Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Learning Qualitative Models by Demonstration

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

Abstract

Creating software agents that learn interactively requires the ability to learn from a small number of trials, extracting general, flexible knowledge that can drive behavior from observation and interaction. We claim that qualitative models provide a useful intermediate level of causal representation for dynamic domains, including the formulation of strategies and tactics. We argue that qualitative models are quickly learnable, and enable model based reasoning techniques to be used to recognize, operationalize, and construct more strategic knowledge. This paper describes an approach to incrementally learning qualitative influences by demonstration in the context of a strategy game. We show how the learned model can help a system play by enabling it to explain which actions could contribute to maximizing a quantitative goal. We also show how reasoning about the model allows it to reformulate a learning problem to address delayed effects and credit assignment, such that it can improve its performance on more strategic tasks such as city placement.

Introduction

A flexible software learning agent should be able to learn through close interaction with a human instructor and progressively become more independent. A natural question, then, is what sorts of knowledge and expectations will enable it to learn from a small number of trials, without unduly constraining what can be learned? We claim that a *qualitative model* can provide useful leverage. This is a kind of conceptual knowledge that can be learned by demonstration using a combination of empirical and analytical methods, and can be exploited in complex performance tasks. Moreover, because it is declarative, it can help to explain a system's behavior.

This paper addresses three main questions: 1) How can a qualitative model improve performance in a dynamic domain? 2) How can such a model be learned by demonstration? and 3) What ancillary domain procedures and rules are needed to exploit a qualitative model and how can they be learned? Our answers to these questions are illustrated in the context of learning to play Freeciv[1], an open-source implementation of a civilization-style game. This domain is challenging because of its complexity and scale, but the mechanisms we present should be applicable to any dynamic domain with quantitative goals that depend on a causal or otherwise deterministic system with continuous inputs and outputs.

We begin by discussing the role of qualitative models, followed by our techniques for learning them and how they are used in turn to learn decision strategies and procedures. We describe an experiment demonstrating that learning qualitative models and using them to support further learning both lead to significant improvements. We then describe related work, future work and conclusions.

Using Qualitative Models to Decompose Goals

Following Qualitative Process Theory (Forbus, 1984), we define a qualitative model as a directed acyclic graph of influences between quantities that are conditioned by active process instances that drive change. These influences fall into two broad categories: *direct influences* of processes, represented by partial information about derivatives and *indirect influences* that propagate their effects, represented by qualitative proportionalities. Together, they represent the causal structure of a system. We believe that such qualitative models have a number of roles to play in learning, including grounding for instruction and advice, and as building blocks for higher-level strategies. Here we focus on how qualitative models can be used to guide the behavior of the game player by linking executable actions to higher-level goals.

[1] http://freeciv.wikia.org

Hinrichs, T. and Forbus, K., Learning Qualitative Models by Demonstration. In Proceedings of the 26th AAAI Conference on Artificial Intelligence, pp. 207-213, July, 2012.

Specifically, by representing the game's performance goal in terms of maximizing or minimizing a quantity, the player can search back through the graph of qualitative influences to make informed decisions about executable actions. Although these decisions may not be optimal, they provide an initial plan which can be debugged and improved. Our primary role for qualitative models, then, is *decomposing* goals into sub-goals.

Of course, regressing to primitive actions is not enough, as they may not be currently legal. In this case, the player must perform still more planning to regress back over the preconditions until a legal action is found. We frame this as a heuristic decision problem of selecting the most direct plan among alternatives. To do this cheaply, we use planning graph heuristics to find the shortest plausible chain from legal action to goal action (Bryce and Kambhampati, 2007).

Qualitative Models in Freeciv

We are exploring the role of qualitative models in the context of learning to play Freeciv (see Figure 1). Freeciv is a good testbed for learning because it embodies complex dynamics, it is sufficiently complex to support sophisticated strategies, and it enables experiments with variable autonomy - the simulation itself provides feedback and built-in bots can be configured to serve as opponents.

A significant aspect of Freeciv is managing the growth of cities. Ultimately, all wealth is produced by working the tiles within city boundaries. Each worked tile produces some amount of food points, production points, and trade points, depending on the terrain and resources present in that location, such as wheat, fish, or iron. This income is then applied to feed the citizens, build and maintain infrastructure and units, and fill the treasury. The exact relationships can be quite complex, factoring in, e.g., corruption, pollution, government types, and civil unrest.

Figure 1: Freeciv city, terrain and resources

A qualitative representation of these relationships captures the influences between quantities as *direct,* e.g., the surplus food per turn directly influences the amount of food in the city's granary or *indirect*, e.g., the food surplus is qualitatively proportional to the food produced there, i.e., it monotonically increases with it, all else being equal. A further complication of this domain is that new entities, such as cities and units, are created dynamically.

Learning Qualitative Models

The target representation for the learned qualitative model is type-level influence statements, rather than propositional influences. For example, instead of learning that the food produced in Boston influences the surplus food in Boston, that statement is lifted to a higher-level statement that the food produced in any city influences the surplus in that same city, e.g.,

```
(qprop+TypeType
 (MeasurableQuantityFn cityFoodSurplus)
 (MeasurableQuantityFn cityfoodProduction)
 freeCiv City freeCiv City equals)
```

The first two arguments are the quantity types related, the 3rd and 4th arguments are the types of entities for which a qualitative proportionality between their quantities holds, given that the relationship in the 5th argument holds between them. In general, the entities need not be the same and the lifting process must search for a binary relation that uniquely relates the entities. Lifting to the type level like this can be viewed as migrating knowledge from episodic memory to semantic memory. It also affords a kind of higher-order qualitative reasoning that scales better, is more easily matched and is likely to be better suited for natural language understanding than propositional representations or traditional logically quantified formulae.

Initially, the learning agent (Forbus et al 2009) starts out with no qualitative model or plans for playing the game, but it does possess a small initial endowment of knowledge about the game in order to recognize possible event triggers, quantity types, and mutually exclusive properties. Specifically, there are representations of:

1) Game types, such as units, cities and technologies,
2) Primitive action types, consisting of preconditions and effect statements,
3) The set of asynchronous game event types, such as UnitBuiltEvent or TechLearnedEvent, and
4) The set of primitive quantity types that may be queried for values, including their units.

These are all formally represented in our knowledge base, using Cyc-style microtheories to contextualize

knowledge[2]. We exploit microtheory inheritance to efficiently retrieve domain-specific quantity types and relationships between game entities.

In addition to these representations, we rely on two kinds of spatial reasoning, due to the spatial nature of the game. Path planning is assumed, as is spatial *scanning*, implemented as a predicate which scans outwards from a starting point for locations matching a given condition.

As the teacher demonstrates the game, the learning agent monitors the communications between the game server and the GUI client. It tries to explain several different aspects of events: 1) the user's motivation in taking an action, 2) the causes behind game events, 3) quantity changes that happen due to actions within a turn, and 4) quantity changes that occur over time across turns. By seeking to explain these events in terms of qualitative influences and processes, it creates defeasible hypotheses that, taken together, comprise a qualitative model.

Learning Qualitative Influences

Just like a scientist, the learning agent makes the most general hypothesis it can to explain an observation and then rejects it later if it is found to be incorrect. We use cases to extract, from the rich perceptual information available in the game state, a concise record consisting of relevant facts to be used in particular learning tasks. For example, it records the type-level influence hypotheses in *influence cases*, along with the propositional observations that support them. These cases serve two functions: they enable the agent to rapidly detect counter-examples to a proposed hypothesis and they enable it to detect trends over time. The learning process can be described in terms of the events, conditions and preferences for proposing hypotheses and the conditions for retracting them.

Two kinds of events drive influence learning: changes to quantities that are detected at the beginning of a turn and changes to quantities within a turn. In Freeciv, multiple actions can take place within a turn, such as moving different units and setting parameters of cities. The effects of these actions are treated as instantaneous, or *synchronic*. Other changes are expressed across turns, such as growth of a city or accumulation of gold in the treasury. These *diachronic* changes are interpreted as the result of processes occurring over time. Qualitative Process Theory (Forbus, 1984) requires that diachronic changes must ultimately be caused by processes, via a direct influence.

The existence of synchronic changes breaks this assumption, since such changes are ultimately driven by discrete actions. This makes finding indirect influences somewhat more subtle. Because we know whether a change was instantaneous or not, we know whether we are looking for a process rate or an action. The challenge for inducing indirect influences is to infer the causal direction. We know intuitively that by placing a worker on a tile, we change the food produced there, which contributes to the food production in the city, which determines the food surplus in that city. Yet, given just a set of numbers, it's much harder for the software to figure out what causes what.

Thus the first step in inducing indirect influences is to determine the quantities at the fringe of the network - the *exogenous quantities* that are determined by the actions rather than influenced by other quantities. We do this in two ways. First, if the arguments to an action directly specify a quantity, such as setting the tax rates, this must be an exogenous quantity, and we label it as such. The second way is if the action introduces (or eliminates) a binary relation that mentions a constant quantity whose magnitude equals the amount of change of some other quantity. For example, if an action assigns a worker to a terrain tile, this adds a binary relationship *cityWorkingTileAt,* that can be interpreted as switching in the contribution of the tile production (a constant) to the city production (a fluent). Being constant, such exogenous quantities cannot be influenced by other quantities.

Next, synchronic changes are compared pairwise to propose hypotheses. For changes to non-exogenous quantities, it searches for likely candidates for influencers, preferring quantities whose change had the same magnitude, with the same units, on the same entity, but progressively relaxing these preferences until a unique influencer can be found. Quantities pertaining to different entities must be related through an explicit binary relationship between their entities in order to lift the influence to the type level. Hypothesized influences that are violated in prior cases or violate the constraints of QP Theory (e.g. no quantity can be directly and indirectly influenced simultaneously) are filtered out. The remaining hypotheses are written out to the KB in an influence case in such a way that they can be rapidly queried for counterexamples. Cases are represented as microtheories in an inheritance lattice and are further partitioned into direct influence cases and indirect influence cases. Efficient microtheory inheritance thus allows it to be quickly determined whether a quantity type has a prior direct or indirect influence hypothesis.

Our influence induction mechanism tends to over-generate. Because this is an on-line algorithm, any coincidence starts out as a potential influence and must be pruned later if counterexamples arise. The most common case is influences with ambiguous causal direction. When two quantities change in lockstep, which causes the other?

[2] The contents of our knowledge base are mostly derived from ResearchCyc, with our own extensions and reasoning engine.

One kind of counter-example that prunes hypotheses is when a purported independent quantity changes but the corresponding dependent quantity does not. Although this can conceivably happen if there are multiple influencers that exactly cancel each other out, it is vastly more likely that the influence is simply wrong.

Learning direct influences is a bit simpler and less prone to error, because there tend to be fewer of them, and by assuming only a single direct influence on a quantity (its rate of change) the magnitude of that rate can be constrained to equal the magnitude of the change of the dependent quantity. Here, instead of examining pairwise changes, it looks for potential direct influencers whose value corresponds to the change magnitude. It still over-generates hypotheses, and prunes using a *3-strikes* heuristic: if a hypothesized influence is violated three times in a row, retract it. Why not immediately? As it turns out, there are sometimes exceptions, i.e., local discontinuities that arise from processes reaching a limit point. So, for example, when a city produces a settler, its normal growth is interrupted and the size of the city drops by one. Since this coincides with an event, we treat it as an exception, rather than an erroneous hypothesis. If it were to happen consistently, though, the influence would be rejected.

Another subtlety of learning direct influences is that some quantities grow as step functions, increasing by one only after many turns, such as citySize. Here, the learner must interpolate a fractional growth rate over many turns and look for possible influencers that are fractional or a percentage. City growth rate is represented as a percentage per turn, so it discovers this influence only after failing to detect a simpler relation.

Figure 2 shows a portion of the learned qualitative model for Freeciv after approximately 10 turns.

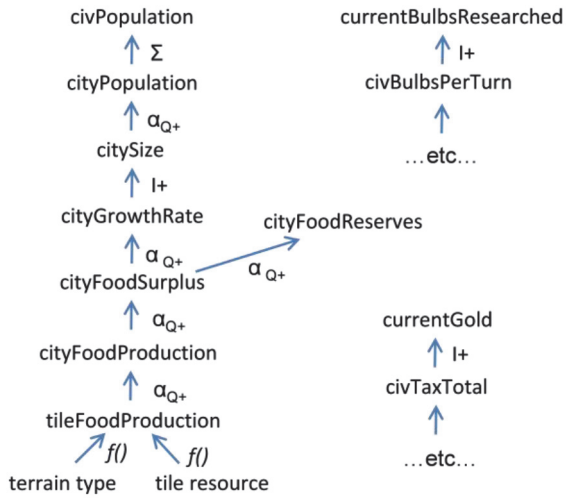

Figure 2: A portion of the learned qualitative model

Learning Decisions and Procedures

To effectively exploit a qualitative model, an agent must also acquire additional procedural knowledge, such as rules to determine when to make certain decisions, plans for achieving preconditions of primitive actions, and policies for choosing among alternatives. These can all be learned via demonstration and a small amount of user interaction.

We divide domain activity into *decision tasks* and *action plans*. Conceptually, we think of decision tasks as selecting one of *N* mutually exclusive alternatives, whereas action plans are sequences of primitive actions that achieve states in order to enable other actions or influence quantities. This ontological distinction has implications for both planning and learning. By itself, a decision consumes no resources in the world, but instead may initiate a process that consumes resources over time, such as deciding what to research or build. Because decisions persist over time, it is important to learn when to make and revise different types of decisions, such as choosing what a city should build when it is first founded, or whenever it finishes building something. Event triggers like these lead to a forward-driven control strategy for decisions.

Actions, on the other hand, are more backwards-driven and may have preconditions whose attainment is arbitrarily difficult to achieve. For example, consider chess, where the entire game consists of achieving the preconditions of capturing the opponent's king. Acquiring plans to achieve action preconditions is a part of the learning process since playing a game entails more than just directly manipulating quantities or initiating processes.

As the learner watches an instructor play, it makes an informed guess about whether a game action is a decision task, based on whether it sets a functional property of an entity. It asks for confirmation, since this isn't a guaranteed discriminator. For example, although moving a unit sets its location, we don't necessarily think of movement as choosing alternative locations: Units move in order to enable some other action.

Learning procedures

Whereas decision tasks are for the most part directly available at any time, actions may have preconditions that must be achieved in order to make them legal. Part of learning by demonstration is learning procedures for making an action legal. These procedures are encoded as methods in a hierarchical task network (HTN).

For example, in Freeciv, the instructor may demonstrate the founding of a city. This increases the population of the civilization by incrementing the number of cities, so given a performance goal to maximize the population, this is interpreted as a good thing to do. Yet the learner must generalize from the training example exactly how to enable

that action. The primitive action takes a unit and a proposed city name as arguments, and the precondition tests the location of the unit to ensure that the city will be built on dry land, but not too close to other cities. However, the location is not a direct argument to the action, but is merely referenced inside some of the conjuncts of the precondition. Although this is a critical independent variable, it is implicit in the execution trace.

To construct a plan to achieve the preconditions of building a city the learner exploits prior hypothesized goals plus the declarative precondition of the action to extract a filter condition on locations that can be applied to a general purpose scanning routine. It creates a plan to achieve the preconditions for building a city by scanning for a legal location, then sending a unit off to that location. This is a combination of retroactively identifying what the user did and inferring the criteria for choosing a location.

Specifically, the learner attempts to explain the motivation behind the teacher's actions. If the action is a step in a known plan, the goal that plan would achieve is recorded as one possible goal for the agent. Later, when trying to extract a precondition plan from the execution trace, it searches these hypothesized goals to see if they match conjuncts of the current actions' precondition. So for example, when a unit is moved, movement serves the (only) built-in plan of going somewhere, which serves the goal of being somewhere. When the build city command is detected, part of its precondition is for the unit to be in a particular location. The conjuncts of the precondition that mention the variable location are extracted into a filter rule that can be passed into a general scanning routine that can be used to find a legal binding for the hidden location variable.

Learning quantitative relations

The mechanism above stops as soon as it finds a legal location to build a city. However, learning *good* locations is a critical part of playing well. Consequently, as soon as a legal procedure is constructed, it posts a learning goal to discover better locations that improve global performance. It seeks to learn an evaluation function that maps a location to a prediction of the goodness of that location, based on whatever turns out to be the dependent variable. The challenge in learning this function is not so much learning the prediction, but rather, determining what the independent and dependent variables are so that it can then learn the function.

There are three subtasks in this process: identifying the independent variable, identifying the dependent variable, and incrementally inducing the function. These subtasks are supported by some additional bookkeeping in the form of an *action case* pertaining to the primitive action type under consideration. Action cases are a subset of indirect influence cases that are indexed by their primitive action type. The representations of the observed added and deleted statements in these cases help to identify the independent and dependent variables.

Identifying the independent variable

The challenge with identifying the independent variable is that it is implicit in the observable action. When the teacher sends a doBuildCity command for a particular unit to build a city with a particular name, the independent variable isn't the city, but the location of the city. Moreover, in order to evaluate a potential city site, we're interested in the salient features of a location, such as the terrain type and special resources present, rather than simply its coordinates.

To do this, it finds a newly added relationship between some argument of the observed action and the selected legal location (in this case, objectAt relates the new city to its location). It writes out rules to extract the intrinsic properties of the location, given an observed doBuildCity action. These rules allow it to reify the values of the salient features in an action case when new cities are founded.

Identifying the dependent variable

Identifying the dependent variable is a different problem. The challenge here is that the difference between building a city in a good location vs. a bad location isn't felt for a long time. The overall performance goal doesn't look any different in the short term. Consequently, the learner regresses through the learned qualitative influences until it finds a quantity that does exhibit variance between cases. In the example of building cities, it must regress from the top-level goal to maximize the overall civilization population, to sub-goals to maximize city sizes, and from there through a direct influence from city growth rate. The growth rate of a city depends in part on the terrain and resources at its center. It discovers this by determining that these are the only properties of the independent variable that correlate with variation in the dependent variable.

Inducing the evaluation function

When the qualitative model is sufficiently fleshed out to permit regressing to the right quantity and there are sufficient action cases stored to identify the dependent quantity, it compiles out a rule to compute the score. As it acquires new cases, it constructs a decision tree in a manner similar to ID3 (Quillian, 1986). Each combination of features discriminates to a range of maximum and minimum dependent variable scores. This range is necessary because the features do not functionally determine the score. In fact, the initial growth rate of a city only partly depends on the terrain and resources of the center tile. A new city has two workers, so it also depends on properties of a second location and on other types of specials, such as rivers, that are not encoded as mutually-

exclusive attributes of a location. The predicted score for a location is simply the midpoint of the range.

This example of learning better city locations shows how decision policies can be learned by first empirically learning a qualitative model, then analytically reasoning about it to extract independent and dependent variables and then re-formulating the problem as a kind of function-approximation.

Experimental Evaluation

In order to show that a qualitative model can be rapidly learned by demonstration and that it has a beneficial effect on performance, we empirically evaluated the system under different learning and performance conditions. With only a single demonstration game, we wanted to measure the improvement due to learning qualitative relations alone, and the incremental benefit of learning a quantitative evaluation function for city siting. To do this, we needed a realistic performance metric.

Freeciv can be thought of as a race to achieve military and technological supremacy. An important part of this is the ability to grow a civilization as fast as possible, especially early in the game. Consequently, the performance goal selected for these experiments was to maximize the population of the civilization over the first 75 turns. After that, expansion has typically saturated the available land and different strategies must be adopted.

We created 10 different terrain maps by starting and saving games on the first turn. We factored out exploration by manually editing the saved games to reveal the entire map. Figure 3 shows the population growth for each of the ten terrain trials under three different conditions: the *legal player* with no learned knowledge of the game picks random legal actions, the *rational player* with a learned qualitative model that allows it to select actions that address goals, and the rational player with *learned evaluation scoring* for placing cities in productive locations.

After one training trial, we can see that the learner has acquired enough of a qualitative model to significantly improve over random play ($p < 5\times10^{-5}$). When it learns to evaluate possible city locations, the performance is better still ($p < 3\times10^{-3}$). This begins to approach an average human player's performance. In order to exceed this performance, it would have to factor in travel cost to reach a location. It sometimes defers settling its first city in favor of moving halfway around the board to reach a marginally better location. It also does not currently learn to spend money to reduce production time. Reasoning about such tradeoffs will be part of our future work.

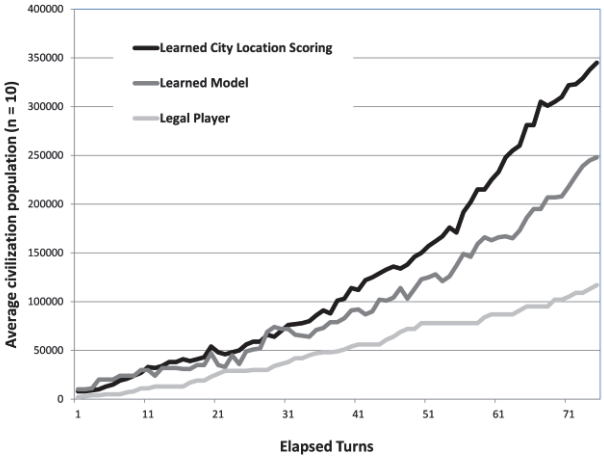

Figure 3: Population growth

Related Work

Previous work has shown learning by demonstration to be very effective for learning concrete procedures. For example, PLOW (Allen et al., 2007) uses spoken-language annotation of teacher actions to learn procedures for filling out web forms. Similarly, TellMe (Gil, Ratnakar and Fritz, 2011), uses controlled natural language to describe procedures interactively to a system. Our work focuses on learning conceptual knowledge from which the system itself derives procedures, and the system only asks users one type of question, otherwise using only observation for input.

Our type-level qualitative representations are inspired by Cycorp's use of type-level representations to provide concise axioms to support natural language processing and reasoning in the Cyc knowledge base.

There has been other research on learning qualitative models, e.g. Padé (Žabkar et al., 2011). Our focus here is on learning incrementally from demonstration. Suc & Bratko (1999) used qualitative representations to learn strategies that "clone" the behavior of expert operators solving continuous control problems (e.g. running a crane), in contrast to our domain which involves more discrete actions. Also, the influences in our Freeciv player are often conditioned on the activation of processes.

An alternative approach to learning qualitative models is used in QLAP, a program that uses Dynamic Bayesian Networks (DBNs) to learn high-level states and actions from continuous input (Mugan and Kuipers, 2012). QLAP is an unsupervised learner and relies on a different set of assumptions than we do. Continuous signals are discretized early into qualitative magnitudes so that it can use statistical inference to progressively introduce landmarks that refine the qualitative representation. It is

able to learn high-level states, but at the cost of requiring many more training instances, which would be problematic for our interactive learner.

Freeciv has been used previously as a testbed for learning, most recently by (Branavan et al 2011), who explored using Monte Carlo simulation to learn action selection strategies. Their approach requires 8 instances of the game running in parallel. Ulam et al. (2005) explored the use of reinforcement learning to learn city defense strategies. Our techniques lead to faster learning, due to their use of qualitative models as an intermediate representation. Hinrichs and Forbus (2007) used a hand-coded qualitative model to learn strategies for city management via a combination of analogy and experimentation. This shows that qualitative models can be used for other tasks than examined here, and we show how such qualitative models can be learned.

Conclusions and Future Work

Learning qualitative models by demonstration is a step towards making agents that increasingly think more abstractly and farther ahead. We have shown that learned qualitative models can provide a significant boost in performance, and can be used to rapidly learn additional strategies that provide further performance improvements. This provides evidence that qualitative reasoning can help an agent to control and focus its learning to incrementally improve its behavior.

This works for two reasons: first, the strong expectation about a qualitative model guides and constrains the learning early on so that it needn't wait for an eventual reward in terms of game performance, and second, by learning from demonstration, the teacher effectively curates the learner's experience so that it is not randomly exploring the game. We expect to see further benefits as we develop capabilities for communicating high-level advice in terms of qualitative relations.

There are a number of other directions we plan to explore in future work. The first is to expand the curriculum for the system, using these same ideas to teach it about economics, exploration, research, and military operations. The second is to explore extracting qualitative models from the text of the manual, by extending previous work on extracting instance-level qualitative models (Kuehne & Forbus, 2004) to extracting type-level qualitative representations from text. Finally, we plan to explore learning in the context of the rest of the apprenticeship trajectory, i.e. with the system taking on increasing responsibility in working in the simulated world.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under Award No. FA2386-10-1-4128.

References

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M., and Taysom, W., 2007. PLOW: A Collaborative Task Learning Agent. In Proceedings of the Twenty Second AAAI Conference on Artificial Intelligence, 1514 1519.

Branavan, S.R.K, Silver, D., and Barzilay, R., 2011. Non Linear Monte Carlo Search in Civilization II. In Proceedings of the Twenty Second International Joint Conference on Artificial Intelligence, 2404 2410.

Bryce, D. and Kambhampati, S., 2007. A Tutorial on Planning Graph Based Reachability Heuristics. *AI Magazine* 28(1): 47 83.

Forbus, K.D., 1984. Qualitative Process Theory. *Artificial Intelligence* 24:85 168.

Forbus, K., Klenk, M., and Hinrichs, T. 2009. Companion Cognitive Systems: Design Goals and Lessons Learned So Far. *IEEE Intelligent Systems*, 24(4), 36 46.

Gil, Y., Ratnakar, V., and Fritz, C., 2011. TellMe: Learning Procedures from Tutorial Instruction. In Proceedings of the 16th International Conference on Intelligent User Interfaces, 227 236.

Hinrichs, T.R. and Forbus, K.D., 2007. Analogical Learning in a Turn Based Strategy Game. In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, 853 858.

Mugan, J. and Kuipers, B., 2012. Autonomous Learning of High Level States and Actions in Continuous Environments. *IEEE Transactions on Autonomous Mental Development* 4(1):70 86.

Quillian, J.R.,1986. Induction of Decision Trees. *Machine Learning* 1(1):81 106.

Suc, D. and Bratko, I. 1999. Modeling of control skill by qualitative constraints. In Proceedings of the 13th International Workshop on Qualitative Reasoning, Loch Awe, Scotland.

Ulam, P., Goel, A., Jones, J., and Murdoch, M., 2005. Using Model Based Reflection to Guide Reinforcement Learning. IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games. 107 112.

Žabkar, J., Možina, M., Bratko, I., and Demšar, J. 2011. Learning qualitative models from numerical data. *Artificial Intelligence* 175:1604 1619.