly, coming into contact with a terrorist will kill the soldier, even if the terrorist is already dead.

Game 3: Rogue. This is a simplified version of the popular Rogue dungeon game[1], limited to a 6x6 grid with two rooms separated by a doorway (see Figure 1). In each room, there is a monster such as a snake or hobgoblin that will attack the "hero". There are various offensive and defensive weapons and magical objects, a valuable amulet and an exit. The hero can attack a monster by trying to move to its square. Location doesn't change in an attack, but the health of both the hero and the monster are decremented by an amount which depends on the weapons and armor used. The objective is for the hero to evade or destroy the monsters, acquire the amulet, and reach the exit. Variations of this game include different layouts, different weapons and magical items and goal requirements (e.g., requiring at least one monster to be killed, to be wearing armor, etc.). The game is simplified by automatically picking up objects when they are traversed and wielding armor or weapons when carried.

1 http://rogue.rogueforge.net/rogue-5-4/dod54/

1.2 The Strategy Learning Problem

In GGP, the game definition tells us the complete rules of the game, but it does not tell us how to win. The performance goal is presented as an arbitrary formula to be satisfied. The learning agent must extract whatever it can from the game definition to operationalize the actions and goals, guide the playing of games, and explain success and failure in order to concisely represent the winning strategy.

Moreover, learning a strategy for a given game is different from simply learning to win. Since these GGP games are deterministic and always have the same initial state, merely replaying a winning execution trace would guarantee another win, but only for the exact same game. A *strategy* is a more explicit and general statement of how to win that can be (in principle) adapted to other situations and games. We represent strategies as hierarchical task networks (HTNs) that can be re-instantiated and expanded in different ways in different situations.

For example, in a game of Rogue, a strategy might be to first acquire and wield a sword, then acquire and wear ringmail, kill a snake, acquire the amulet and exit the dungeon (Figure 2). Each subgoal has its own plan for achieving it, such as simply going to the sword. Notice that the strategy is neither fully general (e.g., "acquire an offensive weapon") nor overly concrete (e.g., "go to location 3,4"). The explanation process must lift coordinates and directions to be relative to entities, but need not generalize the entities themselves. Generalizing entities is needed to support transfer learning across game types, a topic which is beyond the scope of this paper.

```
(preconditionForMethod (true)
  (methodForAction (winGame hero)
  (actionSequence
    (TheList
      (achieve (true (carrying weapon1)))
      (achieve (true (carrying armor1)))
      (achieve (true (health snake1 0)))
      (achieve (true (carrying amulet)))
      (gotoLocationOf hero exit))))
```

Figure 2: Typical learned Rogue strategy expressed as an HTN method

2 Deliberative Experimental Learning

The key idea in DEL is that explicit learning goals and experiments drive exploration and explanation. A learning goal is a representation of a knowledge deficit of some sort, such as the effect of an action primitive, or ways to achieve the preconditions of some action. We define types of learning goals and organize them around a hierarchy of models of the game (see Figure 3). The lowest level model is the *action model*, which includes goals for learning the effects of actions, learning the applicability conditions of actions, i.e., tasks for achieving their preconditions, and learning contextual conditions under which an action has an effect. The *physics model* captures knowledge about the environment of the game, what other entities do (adversaries, resources, obstacles), and what causes trends in quantity values. The *tactics model* is concerned with breaking the performance goal down to subgoals and learning how to achieve them. The `DecompositionLearningGoal` takes a conjunctive performance goal and creates permutations of sub-goals from the conjuncts to permit them to be attempted in different orders. (For complex goals, the permutations are generated as needed.) The *strategy model* consists of the top-level goal of learning how to win the game. The main value of the model organization is that it permits learning goals to be scheduled in a rational way. The agent learns to play a game by working primarily from the bottom-up. Only when the available actions are understood does it make sense to focus on the higher-level goals. Note that this is not necessarily true in real life, where misplaced curiosity can literally kill you. For learning games, though, we expect to lose early and often.

2.1 Learning experiments drive exploration

Most learning goals have associated experimental strategies. When deciding what to do in any given turn, one of the top choices is to pursue an experiment for a learning goal. The experiments are tasks that are likely, though not guaranteed, to provide information to satisfy the goal. For example, to learn what an action does, take the action. Of course, if the action is not legal, then it must first achieve the preconditions of the action. If there is no known method for achieving the preconditions, then it must post a learning goal for learning the applicability conditions of the action and revisit the experiment after the precondition task is discovered. The experiments are where game knowledge is encoded. For example, to learn what an entity does, the experiment is simply to go to the entity. This presupposes a piece moving game and the result is often suicide, but it usually provides critical information.

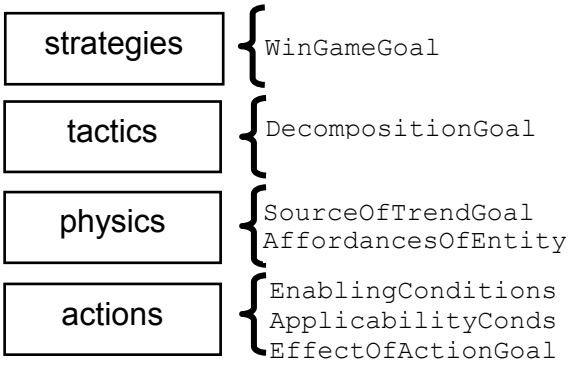

Figure 3: Learning goals are organized by models

2.2 Learning Experiments in Credit Assignment

The execution trace is augmented with the reasons for taking actions. Often the reason is a learning experiment. The experiment tells us what the higher-level task is, such that a result can be explained in terms of the abstract task rather than the ground sequence of primitives. This provides significant leverage in lifting reusable plans. For example, if a subgoal is suddenly satisfied when we intentionally try an action, then we credit the action and not, say, the turn number or particular location.

Credit assignment is defeasible, since it is easy to give an action too much credit. For example, one of the first experiments performed is usually to attempt moving in each direction. This helps establish the correspondence between directions and coordinates, making it possible to plan paths. However, if the agent/avatar is next to some critical resource, it is easy to credit moving north, say, with carrying the resource. This is easy to correct simply by double-checking such learned effects for counter examples at the end of the game and pruning incorrect learned effects. Pruning is harder to do for the effects of complex tasks, but the principle is the same.

For discovering the effects of complex tasks, the difficulties can be characterized as *over-fitting* and *opportunistic discovery*. In over-fitting, the game situation may lead to ambiguous effects. For example, in Rogue, a monster may be co-located with another entity such as a scroll. An experiment to find out what the scroll does would involve trying to go there, but in this case that would mean attacking the monster and losing health points. If the results of an experiment will be ambiguous, it is not pursued and labeled as a failure - possibly to be repeated another time.

Opportunistic discovery entails a different kind of ambiguity. If, for example, the hero stumbles over a potion on the way to the amulet, the agent should learn something, but not that going to the amulet results in carrying the potion. After each turn, if an action became newly legal, a precondition task may be learned by regressing back through the actions and reconciling them against the conjuncts of the precondition. Then, co-locations, adjacencies, and directions are lifted and replaced with path planning tasks to achieve them. Clauses that can be reconciled against known learned tasks are also lifted to those tasks. This prevents precondition tasks from being confused by experimental intent, at the cost of needing to recognize salient spatial relations.

3 Process Description

To apply DEL in a learning agent, are three main processes to implement: 1) an initial domain analysis, 2) planning and executing game actions and within-game learning, and 3) post-game (or post-mortem) analysis.

3.1 Domain Analysis

The first stage in strategy learning is understanding and elaborating the game through static domain analysis. When a new game is first presented, the GDL description is essentially a prolog program with a small number of standardized predicates. From this, domain analysis must determine which of the remaining predicates correspond to game actions, which are types, quantities, spatial coordinates, ordinal relations, and so forth. By looking at the collection of rules, domain analysis first establishes some of the simple algebraic properties of predicates, such as whether or not they are functional, take numeric or symbolic values, are transitive, and/or cyclical. Then, given the assumption of a spatial game, it identifies the most likely candidate predicates for a coordinate system and determines whether the game is a piece-moving or marking-type game and what the tokens or pieces are. Given the defined GDL vocabulary of **`init`**, **`legal`**, **`next`**, **`terminal`**, and **`goal`**, it extracts any special quantity thresholds that may win or lose the game. Sometimes this takes the form of a deadline (e.g., terminate the game after 50 turns), and sometimes there may be a 'health-like' quantity that must not go to zero. In order to facilitate generalization and lifting, it computes equivalence classes of game pieces, which are those pieces of the same type that behave the same way and whose only distinguishing property is their location.

It looks at the **`next`** (state transition) rules to determine how quantities are determined by other quantities or by actions in order to build a simple qualitative model of influences. This permits planning to drive quantities up or down. In a similar way, it examines how the next rules relate legal actions to changes in coordinates of pieces, thereby permitting path planning when appropriate. Quantity analysis and spatial reasoning are fundamental to board games, and are therefore worth extracting in order to apply specialist strategies. Instead of learning how to plan paths, we want to learn how to translate a particular game into path-planning and quantity-planning problems.

After performing this bottom-up analysis, it then works top-down by examining the performance goal description to try to break it into independently achievable sub-goals. This is not always computationally feasible, but when it is, it provides signposts for measuring progress against the overall game goal. Goal analysis works by separating out the positive, achievable parts of the goal state description from the negative conditions to be avoided (which become constraints). It then attempts to satisfy constraints on the positive conjuncts and instantiate possible goal states in terms of concrete pieces, relations, and equivalence classes. Logical simplification rules then clean up the description and record the 'operational' specification of the goal.

3.2 Planning and Executing Game Actions

At some level of description, playing a game is a repeating cycle of choosing a legal action, computing the next state, and trying to learn something. In choosing an action, DEL guides game-play by checking to see if there are any active learning goals to be pursued and preferring actions and experiments that satisfy them. Initially, there are no learning goals and it attempts to win the first game directly by picking random legal moves. The post-mortem from that first game posts learning goals to explain the loss and these guide future games.

Recall that we stratify the goals into models, as per Figure 3 above. When there are active learning goals, it focuses first on goals associated with the lowest-level incomplete model. In this way, the learner starts out working bottom-up by pursuing goals that provide information about actions it can take, their effects, entities in the game, quantities and their trends. When these goals have been achieved, it begins to work top-down by decomposing performance goals, operationalizing the subgoals by replacing variables with concrete entities in the current situation, and trying different orderings of subtasks until the game is won.

After every turn, the agent computes the next state and tries to learn something by explaining the result. Again, learning goals guide the process. For example, if an action became legal and there is an active ApplicabilityConditions goal, it will attempt to explain how it became legal in a way that can be written out as an HTN task denoted as the fluent `(PreconditionOfFn <primitive>)`. If the action became legal at the conclusion of a learning experiment (such as going to an entity to find out what it does), then that experiment plan *becomes* the plan for achieving the precondition of the action. In other cases, it walks back over the execution trace looking for sequences that can be described in terms of known tasks and achievable configurations.

3.3 Post mortem Learning

At the end of a game, the learning agent tries to tries to explain the loss or win through post-mortem analysis. After a loss, the main purpose of the post-mortem is to generate new learning goals. If, for example, an immediate antecedent of the loss was that some quantity had a particular value, then the system should be interested in how that happened. It posts a SourceOfTrend learning goal to monitor and reify the values of that quantity. It posts ApplicabilityConditions and EffectOfAction learning goals to express an interest in how to achieve the preconditions of actions and to characterize their effects. It posts Decomposition learning goals to learn how to decompose a goal to subgoals and sequence them. It posts AffordancesOfEntity learning goals in order to learn what an entity does, i.e., whether it is a resource of some sort, a threat, or a hazard. In addition to learning goals, the post-mortem posts *nogoods*, which are concrete states to be avoided in future trials.

After winning a game, the learner walks back through the execution trace to construct a high-level HTN plan for winning the game in terms of previously-learned subtasks. It applies the credit-assignment criteria described in section

2.2, and lifts particular coordinates and directions into general spatial relations and relative directions. It simplifies the strategy by removing tasks whose outcome is never used, such as achieving preconditions of actions that are never taken.

3.4 Example

To illustrate, consider how it learns to shoot a terrorist in Wargame. It must first learn to avoid the terrorist, and then to acquire the gun, get in position, aim and shoot. In the first trial, the top-level performance goal is simply to be at the exit, which it attempts to do with fatal results. In the post mortem, the antecedent of the loss is found to be that the soldier's health is zero, which upon examining the execution trace is seen to have occurred when the soldier occupied the same location as the terrorist (a salient spatial coincidence). So the terrorist is labeled as a *threat* and will be avoided in future plans. Given the loss, it is clear that the agent needs to understand the game better, so it posts learning goals to discover what all the actions and entities do.

On the next game, one of the top learning goals is to discover what the gun does, so the solder goes to it. When it reaches the gun, the game state shows that it is suddenly holding the gun, so it indexes 'going to the gun' as achieving 'holding the gun'. In addition, it suddenly becomes legal to shoot, so it learns that holding the gun achieves the precondition of shooting. Since there's a goal to understand the effects of actions, it shoots in some random direction, and typically nothing happens. We know from the qualitative model produced in the initial analysis that under certain conditions, the shoot action should have some effect on the health of the terrorist, so the shooting experiment is labeled a failure and the goal remains active.

Because there is only one bullet per game, it takes four trials before it has attempted shooting in all four directions. Usually, it has still failed to affect the terrorist, despite having exhausted arguments to the shoot action. At this point, some other condition must hold before the action can have an effect. The simplest condition that this agent can achieve is a spatial configuration. Since the soldier cannot be co-located with the terrorist, the next simplest configuration is to have the same x or y value. It then repeats the shooting experiments in the context of that configuration. Eventually, it fires in a cardinal direction at the terrorist, killing it. It can now plan a clear path to the exit and win the game. After winning, the fact that the successful experiment was conducted in a simple cardinal direction to the terrorist is used in compiling out a transferable strategic plan.

4 Experiments

One way to characterize this performance is in terms of how fast it can learn to win a game, as measured by how many game trials it takes. For each of 60 different game variants, we ran independent learning experiments, repeating trials until the game was either mastered or exceeded ten trials. A game variant is a difference in initial state, goal description, resources, or impediments. 'Mastery' here means possessing a reliable, but not necessarily optimal, strategy for winning, as evidenced by consecutive wins. Figure 4 shows the average learning curves for game variants in each of the three game domains. These scores are actually averages of winning games and losing games, so a 60% score for trial N really means that 60% of the game variants had been mastered by the Nth trial. In all game families, learning converged within ten trials.

Games that were not mastered failed for one of two reasons: either the agent was reduced to blind choices and failed to stumble on learnable configurations, or there was a behavior it could not explain because it lacked the concept. For example, in Escape the actions for applying resources to a hazard (to destroy or compromise it) are only applicable when the explorer is adjacent to the hazard. Since there is nothing to suggest that it should seek out such adjacent locations, random exploration would sometimes miss them as it wandered aimlessly until it ran out of turns. However, when it did hit on the right configuration, it was able to explain the condition in terms of the known concept of spatial adjacency and create relatively robust strategies.

Other games were lost because they simply lacked a nec-

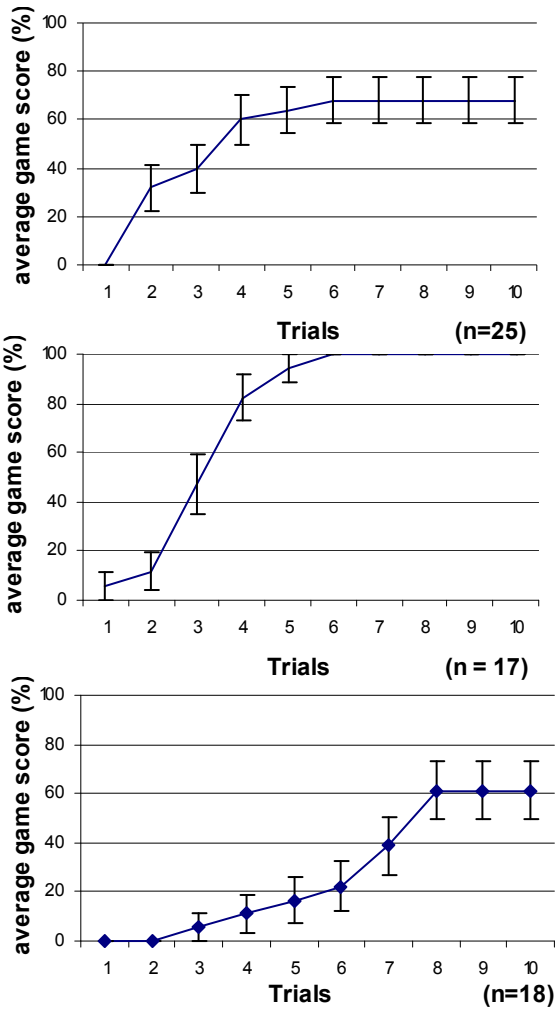

Figure 4: Learning rates for games in the a) Escape, b) Rogue, and c) Wargame domains

essary concept. In Wargame, the games that were not learnable involved using grenades and waiting for multiple terrorists to cluster together so that one grenade could eliminate them all. Here, the difficulty was that the concepts of thresholds, ranging, and doing nothing were not yet present in the system. Even if it were to randomly win such a game, it could not have explained its success or represented a strategy to replicate it. This is not an intrinsic limitation of the learning approach, but illustrates how it is sensitive to, and benefits from, fairly general knowledge. The critical knowledge that was missing was not about grenades and terrorists per se, but about thresholds, distance and time.

5 Related Work

Treating learning as a deliberative process with explicit learning goals is not new [Ram and Leake, 1995], though to the best of our knowledge this is the first time it has been applied to learning game strategies. Carbonell and Gil applied experimentation to learning operator refinement [Carbonell and Gil 1990]. Our work is in the same spirit, but operationalizes known preconditions upon success, rather than learning new conditions by comparing successes and failures.

Domain analysis of GGP games has been described previously in [Kuhlmann *et al.,* 2006]. Their system extracted some similar features in order to construct search heuristics, but did not learn from experience.

CaMeL used candidate elimination over many plan traces to learn complete and correct preconditions of HTN tasks [Ilghami *et al*. 2005]. We learn precondition tasks for primitive actions and sacrifice the guarantee of correctness for 1-shot explanation and lifting.

Partial programming and hierarchical reinforcement learning have also been successfully applied to strategy learning [Marthi, *et al*. 2005]. Partial programs serve a similar function to HTNs in our work, but our emphasis has been less on learning optimal strategies in the limit than on rapidly learning simple, transferable strategies.

Our explanation process for extracting HTNs from the game trace is most similar to that described in [Nejati *et al.* 2006], except that we use explicit learning experiments recorded in the game trace to help lift subgoals and tasks.

Summary and Conclusions

We have shown that a deliberative experimental approach can learn effective strategies in a small number of trials, but also trades off some generality and flexibility. As a knowledge-based technique, it relies on operationalizing known concepts, such as *threat*, *spatial configuration*, and *path*. These seem like reasonable, domain-independent concepts that an agent ought to have before attempting to learn a game. We did not include more game-specific concepts such as *offense* or *defense*, though they might have helped produce more robust strategies.

In general, the rate of learning seems to agree with our intuitions about how humans learn games. While we may refine a concept like 'threat' over many years throughout diverse situations, we devise individual strategies for simple games fairly quickly.

Nevertheless, an important future direction for this work is to incorporate inductive generalization of concepts. This turned out to be impractical for us in the context of GGP experiments, but seems promising for more long-term open-ended learning, and will be necessary for learning strategies that, for example, need to wait for some condition to hold.

Acknowledgments

This research was funded by DARPA under the Transfer Learning program. We would like to thank Jeff Usher for programming assistance and David Aha for co-developing the Rogue implementation with us and running the external evaluations.

References

[Carbonell and Gil, 1990] Jaimie Carbonell and Yolanda Gil. Learning By Experimentation: The Operator Refinement Method. In *Machine Learning: An Artificial Intelligence Approach, Volume III*, Michalski, R.S. and Kodratoff, Y., eds. Morgan Kaufmann, San Mateo, CA, 1990.

[Ilghami *et al*., 2005] Okhtay Ilghami, Héctor Muñoz-Avila, Dana Nau, and David Aha. Learning Preconditions for Planning from Plan Traces and HTN Structure. *Computational Intelligence* 21(4), 388-413, 2005.

[Kuhlmann *et al*., 2006] Gregory Kuhlmann, Kurt Dresner and Peter Stone. Automatic Heuristic Construction in a Complete General Game Player. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 1457-1462, Boston, MA, July 2006.

[Love *et al.*, 2006] Nathaniel Love, Timothy Hinrichs, and Michael Genesereth. General Game Playing: Game Description Language Specification. Stanford Logic Group Technical Report LG-2006-01. April 2006.

[Marthi *et al.,* 2005] Bhaskara Marthi, Stuart Russell and David Latham. Writing Stratagus-playing Agents in Concurrent ALisp. In *Proceedings of the IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*, Edinburgh, Scotland. 2005.

[Nau *et al.*, 1999] Dana Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. 968-973. 1999.

[Nejati *et al.*, 2006] Negin Nejati, Pat Langley, and Tolga Könik. Learning Hierarchical Task Networks by Observation. In *Proceedings of the 23rd International Conference on Machine Learning,* Pittsburgh, PA. 2006.

[Ram and Leake, 1995] Ashwin Ram and David Leake, eds. *Goal-Driven Learning*. MIT Press, Cambridge MA, 1995.