How Qualitative Models can Improve Learning by Experimentation

Thomas R. Hinrichs and Kenneth D. Forbus
Department of Computer Science, Northwestern University, Evanston IL
{t-hinrichs, Forbus}@northwestern.edu

Abstract

One challenge for building software organisms is to support more autonomous, self-directed learning, rather than learning from annotated data or blindly exploring state spaces. We present a method for learning a simple game given a qualitative model. The qualitative model provides partial information about how actions and quantities influence each other, and which goals trade off with each other, allowing the learner to progressively rule out unproductive actions based on qualitative state descriptions of the current situation, and to experimentally adjust the relative importance of competing goals. We show that this amounts to operationalizing a qualitative model into a quantitative prescriptive model, which can lead to rapid improvement in performance on a simple game.

1 Introduction

Any human-like model of learning should account for the role of prior knowledge. When we learn a new task, we do not start from a blank slate, but rather, expectations and beliefs guide actions and explanations to permit learning from far fewer trials than is the norm for today's machine learning. We refer to this as *data efficiency*.

One way that knowledge can guide learning is through self-directed experimentation. We pose questions to ourselves and take actions to triangulate on ever more accurate models. Knowledge about the domain can help pose questions that refine models as well as guide credit assignment. We argue that data efficiency can arise from a more general notion of state. A learned action policy need not map from concrete primitive states to ground primitive actions, but may consist of abstract states and constraints that map to generalized actions. Learning becomes a progressive refinement of states and actions that can stop as soon as performance plateaus, rather than exhaustively searching through primitive states.

This paper brings together experimentation and reinforcement learning, using a qualitative model [Forbus, 2019] as the prior domain knowledge. We show how a qualitative model can support self-directed experiments at a high level by exploring quantitative tradeoffs between competing goals. We also show how the same qualitative model can guide credit assignment to rule out ineffective action policies. Qualitative state representations further serve as antecedent conditions in learned action policy rules.

Previous work in active learning and experimentation has tended to focus on supervised learning of classification tasks or domain theory acquisition and refinement. While this can result in efficient learning, our focus differs in the prior knowledge available to the learner, the means of credit assignment, and learned knowledge being an action policy.

Reinforcement learning tends to focus on a more bottom-up "model-free" learning, at the cost of many learning trials [Sutton and Barto, 2018]. Although our mechanism is also unsupervised, it leverages a qualitative domain model to support efficient learning. We believe this will ultimately enable a continuum of approaches from highly interactive apprentice-like learning to fully autonomous experimentation.

This paper describes a system that learns to play a simple game given a qualitative model of the mechanics of that game. Next we describe the domain task, the Human Resources Manager game, followed by how it is played using a qualitative model and goal network. Section 4 presents the learning mechanism, including credit assignment, experimentation, the representation of experimental controls and learning goals. Section 5 presents the results of empirical experiments. Section 6 compares this to related work and section 7 presents conclusions.

2 The Domain Task: HRM

Human Resources Manager (HRM) is a single-player game in which the objective is to manage a small printing company for twenty months without driving the company into bankruptcy or ending with a negative cash flow. The player starts with $50,000 and a roster of three employees and makes HR decisions about hiring, firing, training, promoting and giving raises. Unhappy employees quit and former employees sue the company if they were fired improperly.

HRM was adapted from a 27-year old corporate training simulator [Feifer and Hinrichs, 1992]. It is implemented via backchaining rules in a form similar to the Game Description Language [Genesereth and Thielscher, 2014]. We chose HRM because it was simple to implement, has a complex underlying mathematical model, and yet it factors out adversarial and stochastic complexities. This provides a simple

testbed to explore ideas about autonomous experimentation by enabling the system to impose experimental controls on quantities and actions. We make no claims for its entertainment or pedagogical value.

Negotiating tradeoffs is key in this game, as in most strategy games. Finding an effective compromise between competing demands is an abstract task that is a major constituent of learning a strategy. One of our research goals is to discover how to acquire such strategic knowledge with the same basic mechanism as learning action-level policies.

There are three main tradeoffs in HRM: First, the goal to reduce labor costs with a low headcount competes with the goal to maximize income. Second, the goal to keep employees happy with high salaries competes with keeping salaries low to minimize labor costs. Third, the goal to invest in employee training competes with keeping payroll costs down. Discovering quantitative compromises for these goals can be thought of as turning a qualitative model into a partly quantitative model.

3 The Game Interpreter

Before describing the learning mechanism, it helps to first understand how the game player works. It first sets up the initial state consisting of quantitative properties and relations of the simulated company. On each turn it queries for legal actions, selects one and applies it, and computes the next state. Most actions are domain-level primitives that can be applied to individual employees, such as giving a raise or evaluating them. These have immediate effects, so we refer to them as *synchronic* actions. There is a special *diachronic* operator, doNextTurn, that advances the simulated time by one month. This allows the player to take any number of actions within a turn and then explicitly advance the time. This happens automatically when there are no more viable actions to take in a turn. When a game is over, the score is computed.

Selecting good actions is what the system must learn. Instead of starting with a blank slate, as most RL systems do, it has a qualitative model of the quantities in the game, the graph of their influences, and the qualitative effects of actions on quantities. For example, giving an employee a raise increases their salary, which in turn positively influences the employee's attitude and the company's labor costs. The learning problem is to figure out how to balance these competing factors and to identify conditions for taking actions.

The qualitative model was produced manually by abstracting the equations in the game's rules. Prior work has shown the feasibility of learning a qualitative model from demonstration [Hinrichs and Forbus 2012], but this was not the current research focus. The HRM model has 37 reified quantity types and 53 influences between quantities, actions, and events. A quantity type may be instantiated for each employee or for the company itself. Because the qualitative model ultimately connects intermediate quantities like salary to the top-level game goal, it is possible to automatically

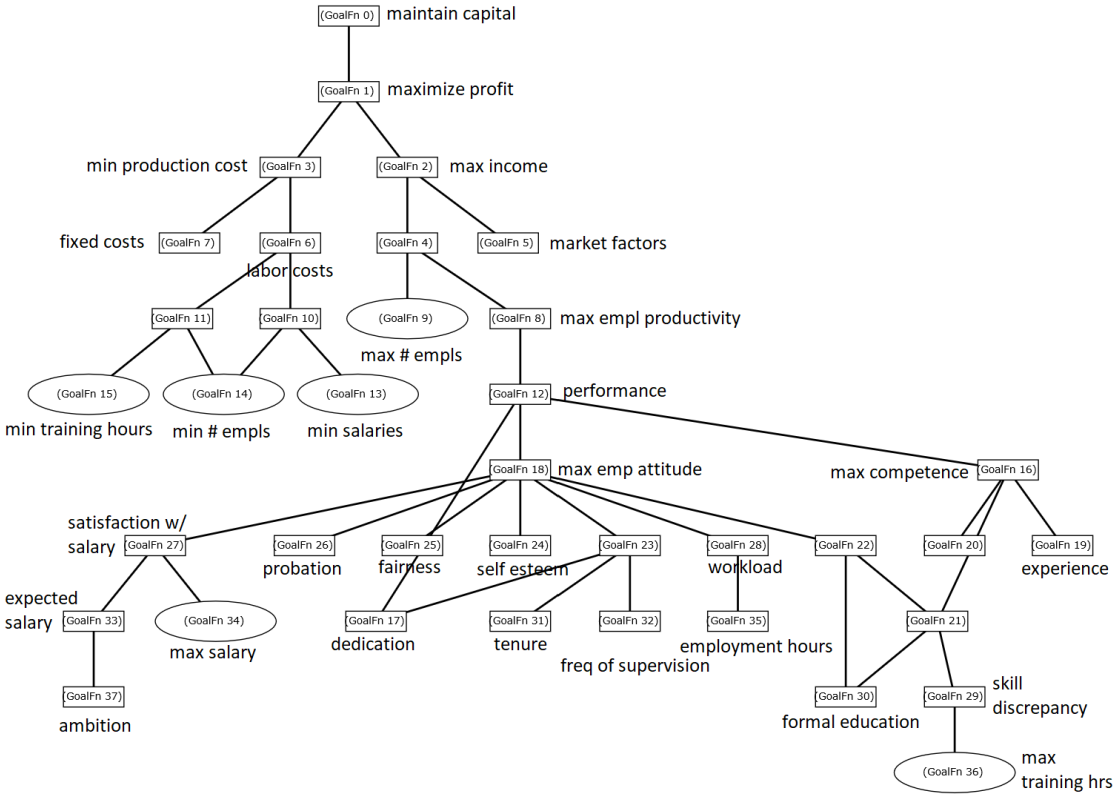

Figure 1: Goal network for HRM computed from the qualitative model

translate the influences into subgoals of the game goal. A static analysis routine walks the qualitative influences to reify goals, as described in [Hinrichs and Forbus, 2016]. Here the goal types produced are all of the form maximize (or minimize) some quantity type. Static analysis during construction detects tradeoffs by identifying quantity types that both positively and negatively influence the same quantity. Figure 1 shows the goal network for HRM, with oval nodes indicating goals involved in tradeoffs.

A goal is *operational* if there is a qualitative influence between some primitive action and the goal quantity, e.g. maximizing an employee's salary is operational because there is a qualitative dependence of employee salary on the action doGiveRaise. Higher level goals, such as maximizing employees' attitudes, may be active, but are not operational because there is no direct control over attitudes.

The reified goal network also keeps track of the relative *activation* of goals throughout the game. The activation of a goal estimates its importance and thereby the proportional allocation of effort expended in pursuing it. Conceptually, if the top goal to win the game has 100% activation, then that activation is subdivided among its subgoals. By default, activation is allocated evenly, so that it serves as an informal proxy for importance relative to the top goal. Also, goal activation can be explicitly set by a meta-level planning action, used to experimentally explore tradeoffs.

The effect of goal activation is to control the likelihood of picking actions that serve one goal over another. For goals pertaining to a single entity, such as the company, this results in stochastically picking an action or not, whereas for goals that apply to many entities, it selects a subset of entities to act on. For example, if the goal of maximizing salaries is only 20%, then only 20% of employees should receive raises. We refer to this as an *action budget* for a type-level goal. The action budget ensures that no single action type monopolizes available resources. Although it still allows raises to be given every turn, the action policy refinement learns to suppress this when the actions have no positive benefit on higher-level goals, as described later.

Algorithm 1 Action Selection

Input: domain goals
Output: execution of actions in simulated world
1: **foreach** domain *goal* in decr. order of activation **do**
2: **while** meets_action_budget(*goal*) **do**
3: *legal* ← legal_actions(*goal*)
4: *acceptable* ← filter_by_action_policy(*legal*)
5: *action* ← argMin(goal_perf (entity(a), *goal*))
6: Perform *action*.
8: record before/after quantity changes
9: refine action policy
10: **end while**
10: **end foreach**

Algorithm 1 outlines the action selection process. When the game player chooses an action to take, it steps through active, operational domain goals in decreasing order of activation. It identifies action predicates that influence the goal quantity and queries for ground legal actions. If there are action policies or experimental conditions on the action predicate, it filters the actions and selects the action whose entity argument is the most underperforming with respect to the goal (hence, argMin with respect to goal_performance). For example, only the most underpaid employees should receive raises. Finally, it takes the action in the game and records the quantity changes as it computes the next state. Any expectation violations here are passed to credit assignment to construct or refine an action policy for the action predicate.

4 Learning Mechanism

Our objective is to learn abstract lessons autonomously with as few trials as possible, using a qualitative model to guide experimentation strategically, and enabling credit assignment to extract more powerful lessons from each trial.

4.1 Credit Assignment

Drawing more general conclusions from each trial promotes data-efficiency in learning. When the learner loses a game, it looks back in time to the most recent action that set it up to lose the game, using the qualitative model to reconstruct the causal trail back to poor decisions. This post-mortem analysis identifies the quantities contributing to the loss. For HRM, this is the company's capital reaching zero. It traces backward, looking for a change in the derivative of capital until it reaches the turn in which some action influenced the company's capital. It searches the indirect influences on capital until it finds an action that negatively impacted the profit rate, such as giving a raise or firing somebody. It posts learning goals to learn the conditions under which action primitives should be applied, creates or refines action policies, and schedules follow-up experiments for further refinements.

4.2 Generalization

To prevent the same mistake from being made in similar circumstances, an action policy is constructed for that action. Whereas an action policy in most reinforcement learners maps directly from states to utilities of actions, our learner instead acquires and progressively generalizes constraints on actions. In particular, an action policy rule relates a qualitative state characterizing the condition with an action specification that may itself be lifted or generalized. For example, a policy might prohibit promoting Alice when her performance is less than 20 and her attitude is less than 50. Such a rule would look like:

```
(controlConditionLowerBound
 (LearnCondForActionFn doHRMPromote)
 (MostSpecificConditionFn doHRMPromote)
 (ruleOut (doHRMPromote Alice)))
```

where first argument is the learning goal, the second argument is a functional term denoting the name of a model fragment that defines a qualitative state, and the third term is the action specification. The model fragment, in turn, relates the quantity conditions:

```
(and (< (performance Alice) 20)
     (< (attitude Alice) 50))[1]
```

As new failure or success instances are encountered, the ranges on quantities are extended and the arguments to the action specifications are lifted as necessary. This representation was adopted to support experimental controls and has the benefit of being relatively concise and explainable.

4.3 Autonomous Experimentation

Autonomous experimentation is the process by which the learner proposes and executes its own experiments to reduce uncertainty. There are two reasons for autonomous experimentation: to strategically curate experience and to simplify credit assignment. We address the former by systematically varying experimental parameters and the latter by controlling other exogenous parameters to restrict possible causes of change. In addition, experiments are organized around explicit declarative learning goals as a way to be more strategic about the exploration process. These learning goals specify two different kinds of experiments that are supported: *action experiments* and *tradeoff experiments.*

An action experiment is created when a postmortem traces a failure to an action that either directly caused a game loss or caused a trend that ultimately led to the loss. The agent posts an action-condition learning goal to refine the conditions under which the action is advisable. It then schedules experiments to refine the conditions by exploring the region between the most specific state to rule out and the most general. In other words, it reduces the uncertainty by driving the qualitative state conditions in a manner similar to candidate elimination in Version Spaces [Mitchell *et al*., 1983].

Tradeoff experiments, on the other hand, explore higher-level decisions by controlling the relative activations of competing goals. If the baseline allocations for competing goals is 50%-50%, then a tradeoff learning goal will spawn two experiments that set activations to 75%-25% and 25%-75% respectively. Subsequent experiments further extrapolate or interpolate the best performing allocation so far. These tradeoff experiments further simplify credit assignment by suppressing all actions that cannot influence either of the competing goals. Consequently, this is an offline policy.

5 Evaluation

We ran learning trials under the two experimental conditions: action learning and tradeoff learning. In the first, we tested action learning by having it play autonomously through pure trial and error while honoring the goals and qualitative model. It learned to rule out actions that failed to have an immediate benefit as predicted by the qualitative model. It also learned from post-mortem analysis to rule out actions that had a long-term negative effect leading to a loss of the game. Initially, performance was spectacularly bad. Because every action in

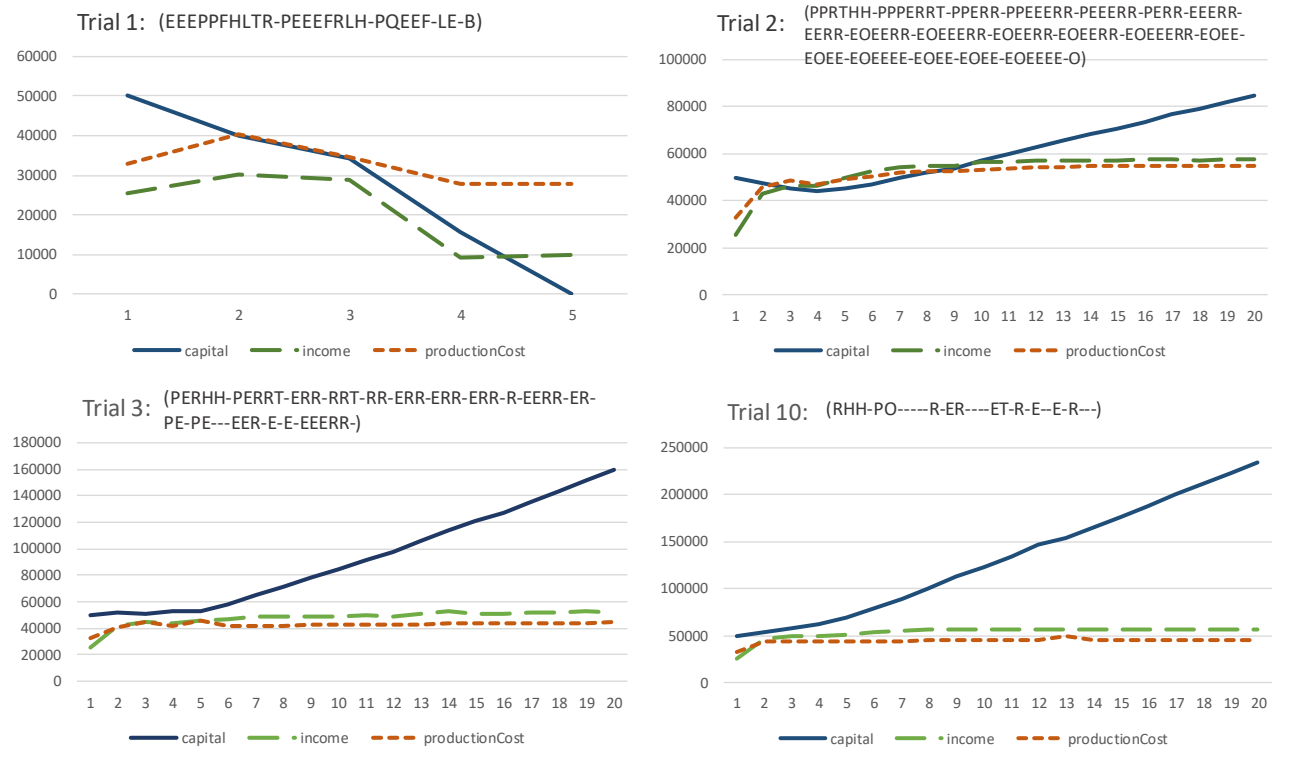

Figure 2: Action learning trials 1-3 & 10. The action abbreviation key is:
Evaluate, **P**romote, **H**ire, **F**ire, **R**aise, **T**rain, **L**awsuit, **O**verpaying, **B**ankruptcy

[1] Simplified syntax for presentation purposes

the game serves some goal, it micromanaged and tried to pursue every action as often as possible. In some cases, it tried firing everybody in the first few turns, leaving the fixed costs to drive the company into bankruptcy shortly afterward.

Figure 2 shows the results of the first three trials and the tenth trial. Each chart shows the progression of the company-wide capital, income, and production cost over time. While the first trial ended with bankruptcy in turn 5, by the second trial, it had learned an action policy that ruled out firing employees in most conditions and had discovered that hiring more employees was the key to surviving past turn 20. Trials 3 through 10 continue to improve the final outcome by increasing the profitability of the company until it banks $240,000 by turn 20 in trial 10.

In addition to performance curves, the charts also present the actual sequence of actions and events in the trial. We can see from this that it quickly stopped firing employees and learned to hire sooner in the game. Moreover, as it refined the action policies, it learned to play with a lighter touch, such that by trial 10, it achieved better performance with far fewer actions consisting of hiring additional employees, giving a few raises and evaluations, one promotion and one training course. Thus the qualitative goal network is refined by the action policy, which provides quantitative constraints on when it is effective to take particular actions. HRM is deterministic and the game objective is not especially difficult to achieve. In fact, under the baseline conditions of taking no actions at all, the company only fails after 19 turns. However, the point of these experiments is to show how quickly it is able to improve given fairly minimal background knowledge.

Figure 3 shows the results of tradeoff learning trials. Here, because tradeoffs can be enumerated ahead of time, an initial set of six trials was scheduled to extrapolate tradeoff ratios in either direction from the baseline tradeoff allocation. In the first two trials, it explores the salary tradeoff by first setting the activation of the goal to minimize salaries at 50% vs 0%

Figure 3: Tradeoff learning trials

for maximizing salaries. Of course, since there is no action to reduce salaries, this translates to never giving a raise. Moreover, since all other actions are suppressed in this offline policy, trial 1 is equivalent to the baseline condition of taking no actions at all.

The next pair of trials explored the tradeoff between reducing labor costs by omitting training (Trial 3) and increasing employee competence by training employees (Trial 4). As with salary, there is no "untrain" action, so by not training anyone and suppressing all other actions, Trial 3 is equivalent to the baseline condition. Trial 4 did train an employee in the first turn, but the only evidence of that is a small spike in production cost, causing it to lose two turns earlier.

The final pair of trials explored the tradeoff between having fewer employees to reduce labor costs (Trial 5) and having more employees to increase production (Trial 6). The effect of reducing headcount by firing approximately half the staff was swift and severe: labor costs dropped, but fixed costs stayed the same causing profits to nosedive leading to bankruptcy in turn 6. Finally, in Trial 6, by hiring two additional employees, income (barely) exceeds production cost and the company remains profitable.

Ultimately, the tradeoff trials are merely suggestive of one way for a learning agent to experiment at a more abstract level than individual primitive operators. As currently implemented, the relative goal activations of competing goals are a coarse mechanism for controlling behavior and further refinement of the tradeoff ratios would not appreciably improve performance in this domain. Despite this, it learned to win in six trials, which is data efficient by most standards.

6 Related Work

The approach described here builds on ideas from several areas, most notably autonomous experimentation, reinforcement learning, and qualitative modeling.

Learning by experimentation requires a learner to design and run experiments to validate or refute its own hypotheses. Part of this involves imposing experimental controls to minimize conflating factors and to simplify credit assignment. Important early work in experimentation includes the operator refinement method [Gil, 1994] which acquired domain knowledge about operator applicability. It used experimentation to identify and refine missing pre- and post-conditions of planning operators that led to anomalous outcomes in execution. Like operator refinement, our system runs experiments to refine the conditions under which an operator can or should be applied. Our approach differs by focusing on learning the advisability of different actions in different situations in order to optimize behavior. A qualitative domain model guides credit assignment and concisely encodes experimental controls. The inequalities in action policy rules effectively turn experimental design into a search in a parametric space.

Like most reinforcement learners, our system performs unsupervised learning. While an important topic in RL is *when* to explore vs exploit learned knowledge [Kearns and Singh 2002; Brafman and Tenenholtz, 2002], we focus instead on experimentation that determines *what* to explore.

Reinforcement learning typically requires hundreds to thousands of trials to learn even simple behaviors because it exhaustively explores the state space of the system. Hierarchical reinforcement learning addresses such high dimensionality scaling problems using temporal abstraction and hierarchical control [Barto and Mahadevan, 2003]. Function approximation accommodates states that take continuous values [Santamaría *et al.*,, 1997]. Using qualitative states to encode action policies could be considered a kind of knowledge-derived function approximation.

In cognitive robotics, [Janež et al., (2013)] used experimentation to learn a qualitative model of robot actions to support prediction and explanation of effects. Part of their strategy for learning faster was to experiment with more complex environments in order to encounter a greater diversity of objects more quickly. In some respect, that is the exact opposite of what our system does, because one of the benefits of experimentation is the ability to simplify credit assignment through controlling and simplifying the environment.

Šoberl et al. (2017) explored the use of qualitative models for driving behavior in a simulated robot. Their qualitative constraints serve a similar purpose to our qualitative action policies, except that they are not revised because the system does not learn.

7 Conclusions

A qualitative model is one kind of prior domain knowledge that can guide learning. It is itself a form of declarative, learnable knowledge that can serve multiple roles in learning to play a game or control a system of some kind. One of those roles is to facilitate experimentation. Experimentation reduces ambiguity in credit assignment by imposing controls on what will be systematically varied and what will be held constant. We have presented two ways to do this: by generalizing or specializing qualitative state conditions on action selection and by manipulating the tradeoff ratios of activations of competing goals. In both cases, the result of learning is to operationalize the qualitative model by learning more quantitative policies for pursuing actions or goals.

A major property of the learning technique described here is that it is data-efficient. It attains good (if not optimal) performance in under ten trials. It achieves this by starting with a qualitative domain model and ruling out vast portions of the potential state space whenever an action fails to provide a predicted performance benefit. It does not need to wait until the end of the game to receive an extrinsic reward, since the model and its derived goal network provide an immediate reward signal via an audit trail from any action up to the top level goal. We believe that the resulting data efficiency is an important property of any learning system that purports to behave in a manner remotely like human intelligence.

Acknowledgments

This research was supported by the US Air Force Office of Scientific Research.

References

[Abel *et al*., 2018] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 10–19. 2018.

[Barto and Mahadevan, 2003] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13(1-2):41–77, 2003.

[Brafman and Tennenholtz, 2002] Ronen I. Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct): 213-231, 2002.

[Feifer and Hinrichs, 1992] R.G. Feifer and T. R. Hinrichs. Using stories to enhance and simplify computer simulations for teaching. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 815–819, 1992.

[Forbus, 2019] Kenneth D. Forbus. *Qualitative representations: How people reason and learn about the continuous world*. MIT Press, 2019.

[Forbus *et al*., 2009] Kenneth D. Forbus, Matthew M. Klenk and Thomas R. Hinrichs. Companion cognitive systems: Design goals and lessons learned so far. *IEEE Intelligent Systems* 24:36–46, 2009.

[Genesereth and Thielscher, 2014] Michael Genesereth and Michael Thielscher. General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(2): 1–229, 2014.

[Gil, 1994] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Machine Learning Proceedings 1994*, pages 87–95. Morgan Kaufmann, 1994.

[Hinrichs and Forbus 2016] T. Hinrichs and K. Forbus. Qualitative models for strategic planning. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems, Atlanta, May*. 2015.

Janež, T., Žabkar, J., Možina, M., & Bratko, I. (2013). Learning Faster by Discovering and Exploiting Object Similarities. International Journal of Advanced Robotic Systems, 10(3), 176.

[Kaelbling *et al*., 1996] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4:237-285, 1996.

[Kearns and Singh, 2002] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning* 49(2-3): 209-232, 2002.

[Mitchell *et al*., 1983] Tom M. Mitchell, Paul E. Utgoff, and Ranan Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine learning*, pages 163–190. Springer, Berlin, Heidelberg, 1983.

[Santamaría *et al*., 1997] J. C. Santamaría, Sutton, R. S., & Ram, A. (1997). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, *6*(2):163–217, 1997.

Šoberl, D., & Bratko, I. (2017, June). Reactive motion planning with qualitative constraints. In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (pp. 41-50). Springer, Cham.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.