Using Qualitative Reasoning in Learning Strategy Games: A Preliminary Report

Thomas R. Hinrichs, Nathan D. Nichols, and Kenneth D. Forbus

Qualitative Reasoning Group, Northwestern University
2133 Sheridan Road
Evanston, IL, 60208, USA
{t-hinrichs, n-nichols, forbus}@northwestern.edu

Abstract

This paper describes the use of qualitative models in learning to plan and execute in a turn-based strategy game. We are using a qualitative model primarily to regress from dependent goal variables to independent variables in order to propose possible actions and sub-goals. Unlike physical systems, this is essentially an economic model of influences between various types of production and consumption in simulated cities in a turn-based strategy game. We describe how we are using qualitative models to plan and learn in a strategy game, and experiments in progress.

Introduction

While qualitative reasoning is most often used to simulate physical systems, it has also been used effectively to reason about economics (cf. Farley & Lin, 1990; Kamps & Peli, 1995). Having an economic model can be an important factor in successfully playing modern strategy games, such as Civilization™. We are exploring how such a qualitative model can support planning and analogical learning in turn-based strategy games. Games of this sort have several interesting properties: 1) they involve incomplete knowledge of the world, 2) they entail complex interrelationships between entities and quantities, 3) goals may be more like optimization problems than achievement of states, and 4) planning and executing are tightly interleaved. Qualitative representations can serve as a partial domain theory to guide planning and learning at different levels of expertise.

We are currently focusing on the subtask of managing cities to optimize their use of resources, build improvements, improve terrain, and research technologies. In this paper, we will describe how a qualitative model of city management can support planning and learning in the game of Freeciv.

In the rest of this section, we outline the context of this work, providing a brief synopsis of the strategy game we are using, HTN planning, and analogical reasoning. Next we describe how we are using a qualitative model of city economics to enable a system to plan actions and learn how to improve its performance in playing the game. An experiment in progress is described, and we close by discussing related work and future plans.

The Freeciv Domain

Freeciv is an open-source turn-based strategy game modeled after Sid Meier's series of Civilization™ games (Freeciv, 2006). The objective of the game is to start a civilization from initial settlers in the Stone Age and expand and develop it until you either conquer the world or win the space race and escape to Alpha Centauri. In either case, the game can be characterized as a race to build your civilization and technological sophistication faster than your opponents. Along the way, there are many competing demands for limited resources, investment, and development. For example, players must improve terrain with irrigation and roads, while avoiding famine and military defeat. Too much emphasis on military preparedness, for example, can make citizens unhappy and therefore less productive. Money must be allocated to research into new technologies, such as iron-making and democracy, which enable players to create new improvements to cities, new types of units, and adopt new types of governments, each with their own tradeoffs.

Figure 1: Freeciv

In our current set of experiments, we are focusing on learning how to manage the growth of cities and maximize productivity. While our planner can direct exploration, city management tasks offer clearer evaluation metrics. We also currently ignore military operations, focusing instead on how to make a rich, productive civilization.

HTN Planning

To support performing and learning in the strategy game, we have implemented a Hierarchical Task Network (HTN) planner using the SHOP algorithm (Nau et al., 1999). In the HTN planner, complex tasks are decomposed into primitive executable tasks. The primitives in FreeCiv correspond to packets that are sent to the game server, representing actions such as sending a unit to a particular location or telling a city what to build. Complex tasks are at the level of figuring out what a unit should do on a particular turn, or deciding how to ameliorate a crisis in a city (such as a famine or revolt.) The planner generates plans for each unit and city at every turn and integrates them in a combined planning/execution environment. Planning is invoked partly in an event-driven manner, such that reified events from the game trigger certain decisions. For example, the planning agent does not re-compute its global strategy on every turn, but checks to see if it has acquired any new technologies in the last turn, and only then does it re-evaluate its strategy.

A critical aspect of this game is that it requires planning with incomplete and uncertain information. Terrain is not known until it is explored. The outcomes of some actions are stochastic, for example, village huts may contain barbarians that will kill an explorer, or they may contain gold or new technologies. There is also vastly more information in the game than can be considered within a planning state. Consequently, the planner cannot plan an agent's actions starting with a complete initial state. It must reify information on demand by querying the game state. At the same time, the planner may project the effects of actions such that the planned state deviates from the game state. To reconcile these competing demands, we maintain two *contexts* (cf., Lenat 1995): a *game context* that always reflects the incomplete, but correct current state of the game and a *planning context* in which states are projected forward. Every query for information that cannot be directly answered from the planned state proceeds to query the game state. Before returning such game-state information, it is checked for consistency against the plan state, to ensure that, for example, a unit is not believed to be in two places at the same time.

Analogical Learning

A high-level goal of this research is to demonstrate how analogy and qualitative reasoning can support machine learning across increasingly distant transfer precedents. To do this, we are using the Structure Mapping Engine (SME) (Falkenhainer et al., 1989), the MAC/FAC retrieval mechanism (Gentner and Forbus, 1995), and the SEQL generalization system (Kuehne et al., 2000) as core components. These analogical mechanisms are coordinated and invoked by the planner as it attempts to construct plans and analyze the effects of actions.

In fact, learning is guided by explicit *learning goals* (Ram and Leake, 1995) that are currently specified as part of the problem scenario. A learning goal determines how a decision task will be solved when there are insufficient analogical precedents or canned plans. A typical strategy is *experimentation*, in which the decision is made randomly in order to generate the requisite variation and provide cases that better cover the decision space. When the goal is to learn the effect of an action, additional goals are posted to control parameters (by suppressing decisions) in order to try to learn one effect at a time.

Later, when the learning goals are satisfied, the game performance goals dominate and decisions are made by querying for remindings and mining the prior cases for solutions that can be transferred.

Cases, in this approach, are not entire games (though some lessons can certainly be gleaned from that granularity), nor even entire cities. Instead, a case is an individual decision in the context of a particular city at a particular moment in time in a given game. For example, cases can capture a decision about what improvements to build, what tiles to work, and at the broader game level, what technologies to research. For each type of decision, there is a set of queries represented in the knowledge base that are designated as possibly relevant to making the decision. There is another set of queries that are relevant to capturing and assessing the case solution. When the decision is acted on in the game, a snapshot of the case is constructed before and after execution and stored in the game context. This case snapshot is used both for analyzing the effects of actions and supporting later analogical transfer.

Exploiting and learning a qualitative model

In addition to analogical learning, we believe that qualitative reasoning will be a key factor in supporting transfer learning. Qualitative representations can serve multiple roles in planning, execution and learning. In this section, we describe these roles and some of the issues we face in implementing this integration, starting with the most knowledge intensive and working towards learning more of the domain through unsupervised experimentation.

Planning with a Qualitative Model

In order to know what actions are possible and desirable in a strategy game, two extreme approaches are 1) to make random decisions, and 2) to search through domain-specific, knowledge-rich decompositions of plans. Using a qualitative model of the domain provides a useful middle ground that allows a problem solver to benefit from partial knowledge. The qualitative model captures the sort of knowledge that a novice human player would acquire by reading the manual. Also, qualitative relations are compatible with the sorts of optimization goals one typically has in playing these games, as opposed to the state-based goals found in classical planners. Finally, the only way that an unsupervised learning system will acquire high-level strategies is by synthesizing them. A qualitative model can constrain this synthesis process.

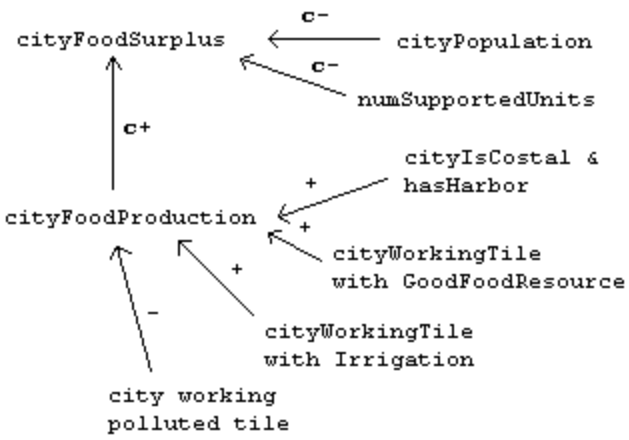

Figure 2: A portion of the city model

In practice, combining a qualitative model with an HTN planner raises some thorny issues. Whereas the HTN planner searches for decompositions of high-level tasks, a qualitative model allows the problem-solver to work backwards from the top-level goal through qualitative influences to sub-goals involving propositions to be achieved or prevented. For example, as shown in Figure 2, one way to increase a city's food surplus is to increase its food production. One way to increase food production is by working tiles that have good food resources on them, such as the wheat or fruit on the map in Figure 1. Regressing back to these sub-goals is straightforward. Operationalizing the goals takes some additional effort.

The idea is to propose actions by comparing the sub-goal propositions to the primitive task representations used by the HTN planner. One issue here is that in an HTN planner, all preconditions are treated as *filter* preconditions; that is, they do not distinguish applicability conditions from potential subgoals that could be achieved. So the first step is infer which of the goal clauses are achievable conditions and which are applicability filters. We determine this by comparing the goal clauses against the direct effects of each primitive task in the domain. For example, `(cityWorkingTileAt ?city ?location)` is achievable by the `doConvertSpecialistToWorker` primitive, whereas there is no primitive that can achieve `cityIsCoastal` for a landlocked city. Note that it is not sufficient to look only at the predicates when determining whether a clause is a filter or a sub-goal. For example, `(specialAt Irrigation ?loc)` can be achieved by assigning workers to improve the terrain, whereas `(specialAt Whale ?loc)` is a feature of the environment and not something that can be achieved.

The next step is to check the satisfaction of the sub-goal's filter preconditions to verify that the goal is applicable at all. This will prevent landlocked cities from trying to build harbors, but unfortunately won't prevent a city from creating pollution in order to clean it up. Such pathological behaviors must be caught by monotonicity constraints on sub-goaling. Verifying satisfaction of individual clauses in isolation can also be problematic because bindings may be under-specified. A clause like `(isa ?special GoodFoodSpecial)` won't filter any goal by itself. The way out of this problem is to carefully select predicates for the qualitative model that can be evaluated independently. For example, `(cityWorkingGoodFoodSpecialAt ?city ?special ?location)` embeds enough information that it can be verified on its own.

The third step in proposing actions is to associate actions with the remaining achievable goal clauses. A difficulty in this step is that the goal states that influence quantities may not directly match the effect states of primitive actions. This primarily happens when the action has a *delayed* effect. For example, a decision to build a granary doesn't immediately result in having a granary, but rather in having an entry for a granary on the planned production queue. It could be many turns later before a granary starts affecting food reserves (if ever). To deal with such delayed effects, we introduced an auxiliary predicate, `eventualEffectOfAction-Props`. A typical delayed effect looks like:

```
(eventualEffectOfAction-Props
   (doChangeProduction ?city ?obj)
   (cityHasProductionItem ?city ?obj))
```

When we search for actions to achieve sub-goals, we search both the immediate and eventual effects of actions.

In the fourth step, the set of actions for a possible sub-goal must be sequenced and their arguments bound in a rational way. This is not quite the same as achieving sub-goal preconditions. For example, the goal may be to achieve the situation in which the city is working a tile that is irrigated. This could be achieved either by finding the best tile to irrigate and then moving a worker to that tile, or finding a tile being worked and irrigating that. Since there are numerous preconditions on what can be irrigated and virtually none on moving a worker, the former strategy is to be preferred.

The above example raises another issue: if there already is some irrigated tile being worked, a traditional planner would simply decide that its goal had been achieved. However, because our goal is to *maximize* food surplus, we want to ensure that a new irrigated tile is created (or at least that a worker is moved on to an irrigated tile from a non-irrigated tile). So in this process, the null plan is disallowed as a solution.

The last step in proposing actions from the qualitative model involves sub-goaling on achievable preconditions of primitive tasks. We can do this using the HTN planner itself. This requires defining a generic complex task that achieves a primitive task by extracting its achievable preconditions, and expanding the task network in a left-recursive fashion.

Learning with a Qualitative Model

Since the ultimate aim of this research is to extend the range of learning, we intend to show that the plan synthesis mechanism described above allows us to construct more complicated plans, which can then be tested in the virtual

environment of the game. Analogical retrieval and transfer allows successful plans to be reused. Beyond this, we expect to use the qualitative city model to guide experimentation and transfer. A qualitative model can help an analogical learning system make better use of precedent cases. SME provides candidate inferences, which are projections from the base to the target of an analogy. SME does not use a domain theory in mapping or projection, but this can be applied post-facto as a reality check and to determine whether to transfer solutions or reasoning/goals.

The qualitative model can be used not only to synthesize plans but also to improve plans. A plan from a precedent case that achieved one goal may be adapted to serve multiple goals by reasoning about the influence structure. This will be especially important for learning in distant transfer problems, such as solving for novel goals to balance multiple quantities. Examples include how much should one invest in improving existing cities versus expanding the civilization by creating new cities, and allocating effort for exploration (including new technologies to improve ocean-going abilities and speed) versus expanding the economy.

Learning Qualitative Influences

If qualitative models are important for problem solving and learning, then it is only natural to wonder whether it is possible to learn a qualitative model for a new domain. To do this, we assume there is at least some information about what quantities exist, and the direction of the relationship between independent and dependent variables. Learning a qualitative model would involve refining that knowledge into more specific qualitative relations by examining the magnitudes of quantitative changes over many samples.

Learning Action Effects

In order to provide the data for learning qualitative models, it is necessary to capture and characterize the effects of actions as they are executed in the game. This is also essential for determining whether or not an action was successful and whether it should subsequently be reused as a precedent case in analogical transfer.

As mentioned previously, primitive actions correspond roughly to the packets that are sent to the game server. They are the primitive commands of the game and their representation contains minimal filter preconditions and effects. Their actual effects can be construed more broadly as the quantitative changes that percolate through the system and any indirect state changes that may occur through this chain of influences. We bootstrap the learning of these effects through experimentation strategies in which decisions are made randomly. A temporal snapshot of the case is recorded before taking the action and stored in the KB. After taking the action, another temporal snapshot case is constructed, and the “before” and “after” cases are compared using SME. The planner examines the correspondences between quantities, collects those quantities whose values have changed and records the direction of change in the case as increasing or decreasing. This can be viewed as extracting qualitative derivatives. From this information it determines whether the overall goal was satisfied (i.e., improved) or not.

The next step in this process will be to compare the qualitative derivatives over multiple cases in order to extract the direction of influence – i.e., which independent variables directly affect which dependent variables. The chain of intermediate quantities is anchored at one end by the (known) direct effects of the action taken, and at the other end by the overall objective function, the top-level performance goal. A simple version-space algorithm should allow progressively refining an initial qualitative model of the flow through quantities, assuming the relationships are monotonic and have no loops.

Experiments

We are currently in the process of performing a number of experiments to measure the performance of the learning system. We describe an example here for concreteness.

Transfer Level 1: Parameterization

The idea is to compare learning performance on tasks with and without prior training on similar scenarios. Here, similar scenarios means having the same components and configuration, but different quantitative values (i.e., "within band").

Experiment: Let the task objective be to maximize city food production while holding the number of cities constant. Train on the same map and set of cities, but with different resources available in different abundances.

Initial Conditions: Start from a saved game with 10 cities, modified by a scenario generator to randomly alter the set of resources ("specials") in each city region.

Objective Function:

```
(achieve (maximize cityFoodProduction))
```

Termination Function:

```
(terminate (numTurns 50))
```

Game Score: The average value of cityFoodProduction over all 10 cities at the end of 50 turns.

Evaluation Metric: Having learned on one set of saved games, it should start out performing better given a new set of saved games, compared to starting out initially with the new set of saved games.

Current Status and Future Work

Our game-playing agent currently plans simple exploration and city management tasks and dynamically builds case libraries for a few types of decision tasks. Action effects are extracted automatically and stored with the cases. We are in the process of integrating qualitative reasoning into the planner in order to begin synthesizing and learning higher-level plans.

We have empirically established the feasibility of analyzing the action models to distinguish filter

preconditions from sub-goal preconditions, but it is not yet clear whether it will be practical to perform this analysis at runtime or whether it would be more effective to analyze the qualitative model during off-line learning and compile out domain-specific task networks.

We are actively extending the learning capabilities of our system to support more distant transfer. An important near-term goal is to extract and reason about trends, deadlines and inflection points in concise histories (Williams, 1986). This is a critical requirement for learning the early warning signs of impending problems and learning how to compensate for them before they become severe.

We also intend to explore advice-taking as a learning mechanism. Using a qualitative model to help explain advice from a human player should help with both the credit assignment problem and generalizing the advice to other situations.

An important open problem is to strategically reason about and spawn new learning goals. The two impediments here are: 1) providing sufficiently sophisticated auto-epistemic strategies to determine where the agent's knowledge is deficient (beyond simple failure to retrieve precedents), and 2) seamlessly supporting learning strategies across games.

Ultimately, we expect the techniques developed from this effort to be applicable to the reflective control of agents in a Companion Cognitive System (Forbus and Hinrichs, 2006). Strategizing and directing agents in a game is in many ways similar to the problem of coordinating computational resources, learning from interacting with a human partner, and maintaining an episodic memory of previous reasoning sessions.

Related Work

This work is part of a larger DARPA program on Transfer Learning. Other groups are pursuing similar goals in somewhat different ways. ICARUS (Langley et al., 2005) and SOAR (Nason and Laird, 2005) have both been applied to learning in real-time game environments, using Markov Logic Networks and Bayesian techniques, respectively.

A number of efforts have focused on learning qualitative models, including Coghill et al.'s (2002) QOPH, a system that combines Inductive Logic Programming with QSIM to learn structural relations between variables. Where QOPH searches through the space of QSIM models, we are proposing to constrain the search by working forward from actions to goals one step at a time. Our effort is perhaps closest to Falkenhainer's (1990) work on PHINEAS, which used SME to construct qualitative domain theories to explain novel observed behaviors. Our approach is more data-driven than PHINEAS, which relied on a single hand-generated abstract description of behavior. PHINEAS also confirmed its theories via qualitative simulation, whereas we are testing our theories via measuring improvement in gameplay.

Other research in planning and executing includes Drummond's (1990) work in integrating planning and control. That work extended the nature of goals that could be addressed, though the TILEWORLD environment was significantly less rich than Freeciv. Qualitative models have been used in planning previously, notably Hogge's (1988) TPLAN, which compiled a qualitative domain theory into planning operators, Forbus' (1989) action-augmented envisionments, which integrated STRIPS-style actions into an envisioner, and Drabble's (1993) EXCALIBUR planning and execution system that used QP theory with a hierarchical partial-order planner. The strategy game domain is more complex than any of the domains tackled in previous efforts. Our use of HTNs in planning for strategy games is inspired by Muñoz-Avila & Aha (2004), who used HTN planning with a real-time strategy game.

Prodigy/Analogy tightly integrated analogy with problem solving (Veloso, 1994). The core means-ends analysis strategy is perhaps more amenable to inferential tasks than the kinds of optimization goals we face. Peter Stone's (2000) layered learning of complex behaviors might be applicable, though strategy games are less reactive and distributed than team soccer playing.

Other researchers have also used FreeCiv as a learning domain. Ashok Goel's group at Georgia Tech has applied model-based self-adaptation in conjunction with reinforcement learning (Ulam et al., 2005). We believe that analogy will better support distant transfer learning, and that qualitative models will permit strategic reasoning in ways that their TKML models will not.

Summary

This paper has been a progress report on our efforts towards integrating qualitative reasoning into a system for planning, executing, and learning in strategy games. We suggested that qualitative models provide an intermediate level of domain knowledge comparable to what a novice human player might start with. Our initial intuition about proposing actions by regressing through the qualitative model has proven somewhat more difficult to implement than anticipated, though clearly viable. We described the use of analogy to compare before and after snapshots in order to extract the effects of actions. Ultimately, the value of using qualitative reasoning in learning strategy games, we believe, is that it will provide a way for an unsupervised learner to acquire higher-level strategies in a wide variety of tasks that involve execution in a virtual or real world.

Acknowledgements

This research was supported by DARPA under the Transfer Learning program. We thank Phil Houk, Jon Sorg, Jeff Usher, and Greg Dunham for their programming contributions.

References

Coghill, G.M., Garrett, S.M., and King, R.D. 2002. Learning Qualitative Models in the Presence of Noise. *Proceedings of QR2002.*

Drabble, B. 1993. EXCALIBUR: A Program for Planning and Reasoning with Processes. *Artificial Intelligence* **62**(1) 1-40.

Drummond, M. and Bressina, J. 1990. Planning For control. Intelligent Control 1990: Proceedings of the 5th IEEE international symposium, vol 2. IEEE Computer Society Press, Los Alamitos, CA. pp. 657-662.

Falkenhainer, B. 1990. A unified approach to explanation and theory formation. In Shrager, J. and Langley, P. (Eds.), *Computational models of scientific discovery and theory formation*. San Mateo, CA: Morgan Kaufmann

Falkenhainer, B., Forbus, K., and Gentner, D. 1989. The Structure-Mapping Engine: Algorithm and Examples. *Artificial Intelligence* **41**(1): 1-63.

Farley, A. and Lin, K. 1990. Qualitative reasoning in economics. *Journal of Economic Dynamics and Control*, **14**(2) 465-490.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* **24**, 85-168

Forbus, K. 1989. Introducing actions into qualitative simulation. *Proceedings of IJCAI89*.

Forbus, K., and Hinrichs, T. 2006. Companion Cognitive Systems: A step towards human-level AI. To appear., *AI Magazine*, vol. 27 # 2

Freeciv, 2006. Freeciv official site. [http://www.Freeciv.org/]

Gentner, D., Structure-mapping: A theoretical framework for analogy, *Cognitive Science* **7**(2), 1983

Gentner, D., and Forbus, K. 1995. MAC/FAC: A model of similarity-based retrieval. *Cognitive Science* 14, 144-206.

Hogge, J. 1988. Prevention techniques for a temporal planner. *Proceedings of AAAI88*

Kamps, J. and Peli, G. 1995. Qualitative Reasoning beyond the Physics Domain: The Density Dependence Theory of Organizational Ecology. *Proceedings of QR95*

Kuehne, S., Forbus, K., Gentner, D. and Quinn, B. 2000. SEQL: Category learning as progressive abstraction using structure mapping. Proceedings of CogSci2000

Langley, P., Choi, D., and Rogers, S. 2005. Interleaving Learning, Problem-Solving, and Execution in the ICARUS Architecture. Technical Report, Computational Learning Laboratory, CSLI, Stanford University, CA.

Lenat, D. B. 1995. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38, 33–38.

Muñoz-Avila, H. & Aha, D. (2004). On the Role of Explanation for Hierarchical Case-Based Planning in Real-Time Strategy Games. Proceedings of ECCBR-04 Workshop on Explanations in CBR

Nason, S. and Laird, J.E. 2005. Soar-RL, integrating Reinforcement Learning with Soar. Cognitive Systems Research, **6**(1), pp.51-59.

Nau, D.S., Cao, Y., Lotem, A., and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (pp. 968-973).

Ram, A., and Leake, D., eds. 1995. *Goal-Driven Learning*, MIT Press / Bradford Books, Cambridge MA.

Stone, Peter. 2000. Layered Learning in Multiagent Systems. MIT Press, Cambridge, MA.

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

Veloso, M. 1994. Planning and Learning by Analogical Reasoning. Lecture Notes in Artificial Intelligence No. 886. Springer-Verlag Berlin.

Williams, B. 1986. Doing time: Putting qualitative reasoning on firmer ground. *Proceedings of AAAI86*.