Beyond the Rational Player: Amortizing Type-Level Goal Hierarchies

Thomas R. Hinrichs T-HINRICHS@NORTHWESTERN.EDU
Kenneth D. Forbus FORBUS@NORTHWESTERN.EDU
EECS, Northwestern University, Evanston, IL 60208 USA

Abstract

To what degree should an agent reason with pre-computed, static goals? On the one hand, efficiency and scaling concerns suggest the need to avoid continually re-generating subgoals, while on the other hand, flexible behavior demands the ability to acquire, refine, and prioritize goals dynamically. This paper describes a compromise that enables a learning agent to build up a type-level goal hierarchy from learned knowledge, but amortizes the cost of its construction and application over time. We present this approach in the context of an agent that learns to play the strategy game Freeciv.

1. Introduction

One of the premises of goal reasoning is that explicit, reified goals are important for flexibly driving behavior. This idea is almost as old as AI itself and was perhaps first analyzed and defended by Alan Newell (1962), yet it is worth revisiting some of the rationale. Explicit goals allow an agent to reason about abstract future intended states or activities without making them fully concrete. A goal may be a partial state description, or it may designate a state in a way that is not amenable to syntactic matching (e.g., to optimize a quantity).

A challenge arises when a domain is dynamic and involves creating new entities on the fly, and therefore potentially new goals, or simply involves too many entities to permit reifying propositional goals. For example, in playing Freeciv[1], new cities and units are continually being created and it is not possible to refer to them by name before they exist. Moreover, each terrain tile in a civilization may be relevant to some goal, but it is infeasible to explicitly represent them all. In short, static goal trees can be inflexible and reified propositional goal trees can be prohibitively large. Goal representations may not be amenable to matching, and dynamically inferring subgoals can be expensive.

These challenges have led us to a compromise solution with six parts:

1. Elaborate a goal lattice from a given performance goal and a learned qualitative model.
2. Represent goals at the type level.
3. Indexicalize goals for reuse across game instances.
4. Reify denotational terms (names) for goals to reduce computational cost of subgoaling.
5. Identify and reify goal tradeoffs.
6. Index goals and actor types by capability roles.

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

Given this information, a very simple agent can efficiently make informed decisions with respect to goals. We call such an agent a *rational player*. A rational player has the property that its actions can be explained or justified with respect to a hierarchy of domain goals. This is not a planner per se, because it does not necessarily project future states of the world, but a rational player prefers actions that it believes will positively influence higher-level (and therefore more important) goals. As we progress beyond the rational player, this weak method for deciding what to do can be incrementally augmented and overridden by learned knowledge that looks more like task-decomposition plans and policies for choosing among tradeoffs.

This paper presents our type-level goal representation and describes how it is produced and used by a rational player in the context of learning to play Freeciv, and previews our extension to a more reflective player.

2. The Freeciv Domain

Freeciv is an open-source implementation of the popular Civilization series of games. The main objective is to build a large civilization over several simulated millennia and conquer all other civilizations. This involves many types of decisions, long-term goals, and tradeoffs. Examples of some of these types of decisions can be seen in Figure 1.

There are clear challenges in learning a game like this. The sheer diversity of activities, complexity of the world model, uncertainty of adversarial planning and stochastic outcomes, incomplete information due to the "fog of war", and dynamic nature of the game makes it fundamentally different from games like chess or checkers.

Some salient features of this game are that many actions are durative, actions have delayed effects, and the game simulates a number of quantitative systems. This makes it relatively easy to model in terms of *continuous processes* (Forbus, 1984). In fact, one of our hypotheses has been that by learning a qualitative model of the game, a simple game playing agent should be able to exploit the model to achieve flexible behavior in far fewer trials than it would need using reinforcement learning. An additional benefit of an explicit model is that it can be extended through language-based instruction or by reading the manual, and its behavior is explainable with respect to comprehensible goals.

Figure 1: Freeciv map, typical production decisions, and partial technology tree

3. Representing a Reified Goal Lattice

We assume that games are defined in terms of one or more top-level goals for winning along with a system of rules. We also assume that there is a set of discrete primitive actions with declarative preconditions, plus an enumerable set of quantity types whose value can be sampled. Beyond that, we assume almost nothing about the domain. Initially, therefore, there is no goal lattice, no HTN plans, or goal-driven behavior. The system must learn to decompose goals to subgoals by observing an instructor's demonstration and inducing a qualitative model of influences between quantities and the effects of actions on quantities. This directed graph of influences is translated into a lattice of goals. So for example, given a positive influence between food production on a tile and food production in a city, a goal of maximizing city food production is decomposed into a subgoal of maximizing food production on its tiles. This lattice-building process starts with a high-level goal and works down, bottoming out in leaves of the influence model. Notice that the construction of this goal lattice does not suggest that the system knows how to *pursue* any particular goal or goal type. It merely provides a general way to search for subgoals that might be operational – there might be a learned plan for achieving a goal or a primitive action that has been empirically discovered to influence a goal quantity.

As we talk about goals in this paper, we will occasionally refer to *performance* goals, as opposed to *learning* goals. Performance goals are domain-level goals that contribute to winning the game. Learning goals, on the other hand, are knowledge acquisition goals that the agent posts to guide experimentation and active learning. In addition, the performance goals in the lattice can be further partitioned into *propositional* goals (achieve, maintain, or prevent a state) and *quantitative* goals (maximize, balance, or minimize a quantity). Identifying subgoal relations between propositional and quantitative goals involves a combination of empirical and analytical inferences. In this section, we describe how this works and how these goals are decomposed and represented.

3.1 The learned qualitative model

A key idea is that subgoal decompositions can be derived directly from a learned qualitative model of the domain (Hinrichs & Forbus 2012a). The system induces qualitative relations in the game by observing an instructor's demonstration and tracking how quantities change in response to actions and influences over time, in a manner similar to BACON (Langley et al., 1987). Influences are induced at the *type-level* using higher-order predicates (Hinrichs & Forbus 2012b). For example:

```
(qprop+TypeType (MeasurableQuantityFn cityFoodProduction)
                (MeasurableQuantityFn tileFoodProduction)
                FreeCiv-City FreecivLocation cityWorkingTileAt)
```

This represents the positive indirect influence between the food produced at a tile worked by a city and the total food produced by the city. This second-order relation implicitly captures a universal quantifier over all cities and locations related by the `cityWorkingTileAt` predicate. We refer to this as a type-level relation because it does not reference any instance-level entities such as particular cities or locations in the game. In addition to influences, we also induce qualitative *process limits* by looking for actions and events that bookend monotonic trends.

3.2 Building the Goal Lattice from the Qualitative Model

Given a model of type-level qualitative influences and a performance goal, such as maximizing food production in cities, it is straightforward to infer that one way to achieve this is to maximize the food produced on tiles worked in cities. This inference is more complex when the parent goal is quantitative and the subgoal is propositional, or vice versa. In the former case, the influence model may contain a *dependsOn* relation, which is a very general way of saying that a quantity is influenced by a proposition, for example, the food production on a tile depends on the presence of irrigation. Our system currently learns this through language-based instruction.

The opposite case is more complex. What does it mean for a propositional goal to have a quantitative subgoal? We've identified two ways this happens: maximizing the *likelihood* of achieving or preventing the proposition and maximizing the *rate* of achieving it. Our current implementation focuses on subgoaling via the rate. For example, given a propositional goal such as `(playerKnowsTech ?player TheRepublic)`, it identifies this as the outcome of a durative action whose process is directly influenced by the global science rate (learned empirically). Maximizing this will achieve the proposition sooner, and is therefore a subgoal.

We found such inferences became unacceptably slow when they are re-computed for each decision and transitively propagated through the influence model. To avoid repeatedly instantiating intermediate results, we translate the influence model into an explicit goal lattice offline, store it with the learned knowledge of the game, and elaborate it incrementally as the learned model is revised. A partial goal lattice for Freeciv is shown in Figure 2.

3.3 Representing goals at the type level

Because type-level goals are not tied to particular entities that change across instances of the game, they remain valid without adaptation. For example, a goal to maximize surplus food in all cities could be represented as:

```
(MaximizeFn
 ((MeasurableQuantityFn cityFoodSurplus)
  (GenericInstanceFn FreeCiv-City))
```

Here, `GenericInstanceFn` is used to designate a skolem to stand in for instances of a city, as if there were a universally quantified variable.

3.4 Indexical Goal Representations

One problem with the goal above is that it does not restrict the cities to those owned by the current player whose goal this is. The goal representation must relate everything back to the current player, except that the player's name changes in each game instance. The player may be called Teddy Roosevelt in one game and Attila the Hun the next. Consequently, we introduce an *indexical* representation for the current player as:

```
(IndexicalFn currentPlayer)
```

where currentPlayer is a predicate that can be queried to bind the player's name. Now, instead of our skolem being simply every city, we must construct a specification that relates the subset of cities to the indexical:

```
(CollectionSubsetFn FreeCiv-City
  (TheSetOf ?var1
   (and (isa ?var1 FreeCiv-City)
        (ownsCity (IndexicalFn currentPlayer)?var1))))
```

This subset description is constructed by recursively walking down from the top-level indexicalized game goal through the type-level qualitative influences. These influences relate the entities of the dependent and independent quantities. These relations are incrementally added to the set specification to form an audit trail back to the player. Thus the translation from model to goal involves no additional domain knowledge.

3.5 Denotational Terms for Goals

The fully-expanded representation for a simple quantity goal starts to look rather complex. We found that simply retrieving subgoal relationships at the type level suffered from the cost of unifying such large structures. As a consequence, we construct explicit denotational terms for type-level goals and use these more concise names to capture static subgoal relationships. The `goalName` predicate relates the denotational term to the expanded representation:

```
(goalName (GoalFn 14)
  (MaximizeFn
   ((MeasurableQuantityFn cityFoodSurplus)
    (GenericInstanceFn
     (CollectionSubsetFn FreeCiv-City
      (TheSetOf ?var1
       (and (isa ?var1 FreeCiv-City)
            (ownsCity (IndexicalFn currentPlayer)?var1)))))))))
```

Representing the subgoal relationship is now more concise:

```
(subgoal (GoalFn 14) (GoalFn 15))
```

This leads to more efficient retrieval, unification and reasoning. The savings in run time to retrieve a fact of this form versus compute it ranges from a factor of about 36 to a factor of 250.

3.6 Reifying Goal Tradeoffs

Goals are often in conflict. Resource allocation decisions in particular almost always involve tradeoffs. Most commonly, there's an opportunity cost to pursuing a goal. The appropriate balance usually changes over time and often varies across different entities based on extrinsic properties such as spatial location. In the process of building the goal lattice, we also record whether sibling goals involve a tradeoff. In Figure 2, this is indicated by an oval node shape.

The tradeoffs are partitioned along two dimensions: *partial/total* indicates whether every instance of the goal is affected or only some, while *progressive/abrupt* indicates whether the tradeoff happens gradually or instantaneously. So a total-abrupt tradeoff describes mutually exclusive goals, while total-progressive describes a global balance that could shift over time (such as changing tax allocations). A partial-abrupt tradeoff instantaneously switches the goal preference for some subset of possible entities and not others (such as suddenly putting all coastal

cities on a defensive war footing), whereas a partial-progressive tradeoff gradually changes the goal preference over time and for different entities.

The existence of a tradeoff is inferred based on whether two quantities are percentages of a common total, inversely influence each other, or constitute a *build-grow* pattern that trades off the number of entities of a quantity type versus the magnitude of that of that quantity for each entity.

3.7 Indexing Capability Roles

While the goal network amortizes subgoal relations, assignment decisions can also benefit from reifying type-level information. We treat the Freeciv game player as a kind of multi-agent system to the extent that there are different units and cities that can behave independently or in concert. The different unit types have different capabilities such that, for example, only Settlers can found cities, and only Caravans or Freight can establish trade routes. Static analysis run before the first game inspects the preconditions of primitive actions and clusters unit types into non-mutually exclusive equivalence categories with respect to groups of actions.

4. Reasoning with the Goal Lattice

To illustrate how the goal lattice can be used flexibly and efficiently, we present a sequence of

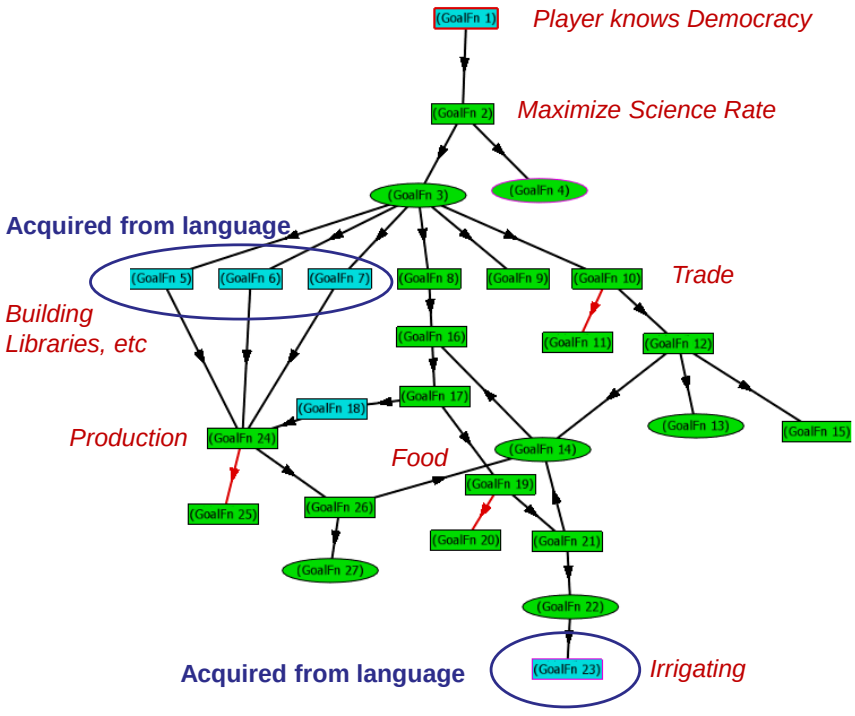

Figure 2: Reified Freeciv Goal Lattice. Oval shapes indicate goals with tradeoffs. Full goal representations would be too large to display, but see section 3.5 for the expansion of (GoalFn 14).

game-playing interpreters. We do not refer to these as planners because they do not project future states. They make individual decisions or expand learned HTN tasks to execute primitives, after which they re-plan to adapt to the dynamically changing environment.

4.1 The Legal Player

The legal player is essentially a strawman game interpreter. It works by enumerating all legal actions on each turn for each agent and picking randomly. This provides a point of comparison to show the effect of domain learning, and also serves to illustrate the size of the problem space. On the initial turn, with five units and no cities, there are 72 legal actions. This problem space quickly grows as new units and cities are constructed so that on a typical game, after 150 turns there are over 300 legal actions (388 on a recent game, but this depends on how many units and cities a civilization has). The gameplay is abysmal, as expected. In particular, it tends to continually revise decisions about what to research, what to build and what government to pursue.

4.2 The Rational Player

The rational player is the simplest player that makes use of goals when deciding what to do. It uses the goal lattice in two ways: it makes *event-triggered* decisions and it *polls* for things for agents to do. To clarify this distinction, consider that some decisions can legally be made at any time, such as choosing what to build, selecting a government or a technology to research. To avoid constantly churning through these decisions on every turn, we learn via demonstration what domain events trigger these decision types. So for example, we choose what a city should produce when it is first built or whenever it finishes building something. By factoring out the sequencing and control issues, the rational player can focus on making these individual decisions with respect to its goal lattice. It does this by enumerating the alternative choices and searching breadth-first down the goal lattice. Recall that the lattice reflects the causal influence structure of the domain model. In other words, traversing the goal lattice is a kind of regression planning from desired effects to causal influences. At each level in the lattice, it collects the sibling goals and uses simple action regression planning to gather plans that connect the decision choices to the goals. If no such connection is found, it proceeds to the next set of subgoals and tries again. Alternatively, if more than one action leads to a goal at that level, the competing plans are used to populate a dynamic programming trellis in order to select a preferred choice. Due to the dynamic nature of the domain, we use a heuristic that prefers choices that maximize future options. So for example, to achieve `TheRepublic`, a good place to start is by researching `TheAlphabet` because it enables more future technologies sooner than do the alternatives.

The other mode of operation is resource-driven. The rational planner enumerates the available actors in the game and assigns them to goals by again searching breadth-first through the goal lattice and finding the highest-level operational goal with which the actor can be tasked. Here, the notion of an operational goal is different than it is for decision tasks. Instead of applying action regression and dynamic programming, it looks for primitive actions and learned HTN plans that have been empirically determined to achieve the goal type. The most common learned plans are those that achieve the preconditions of primitives. So for example, the goal of having irrigation in a location is operational because of the existence of a plan for achieving the preconditions of the `doIrrigate` primitive. The precondition plan is learned from demonstration and by reconciling the execution trace against the declarative preconditions of the action.

Consequently, when the rational planner searches the goal lattice, it finds the goal to achieve irrigation, which it can directly perform.

In these ways, the rational player uses learned knowledge both in the decomposition of goals and in the construction of plans to achieve them. It does not, however, use known goal tradeoffs to search for appropriate balances between goals.

4.3 The Reflective Player

The reflective player is a superset of the rational player. It is designed to allow learned knowledge to override the blind search methods of the rational player. Although still under development, the reflective player will exploit higher-level strategies acquired by operationalizing domain-independent abstract strategies. Operationalization works by reconciling demonstration and instruction against the known goals and goal tradeoffs. Our belief is that effective, coordinated strategies are not something that people invent from scratch when they learn a game, but instead they are adapted from prior background knowledge. This is markedly different from a reinforcement learning approach that re-learns each game anew.

5. Related Work

To the extent that we are concerned with rational behavior in a resource-bounded agent, our approach can be viewed as a kind of Belief-Desires-Intentions (BDI) architecture. As such, it is novel in what it chooses to represent persistently in terms of goals (the type level lattice), and how it breaks behavior down into action plans and decision tasks and what or how much to re-plan on each turn.

Goel and Jones (2011) investigated meta-reasoning for self-adaptation in an agent that played Freeciv. We are pursuing similar kinds of structural adaptation, while trying to learn higher-level strategies from demonstration and instruction.

Many researchers have explored using reinforcement learning for learning to play games. In the Freeciv domain, Branavan et al. (2012) combine Monte-Carlo simulation with learning by reading the manual. While information from the manual accelerates learning, it still requires many trials to learn simple behaviors and the learned knowledge is not in a comprehensible form that can serve to explain its behavior.

Hierarchical Goal Network planning (HGN) is a formalism that incorporates goals into task decomposition planning (Shivashankar et al, 2012). It combines the task decomposition of HTN planning with a more classical planning approach that searches back from goal states. This is similar to the way we use HTN decomposition to set up regression planning from propositional goals, however we do not produce a complete plan this way and execute it. Instead, we use the planner to identify the best initial step and execute that, and then re-plan. This works well when the actions are durative, such as researching a technology. In other cases, where actual procedures make sense, we bottom out at learned HTN plans, rather than classical planning.

6. Summary and Future Work

We have described a compromise between pre-computing or providing static goals versus completely inferring them on the fly. Since the structure of the game doesn't change, the lattice

of goal types can be saved for efficient reuse without over-specifying particular entities. This goal lattice can be used in a simple reasoning loop to guide planning and decision making.

This does not preclude learning more strategic and reactive behaviors. Our current focus is on learning more coordinated and long-term strategies through language, demonstration, and experimentation. We believe that a key benefit of this weak-method for applying learned knowledge will be that as it learns to operationalize higher-level goals, it will be able to short-circuit more of the reasoning. This is a step towards our main research goal of long-term, high-performance learning.

Acknowledgements

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

References

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.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Forbus, K., Klenk, M., & Hinrichs, T. (2009). Companion Cognitive Systems: Design Goals and Lessons Learned So Far. *IEEE Intelligent Systems*, *24*, no. 4, pp. 36-46, July/August.

Goel, A. & Jones, J. (2011). Metareasoning for Self-Adaptation in Intelligent Agents. In M. T. Cox & A. Raja (Eds.), *Metareasoning: Thinking about Thinking*. Cambridge, MA: MIT Press.

Hinrichs, T. & Forbus K. (2012) Learning Qualitative Models by Demonstration. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (pp. 207-213), Toronto, CA.

Hinrichs, T. & Forbus, K. (2012). Toward Higher-Order Qualitative Representations. In *Proceedings of the 26th International Workshop on Qualitative Reasoning*, Playa Vista, CA.

Langley, P., Simon, H.A., Bradshaw, G.L. & Zytkow, J.M. (1987). Scientific Discovery: Computational Explorations of the Creative Processes. Cambridge, MA: MIT Press.

Newell, A. (1962) Some Problems of Basic Organization in Problem-Solving Programs. RAND Memo RM-3283.

Rao, A. S. & Georgeff, M. P., (1995). BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems* (pp. 312-319).

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A Hierarchical Goal-Based Formalism and Algorithm for Single-Agent Planning. In *Proceedings of the 11th International Conference on Automomous Agents and Multiagent Systems.*