Optimizing Strategy Game Planning Via Combining Analogical Learning with Function Learning

Will Hancock WWHANCOCK@U.NORTHWESTERN.EDU
Kenneth D. Forbus FORBUS@NORTHWESTERN.EDU
Thomas R. Hinrichs T-HINRICHS@NORTHWESTERN.EDU

Qualitative Reasoning Group, EECS, Northwestern University, 2133 Sheridan, Evanston, IL 60201, USA

Abstract

Agents often need to make decisions with limited knowledge and computational resources. Analogical retrieval has been shown to provide a natural approach to performing decision-making given these constraints. However, analogical reasoning does not handle continuous function learning well. This paper describes our progress in combining analogical generalization with functions learners to combine knowledge with continuous signal estimation. We implement these ideas in a system that learns aspects of how to play Freeciv, an open-source strategy game.

1. Introduction

In learning how to operate in a dynamic domain, it is useful to be able to reason about costs of actions. While this information alone is insufficient for developing a comprehensive strategy, it is nonetheless an important component that directly influences other signals of interest, such as future value. The issue of estimating cost is inherently a credit assignment problem, in determining what features in a world state correlate with the time required to achieve tasks. We believe that analogy can play a key role in this process. Yet, analogy by itself cannot account for other necessary processes such as quantitative estimation. In this paper, we show how analogical retrieval, along with a basic function learner, can aid in parameter estimation. With accurate cost estimation, our Freeciv agent is shown to empirically perform better within the game.

Games of this sort have many interesting properties that make them ideal testbeds for cognitive agents. They are *constructive* dynamic domains, i.e. entities and systems of multiple types must be created, making them more complex than games like chess and go. They involve incomplete knowledge of the world, they entail complex relations between actions and observable world state, and goals may be continuous rather than discrete states to be achieved. Thus, it is difficult to estimate whether a decision will be beneficial in the long run. Even given the simplified problem of estimating how long an action will take, there are still many factors that can affect the outcome. Barbarians may show up randomly and destroy a city, effectively ensuring any goals pertaining to that city will never be achieved. There are hundreds of variables to consider at any particular time, making correlative learning difficult. We approach this problem by narrowing down the potential

space of influence using analogical retrieval. This enables more tractable and less noisy parameter estimation for any given action.

1.1 The Freeciv Domain

Freeciv is an open-source turn-based strategy game based on Sid Meier's series of Civilization games. The objective of the game is to defeat one's opponents either by military conquest or by winning a space race. In either case, success in the game requires one to quickly develop technological advancements to get a leg up on opponents. This requires careful resource management and planning. For example, the first decision a player usually makes is where to build the first two cities. The terrain in the vicinity of any particular city will greatly affect its characteristics throughout the game. It is advantageous to build on tiles that will promote city growth and production. Cities with good production capabilities can build new settlers faster, therefore increasing the size of the civilization faster as well. However, one should minimize the distances between cities for defensive purposes but maximize distances for trade purposes. Cities with food production bonuses on their tiles will increase their populations quicker. Tiles with high food production typically have defensive disadvantages. Coupled with unknown terrain early in the game, there is a complex optimization decision to be made that will affect how a civilization will grow for the rest of the game.

There are a number of variables to take into account when strategizing. The high level objective of the game is to balance resource spending amongst many different subgoals. How this is achieved is up to the player. Scientific research is a central tenet of the game, allowing more powerful military units as well as city improvements. Research capability effectively scales with the number of cities in one's empire, and with specific improvements that cities can build, such as libraries and research labs. Ill effects imposed by the game rules begin to manifest themselves when civilizations become too large.

Levels of attention are important for strategizing. Civilization wide decisions such as tax/science rates and government types must be mixed with unit goals of terrain improvement, along with individual city goals. Improvements are built within a city to maximize its potential. Coastal cities can build harbors to increase food production on water tiles. Cities benefit more or less from improvements based on their surrounding tile features. It makes sense to build banks in cities that are already producing relatively higher amounts of gold. Likewise, libraries, which double science output. make sense for cities with already high science outputs.

On a high level, one is constantly monitoring to make sure that citizens are happy, territory is well defended, cities are growing at a reasonable pace, research is progressing, diplomacy is being pursued, resource allocation is being balanced well, and so on. Juggling these interdependent activities is what makes Freeciv such an interesting testbed for reasoning and learning.

Freeciv has been used by other AI researchers as well. Branavan et al. (2012) explored using Monte Carlo simulation and text analytics to construct a heuristic evaluation function. While it played well on a small subset of the game (smaller map, games ended at 75 turns), it required many trials to learn the game and used the game engine to do lookahead search while playing, a tactic which is not available for most domains. It also did not construct an inspectable model of causality in the domain, unlike our learned qualitative models. Ulam et al. (2008) investigated combining metareasoning and reinforcement learning for the subtask of city defense in Freeciv. While it uses model-based reasoning, the quantitative model it uses is constructed by hand, by contrast with our automatically learned qualitative models.

1.2 Our approach

We aim to combine analogical generalization with function learners in a novel way. We hypothesize that this will address the curse of dimensionality regarding learning outcomes of decision making.

There is currently a large amount of knowledge within our game playing agent. Previously, we have learned qualitative models of domain physics, both by demonstration (Hinrichs & Forbus, 2012) and by reading (McFate et al. 2014). While incomplete, these models cover a large portion of the qualitative causal relations in the domain. Also ontologized are constraints on actions; cities are capable of building improvements but cannot move on the map like units can.

This knowledge already goes a long way in aiding an agent to make decisions in the game. It also allows for powerful reasoning in explaining an agent's decision-making process, an ability lacking in many model-free reinforcement learning techniques. While the qualitative model encodes causal relations on a high level, there are many instances in which it is beneficial to learn functions of parameters in the model. Take a simple example of learning the cost of moving a unit to a destination. This is a simple linear function mapping the unit's move points per turn and the distance to the destination to turns required. Pure analogical retrieval will not help to learn this information. It is therefore beneficial to implement an underlying mechanism to learn functions within any generalization. However, analogical induction can go a long way in taking existing domain knowledge and segmenting action outcomes into distinct groups, therefore adding commonsense knowledge and decreasing the complexity required of the function learner.

1.3 Experimentation

While analogy can be a powerful technique for learning, there must first be cases to retrieve and compare. To bootstrap a case library and ensure a variety of cases, we draw from a library of games played by human agents. The games are recorded as a series of actions, so that they can be replayed on demand. Cases for learning goals are then generated on the fly while replaying a game and saved for future retrieval. This is effectively a transfer learning task. Our focus specifically for this work is to learn the costs of actions and use this information to aid in decision making. We evaluate our results by quantitatively maximizing the number of cities across our empire.

2. 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 location or telling a city what to build. Complex tasks are at the level of figuring out what a unit should do during its 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. This is a simple heuristic that checks for explicit negations and for inconsistent values of functional predicates.

3. Qualitative Modeling

A qualitative model is a partial domain theory that captures influences between quantities. One of the outstanding features of a game like Freeciv is the complexity of quantitative relationships in the game engine. Understanding these relationships is key to playing the game well. Previously, we have learned a partial qualitative model of Freeciv game physics by demonstration (Hinrichs & Forbus 2012).

The primary way the qualitative model is used is to determine which changes incurred by an action correspond to goals and whether those changes signify success or failure. The model is encoded with high level goals, and leaf nodes corresponding to primitive quantities such as the number of military units, or the food production of a city. Using this model, we can infer facts about the game domain; e.g. building a library increases science production, or irrigating a plains tile increases food production. Together with the planner, this model can be used to achieve abstract goals. As the complexity of the model grows, so does the number of ways in which a goal can be satisfied. A goal to increase the size of a civilization can be achieved by manufacturing settlers to found new cities, or to conquer opponents' existing cities. The mechanism of learning goal costs in this paper helps this decision-making process.

There can be many methods to achieve any given goal, and many goals can be active at the same time. Multiple active goals can conflict with each other. There are inherent tradeoffs when making strategic decisions. For example, one must explicitly allocate taxation, happiness, and science output on an empire-wide level. One can set science output to 80%, taxation to 20%, and happiness to 0. Research will move quickly, but cities will be more prone to revolution. We explicitly set high level goals (e.g. maximizing city size) and reason over the qualitative model in order to generate sub goals and methods to achieve these sub-goals (Hinrichs & Forbus, 2012).

4. Analogical Learning

A goal of this research is to demonstrate how analogy and quantitative 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; Forbus et al. 2017), the MAC/FAC retrieval system (Forbus et al., 1995), and the SAGE generalization system (McLure et al. 2015) as core components. These analogical mechanisms are tightly integrated in the underlying

reasoning engine and provide the mechanisms for retrieval, comparison, and transfer. The Structure Mapping Engine not only assesses the similarity of a precedent to the current situation, but also projects the previous solution into the new case by translating entities to their mapped equivalents, in the form of candidate inferences. Thus, analogy provides the first level of adaptation automatically. The unit of comparison and retrieval is a case, and in this approach, cases 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 an entity at a moment in time in a given game. For example, cases can capture a decision about what improvements to build, what tiles to work, and 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 later used to learn action costs, via analogical generalization. That is, each decision has a SAGE *generalization pool* which maintains a set of automatically constructed generalizations and outliers. A generalization pool is also a MAC/FAC case library, so adding a new case causes the most similar item to be retrieved. If the degree of similarity is over the *assimilation threshold*, then the new case is assimilated into the retrieved item. This assimilation process produces a new generalization, if the retrieved item was an outlier, or if the retrieved item was a generalization, updates it with the contents of the new case. Every statement in a generalization has a probability based on the fraction of cases it was true in, and non-identical entities are replaced by more abstract entities. As the planner attempts to learn decision tasks, it creates and populates generalization pools for each type of task. After executing a plan, the current relevant facts are queried and stored as a temporal sub-abstraction in the game context. When the result of the action is evaluated, the case is added to the task-specific library.

5. Learning Costs of Achieving Goals

There are a finite number of primitive actions that one can take in Freeciv. Cities have different action sets than units, and there are combat units which can fight, and worker units which cannot. Actions taken vary in the amount of time required for completion. A simple example is that of movement: the cost of moving a unit from one location to the other is roughly linear in the number of tiles required for moving. Another example is the irrigate action. Irrigation takes longer to complete for marsh tiles vs plains tiles. Our goal is to learn what game states affect the cost of any action.

Our current strategy implementation utilizes a learned goal network for planning. There can be multiple paths to satisfying a goal, but currently no way of deciding how to prioritize one path over the other. Ideally, we would like to know the value function for a state → action transition. Freeciv is a particularly interesting domain because of the magnitude of its decision space. Our work in this paper aims to mitigate the curse of dimensionality by looking at a simplified problem: estimating the required time for series of actions. We hypothesize that this signal will allow planning prioritization by preference ordering over multiple methods of achieving the same goal.

We specifically look at cost of goals rather than atomic actions to allow for higher level abstract reasoning. Given the goal of winning the game, and a sufficiently specified plan to achieve it, we could theoretically estimate the cost. However, even barring the uncertainty of military invasion and the randomness of the game, it would not be useful to reason at this level. We also do not want

all goals to ground in single primitive actions, as reasoning at this level would also not be useful for general strategizing. For this work, we limit time estimation to goals involving individual units and cities, but there is no reason why we could not also use this system for other civilization-wide goals such as scientific research achievement.

We model the cost of a goal as a linear combination of the costs of its constituent primitive actions. The planner is queried with the specific goal and returns a list of methods for achieving it. For each method, we extract any primitives for which we have accrued case libraries. Each action's cost is estimated and added to the total. The set of methods for achieving the goal is ordered by this estimated time, and the cheapest method is executed by the agent.

One goal that often arises is that of increasing city food production. A method to satisfying this is to assign a worker unit to irrigate farmland around the city. This involves two primitive actions: moving the unit to a location, and the actual act of irrigation by the unit. Given candidate locations to irrigate, we can estimate temporal cost in a general way using techniques from this paper and minimize cost. In this case, this entails minimizing distance as well as choosing appropriate terrain on which to perform the action.

5.1 Choosing Features for Case Generation

One simple way of approaching this problem would be to train a function learner on all game features for every action we wish to estimate a cost for. This model would be heavily prone to noise due to the massive number of features present (most of which would be irrelevant). The search space would also be highly dimensional, requiring an appropriate amount of computing power. On the other hand, we do not want to manually map each action to its relevant features, as this discourages general learning. We instead are inspired by a commonsense model of attention, following (Bello & Bridewell, 2017).

The solution that we have adopted is somewhere between these two extremes. We use a set of plausible commonsense classes: cities, buildings, map tiles, units. We explicitly map between actions and these commonsense classes. This is a one-to-many mapping: for example, the doIrrigate action's case will include information on map tiles and units. Each commonsense class in turn maps to a set of features that describe it. The city class, for example, lists its buildings, its resource outputs (production, taxation, food, etc.), its size, the number of units associated with it, etc. A case for a map location lists information such as the terrain type, any specials, and any present units. In this way we generate a constrained set of features. Note that this mapping does not guarantee correlation between all features and cost outcomes. However, it serves as a preliminary filtering step, greatly simplifying the cost estimation problem.

For the irrigation action, the set of features to analyze draws from map tile information as well as unit information. Our planner binds the actor variable to a unit capable of irrigation, thus giving us a concrete entity on which to extract unit information. A case for irrigate will contain this unit's type, its combat status, and its home city. The case will also contain map tile information such as tile type, and whether it contains any specials such as food or production bonuses.

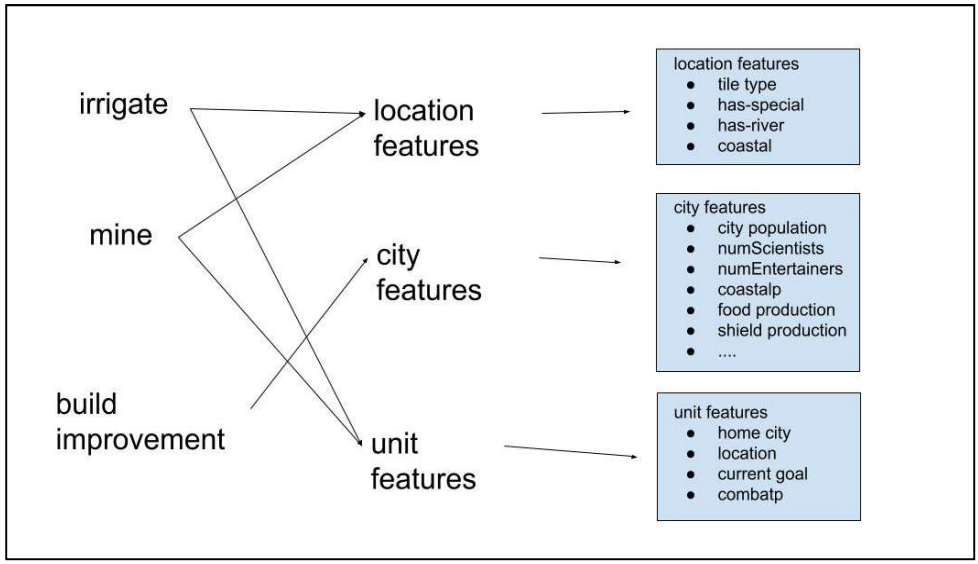

Figure 1. Commonsense feature mapping for primitive actions.

5.2 Analogical Generalization

To gather cases, we have stored human-played games which our system is able to replay. When an action that we wish to monitor is replayed, we take a snapshot of its relevant features. When the action has completed, we take another snapshot and store these before/after snapshots into a single case. The number of turns taken for the action is also stored within the case. To build our case-library, we draw from ten previously played games, and record cases for a subset of actions in the game. These actions are: irrigate, mine, goToDest, buildRoad, changeProduction, research, and buildCity. The research action is the act of starting and discovering a scientific achievement. Change production refers to the action of a city starting and completing an improvement, e.g. building a granary.

;;; Case (FC-CommandOutcomeCaseFn (CommandFn fc-game10 123) fc-game10)
(in-microtheory (FC-CommandOutcomeCaseFn (CommandFn fc-game10 123) fc-game10))
(currentActivity Unit-108 FC-ActivityIrrigate)
(fcObjectAt Unit-108 (FreecivLocationFn 36 45))
(infoTransferred (CommandFn fc-game10 123) (doIrrigate Unit-108 (FreecivLocationFn 36 45)))
(isa (FreecivLocationFn 36 45) (SubcollectionOfWithRelationToFn FreecivLocation terrainAt FC-Terrain-Plains))
(isa Unit-108 (SubcollectionOfWithRelationToFn FreeCiv-Unit unitType FC-Unit-Workers))
(timeElapsed (Turns 4))
(sovereignAllegianceOfOrg Unit-108 FC-Nation-American)

Figure 2. An example case for the irrigate action.

Our agent is then able to use this information in strategizing. When attempting to achieve a goal, the planner generates a sequence of actions. For each action, a snapshot of the current game state is taken for relevant features. This is turned into a case and is classified by SAGE into a generalization for action outcomes. This can be seen as a form of supervised learning.

An assumption that this system makes is that similar case features for an action will correlate with similar time requirements to complete that action. To give a concrete example: the irrigation action takes varying amounts of time based on the tile type. Plains tiles take four turns to irrigate, while marshland takes seven turns. Since our cases for irrigation contain this information, we expect that we may see a clustering threshold along this feature, effectively partitioning the cases into two distinct time categories. Another potential example involves an enemy within a city's range causing longer production times due to the city not being able to use resources from that tile.

It should be noted that analogical generalization is not guaranteed to separate along significant features. SAGE, as any inductive generalizer, is susceptible to noise. The home city for a unit has no correlation with how long it will take that unit to perform an action but is included as a feature. This does not eliminate the possibility of a noisy signal coming through, for example if by random chance all units from the same home city correlate with a lower than average time for an action. While this is a potential problem, it is mitigated primarily because our selective feature mapping described in (5.1) reduces spurious features from arbitrary correlation. Incidentally, it also generally minimizes the size of generalization pools to learn from, maintaining the ability for the function learner to dynamically estimate time costs while playing, eliminating the need for an explicit train/test cycle. More training data will alleviate noise, but we argue that our system requires less relative to other non-analogical systems.

5.3 Function Learning

While analogical generalization is powerful, it lacks the ability to generalize on continuous features. One could theoretically quantize continuous signals, but we find this to be outside the scope of this work. We instead choose to use a simple function learner to infer costs from within generalizations. We have chosen to use a simple least-squares linear regression estimator as the basis for this layer. Currently, we learn on all numeric features of a case, but could easily generalize to non-numeric features as well. We fit this estimator to a learned time cost for each case.

There are many available and proven function learners currently in existence. They vary in expressivity, speed, explainability, etc. In this work, our primary goal is to augment analogical learning, not subsume it. We hope in the future to ontologize knowledge learned from this base layer, though we do not address this task within this work. Regression fits our requirements quite well. It has been shown to perform acceptably with small amounts of data relative to other learners (Forman & Cohen, 2004). We are sampling the Freeciv game state for our training data and so theoretically can build quite large datasets by playing many games. However, we do not want to operate under this assumption, and certain relatively infrequent actions may take quite some time to develop large generalization pools.

We also would like to learn online, as in game experience can be directly incorporated into our reasoning system in real time. Human players do not require explicit train/test cycles in order to improve. The linearity assumption of regression means that training requirements will scale reasonably as our pools grow larger.

5.4 Preference Ordering

The learned time outcomes for actions are reincorporated into decisions made in the game. In a given turn, our goal reasoner generates a set of active high-level goals. For this work, we filter goals that deal with individual units and cities. A city might have an active goal to increase its food production. We invoke our HTN planner to generate a set of candidate methods to accomplish this. We estimate temporal cost for each candidate plan and invoke the series of actions with the lowest cost.

6. Experimentation

Twenty games of 70 turns were played both with and without time learning. Ten pre-defined scenarios were used to minimize randomness in the game. Game maps are generated randomly and can vary to some extent in their topological features. One's starting units can be placed on a small island, limiting the number of cities that can be built. They can also be placed on a large continent with enemy civilizations, making an early demise likely. To minimize this randomness, we played the control/knowledge trials on the same set of saved maps. Enemies were removed from the game, as combat is random. Huts were also removed due to the fact that they can randomly give extra cities to one's empire. Land to water ratio was increased from 3:7 to 4:6 to increase potential city site area.

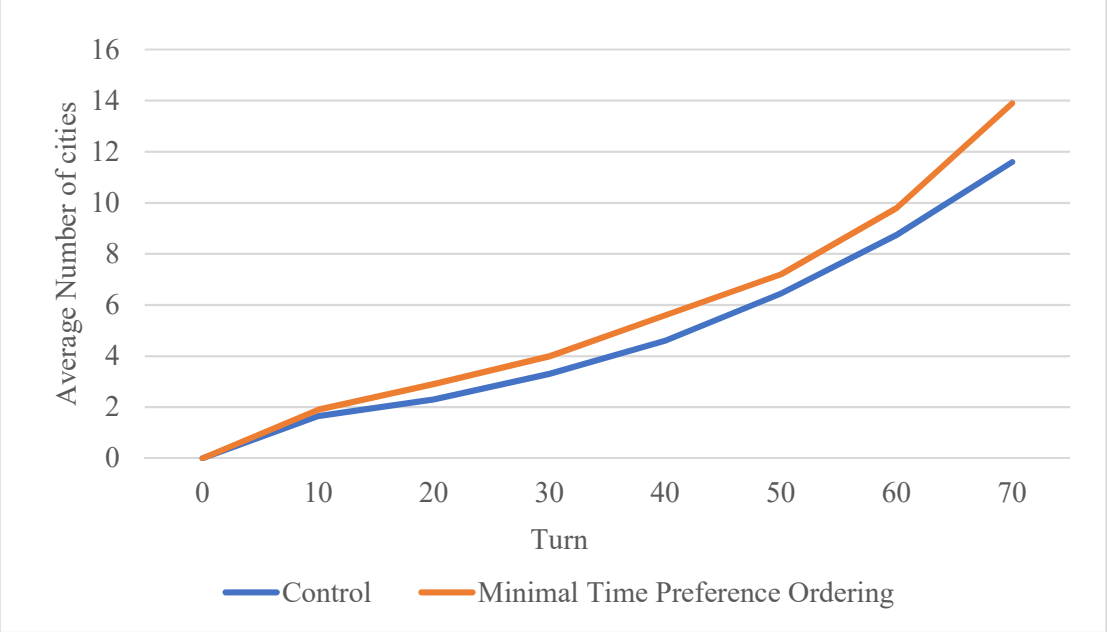

Figure 3. Average number of cities over 70 turns.

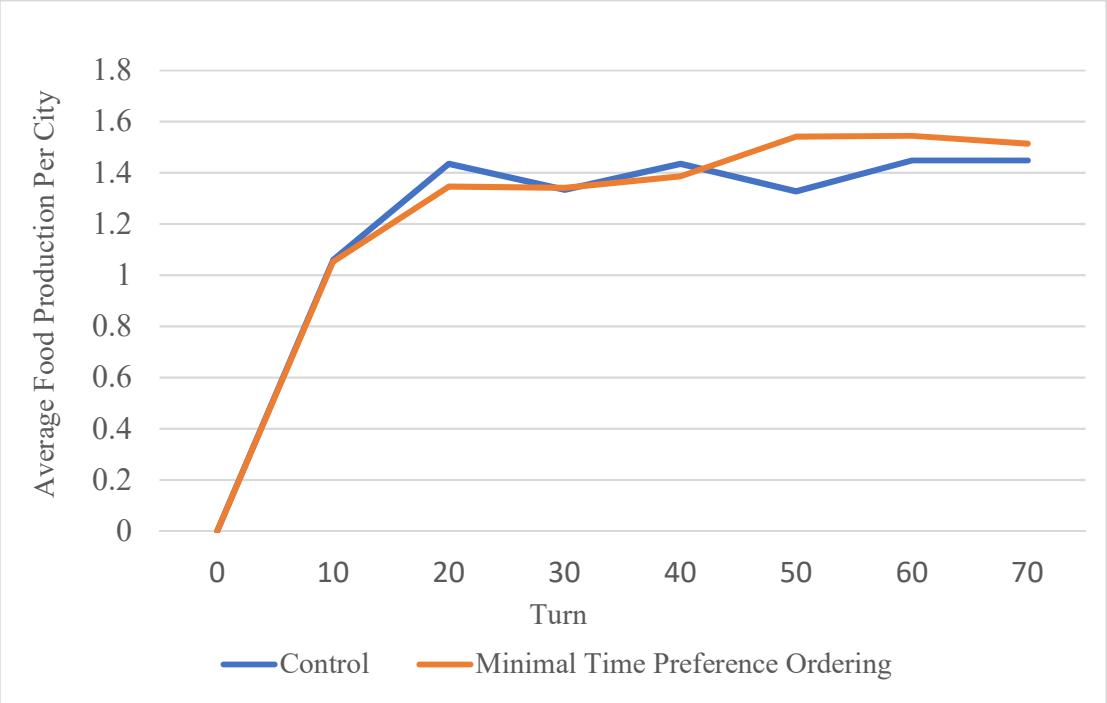

Figure 4. Average Food Production per city over 70 turns.

7. Results

The main metric used to evaluate results was the number of cities built after 70 turns. We averaged this over each of the ten games played. There were 16.5% more cities created with temporal preference ordering than in the control trials. Qualitatively, it is apparent that the growth in both cases is exponential: each city built has an ability to found more cities. The quicker that one accomplishes this, the faster that a civilization will grow. We also measured average food production per city after 70 turns. The results here are not so clear; and qualitatively these results are harder to interpret. To found a new city, one must build settlers. When this happens, the city that builds the unit has its size reduced by 1. This automatically decreases food production for that city. The result is that in order to expand, one must take the momentary hit of reduced food production. Here we begin to see a small optimization problem at play. Thus, food production decreases for the knowledge trial after turn 50, due to expansion. This is not seen in the control trial, perhaps because cities are not being built at the same rate as the knowledge trial cities.

A 2-sample t-test analysis on the number of cities built in the temporal preference ordering vs control trials yielded a p-value of .0397. The correlation between average food production over time in the two trials was insignificant.

8. Discussion

A main goal of this work is to explore how structured knowledge can be used effectively in combination with primitive signal learners. The addition of knowledge reduces the complexity of causal attribution, allowing for reduced computational requirements as well as more accurate models.

We show that analogical generalization can be used to learn action outcomes. By clustering on the beginning and end states of an action, we work towards learning significant features that affect the amount of time required for that action. The time required for irrigating a tile depends on the tile type. It takes longer to irrigate swamp than plains. By pre-clustering in this way, we aim to reduce the computational requirement and increase the accuracy of the function learner. Our tile example is quite simple, but our method can theoretically scale to more abstract concepts, as long as they are ontologized. One can imagine how any action could be affected by a state of war. Resources may be diverted, units might be lost in battle, causing actions to take longer than usual. While we have not seen this scenario due to constrained game rules, our system as it stands seems capable of temporal estimation in these types of scenarios.

Furthermore, we show that learning temporal costs of actions positively affects strategizing. By preference ordering methods to achieve goals, our system achieves milestones faster. Action duration estimation may not seem to be immediately useful in a strategic context such as this one. The fact that one action takes less time than another may not be relevant if the value of the cheaper option is low. Long term plans need to be considered. We may want to spend twice the time to develop a research goal if it allows us to build a stronger military unit. Collective goal achievement is also important. It may be more efficient to have multiple cities work in parallel to more quickly achieve a desired outcome.

Our underlying goal-driven planner goes a long way to mitigate these issues. By reasoning about multiple goal tradeoffs on a higher level, we avoid forcing ourselves into greedy local maxima. Maximizing the number of cities in our civilization is one of these higher-level goals. Perhaps the fastest way to achieve this is to expand with no resources spent on defense. This works well with an isolated civilization but being able to switch to a defensive footing when enemies are spotted becomes important. A safer approach is to pursue defense goals in parallel. Managing these tradeoffs is handled by the goal reasoner (Hinrichs & Forbus, 2012), and so we depend on an appropriate level of goal abstraction for temporal preference ordering to be effective.

8.1 Related Work

The General Game Playing effort (Pell 1993; Genesereth & Thielscher 2014) seeks to create systems that learn arbitrary games from obfuscated predicates, to stress-test the analytic capabilities of systems. Although GGP with some restrictions can be used in transfer learning research (e.g. Hinrichs & Forbus, 2011), predicate obfuscation introduces an artificial complexity. Our goal instead is to be able to transfer knowledge both between games and between games and real life. Many games also share higher level conceptual requirements, such as win scenarios, opponents, strategy, etc. Transferring knowledge from one game domain to another shortens learning time and compresses knowledge space requirements. A long-term goal for this research is to develop qualitative models general enough to use in other game domains. Aha et. Al (2009) use a case-based transfer learning method to strategize in a simulated football game, though the action space is much more constrained than Freeciv.

There has been other research that uses Freeciv. Ulam et al. (2008) aimed to combine domain models with reinforcement learning to improve agent performance. Specifically, the agent was trained to defend cities against attackers. The domain model was compositional, in that a high level goal could be decomposed into subtasks. The reinforcement learner could then focus on a specific subtask, reducing the search space. The spirit of this paper is similar to ours, in using knowledge to constrain more primitive inference. However, their model was hand constructed, while ours was learned.

9. Future Work

While we have shown that combining analogy with function learning can be useful, some of the information that we have learned might be difficult for our system to explain. One issue with a function learning approach is that of data interpretability. Linear regression can accurately infer the costs of actions, but cannot easily attribute action costs to specific features. We may be able to predict that irrigation will take seven turns in swamp land, but we may not know which feature is relevant, e.g. tile type vs tile specials. Further work is required to make these types of causal relations explicit in our ontology. This explicit causal model building can help to augment our existing qualitative model.

An interesting avenue to explore is the cross-domain transfer of knowledge learned in Freeciv to other strategy games, or to situations from real life. The specific functions in terms of units such as turns and tiles are unlikely to transfer, but ordinal relationships often would (e.g. tanks travel faster than horses, which travel faster than people on foot).

Another intention of this work has been to model high-level goals. One issue with this task is the problem of uncertainty. As a goal becomes more abstract, it also becomes less clear how exactly it should be reached. Achieving the technology of electricity takes at least 100 turns and many decisions. Our civilizations may have been at war, etc. It is inherently difficult to estimate the time cost of an underspecified series of actions. We hope that as we build cases for more game scenarios, we will be able to address more and more abstract goals. We have not explicitly quantified this in our current research but have kept the possibility in mind.

Acknowledgements

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

References

Aha, D. W., Molineaux, M., & Sukthankar, G. (2009). Case-based reasoning in transfer learning. In *International Conference on Case-Based Reasoning* (pp. 29–44). Springer.

Bello, P., & Bridewell, W. (2017). There Is No Agency Without Attention. *AI Magazine*, *38*, 27.

Branavan, S. R. K., Silver, D., & Barzilay, R. (2012). Learning to Win by Reading Manuals in a Monte-Carlo Framework. *Journal of Artificial Intelligence Research*, *43*, 661–704.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, *41*, 1–63.

Forbus, K. D. (1984). Qualitative process theory. In *Qualitative reasoning about physical systems* (pp. 85–168). Elsevier.

Forbus, K. D., Ferguson, R. W., Lovett, A., & Gentner, D. (2017). Extending SME to Handle Large-Scale Cognitive Modeling. *Cognitive Science*, *41*, 1152–1201.

Forbus, K. D., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, *19*, 141–205.

Forman, G., & Cohen, I. (2004). Learning from little: Comparison of classifiers given little training. In *European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 161–172). Springer.

Genesereth, M., & Thielscher, M. (2014). General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *8*, 1–229.

Hinrichs, T., & Forbus, K. D. (2011). Transfer learning through analogy in games. *AI Magazine*, *32*, 70–83.

Hinrichs, T. R., & Forbus, K. D. (2007). Analogical Learning in a Turn-based Strategy Game. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (pp. 853–858). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hinrichs, T. R., & Forbus, K. D. (2012). Learning Qualitative Models by Demonstration. In *AAAI*.

Hinrichs, T. R., & Forbus, K. D. (2014). X goes first: Teaching simple games through multimodal interaction. *Advances in Cognitive Systems*, *3*, 31–46.

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

McFate, C. J., Forbus, K. D., & Hinrichs, T. R. (2014). Using Narrative Function to Extract Qualitative Information from Natural Language Texts. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (pp. 373–379). Québec City, Québec, Canada: AAAI Press.

McLure, M. D., Friedman, S. E., & Forbus, K. D. (2015). Extending Analogical Generalization with Near-Misses. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9803

Paritosh, P. K., & Klenk, M. E. (2006). Cognitive processes in quantitative estimation: Analogical anchors and causal adjustment. In *the Proceedings of the 28th Annual Conference of the Cognitive Science Society, Vancouver*.

Pell, B. (1993). *Strategy generation and evaluation for meta-game playing*. Citeseer.

Ulam, P., Jones, J., & Goel, A. K. (2008). Combining Model-Based Meta-Reasoning and Reinforcement Learning for Adapting Game-Playing Agents. In *AIIDE*.