Your Title Goes Here (It Can Be Really Really Really Really Long)

Your Name Here

Abstract

A nice abstract goes here.

Acknowledgments

Some acknowledgments go here.

Your Title Goes Here (It Can Be Really Really Really Really Long)

Your Name Here

A departmental senior thesis submitted to the
Department of Computer Science at Trinity University
in partial fulfillment of the requirements for graduation
with departmental honors.

April 1, 2005

Thesis Advisor

Department Chair

Associate Vice President
for
Academic Affairs

Student Copyright Declaration: the author has selected the following copyright provision:

[ ] This thesis is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License, which allows some noncommercial copying and distribution of the thesis, given proper attribution. To view a copy of this license, visit `http://creativecommons.org/licenses/` or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

[ ] This thesis is protected under the provisions of U.S. Code Title 17. Any copying of this work other than "fair use" (17 USC 107) is prohibited without the copyright holder's permission.

[ ] Other:

Distribution options for digital thesis:

[X] Open Access (full-text discoverable via search engines)

[ ] Restricted to campus viewing only (allow access only on the Trinity University campus via `digitalcommons.trinity.edu`)

Your Title Goes Here (It Can Be Really Really Really Really Long)

Your Name Here

Contents

List of Tables

List of Figures

Chapter 1

Example chapter

Example chapter, with apologies to Alex Kolliopoulos, from whose thesis the examples of math and tables were borrowed.

1.1 Examples of figures and tables

This section contains some words, plus Figure 1.1 and Table 1.1.

words, words,

words, words,

1.2 Examples of math

This section contains some math. First, here's a set of equations.

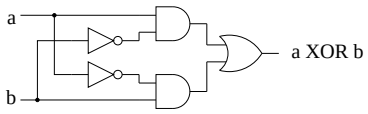

Figure 1.1: An example figure.

Trial 1	Expanding a node to select a child
Trial 3	Selecting a node near the middle of a long, linear list
Trial 4	Selecting a node near the top of a long, linear list
Trial 5	Selecting a node near the bottom of a long, linear list
Trial 6	Scrolling and expanding folders in a large tree
Trial 7	Finding a node deep and near the bottom in a large tree
Trial 8	Finding a node near the top of a large tree

Table 1.1: Purposes of each experimental trial.

$$
\begin{aligned}
y_p &= \frac{y}{\sqrt{y^2+a^2}}, \\
y_p^2 &= \frac{y^2}{y^2+a^2}, \\
y_p^2 &= \frac{y^2+a^2-a^2}{y^2+a^2}, \\
y_p^2 &= 1-\frac{a^2}{y^2+a^2}, \\
y_p^2-1 &= -\frac{a^2}{y^2+a^2}, \\
1-y_p^2 &= \frac{a^2}{y^2+a^2}.
\end{aligned}
$$

words, words,

Now here's a numbered equation.

$$0 = 0 \tag{1.1}$$

1.3 Examples of references

Section 1.1 contains Figure 1.1 and Table 1.1. Section 1.2 contains Equation (1.1). The sample bibliography file contains references to a book [**?**] and a Web site [**?**], plus some other things.

Chapter 2

Partially Observable Environments

2.1 Introduction

Until now, both the GTGR and GTGRD models have given the observer full knowledge of the adversary's state for the entirety of the game. In real-world environments, observers may not have perfect information regarding the states and actions of an adversary.

To accommodate for scenarios with incomplete information for the adversary, we introduce a partially observable variant of the GTGR scenario. In partially observable scenarios, the rules of the game remain largely unchanged, except for addition of shadow states." The observer can not discern the current state of the adversary, while the adversary occupies a shadow state. When the adversary enters an observable portion of the graph, the observer will become aware of the adversary's position once more.

Figure 2.1 illustrates a partially observable environment. Visible states, in which the observer can see the adversary white. Shadow states, in which the adversary is hidden from the observer, are black. The agent starts the game in state S. When the adversary moves to states 4 ,5, 6, or 7, the observer is unable to determine their position until the adversary

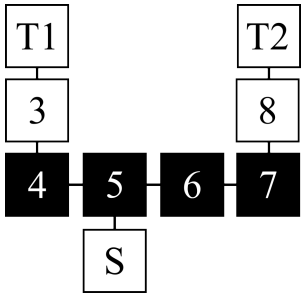

Figure 2.1: A partially observable graph.

re-enters a visible portion of the graph. We will examine two solutions to the partially observable model, both of which involving linear programming.

2.2 The Whale Method

The first method of solving partially observable environments, which we will call the "Whale Method," will utilize disjoint sets of shadow states. We call these sets of shadow states "shadow sets." We say that two shadow states belong to the same shadow set, if the adversary can travel between the two without entering an observable state.

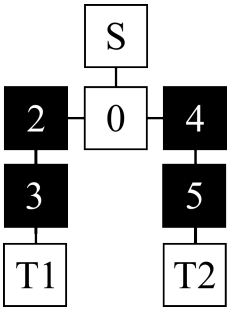

Figure 2.2: A partially observable graph with two shadow sets.

The graph in Figure 2.2 has two disjoint shadow sets, one composed of states 2 and 3,

the other composed of states 4 and 5. In using the Whale Method, the observer treats each shadow set a single state, which we will call a "whale state." We can identify the single shadow set in Figure 2.1, composed of states 4, 5, 6, and 7. In using the Whale method, the observer treats each state in the shadow set as the same state. While the adversary may require several turns to travel among states 4, 5, 6, and 7, the adversary will act as if the has decided to remain stationary in the newly created state W seen in Figure 2.3

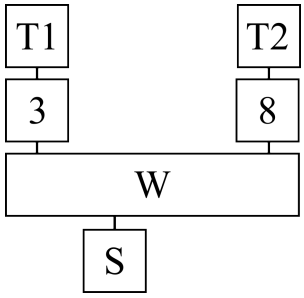

Figure 2.3: A partially observable graph with a single state in place of shadow states.

We can add the following to the mixed integer program to accommodate for partially observable environments when using the Whale method.

$$V(\theta, s) \leq \sum_{i \in B} r(s, i, j, \theta) f_i(s) + V(\theta, j) \forall \theta \in B, \forall s \mid s \neq \theta, s \notin H, \forall j \in \nu(s) \tag{3}$$

$$V(\theta, s) \leq \sum_{i \in B} r(s, i, j, \theta) f_w(s) + V(\theta, j) \forall \theta \in B, \forall s \mid s \neq \theta, s \in H, \forall j \in \nu(s) \tag{3}$$

$$\sum_i f_i(s) = 1 \quad \forall s \tag{3}$$

$$\sum_{w} f_w(s) = 1 \quad \forall s \tag{3}$$

$$f_i(s) \geq 0 \quad \forall s, i \tag{2.1}$$

$$f_w(s) \geq 0 \quad \forall s, w \tag{2.2}$$

We let H denote the set of all shadow states, and w denote the whale state the observer knows the adversary to be occupying. An observer action for any whale state w is written as $f_w(s)$. These changes to the linear program require the observer to take the same action for each turn the adversary spends in a particular shadow set. The performance of the Whale method will be examined in a later section.

2.3 The Transmogrification Method

When using the Whale method, the observer ignores some of the information available to them. The Whale method does not account for where the adversary entered a shadow set, or how long the adversary has remained hidden. The "Transmogrification Method" takes both of these pieces of information into account, by generating a fully observable environment from a partially observable environment. To do this, the observer must make some basic assumptions about the adversary's strategy. The following lemmas and corollary assume the agent is playing optimally against the observer's stationary strategy.

Lemma 2. *If the adversary's target does not lie within a shadow set, the adversary will eventually exit the shadow set.*

Proof (Sketch). As mentioned previously, the game could theoretically go on forever.

But because of the potential per-timestep cost of d and the observer's predictions, any sufficiently long path for the adversary would be dominated by the strategy of taking the shortest path to their target θ. If the adversary never leaves the set of shadow states, then the game will go on forever. Thus, the adversary will eventually leave a shadow set.

Corollary 2. *An optimal agent occupying a state in a shadow set will take a shortest path to the exit state of their choosing.*

Proof (Sketch). We established that an optimal adversary in a shadow set must exit that shadow set. Each unnecessary turn the adversary spends in a shadow state invites the observer to guess their intended target. Thus, it is in the observer's interest to reach their chosen exit as quickly as possible.

With Corollary 2, the observer can generate a new graph to play on.

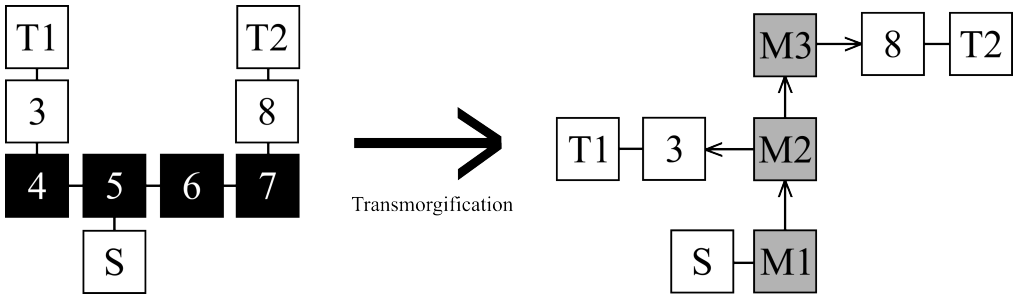

Figure 2.4: The partially observable environment from Figure 2.1 (left) transmogrified into an, fully observable graph.

Figure 2.4 illustrates a transmogrified version of the graph from Figure 2.1. The grey nodes in the transmogrified graph are introduced to represent the number of turns the adversary has remained hidden from the observer. For instance, if the adversary has been hidden for two turns, then the observer would see the adversary as occupying state $M2$ in the transmogrified graph. Take note that the transmogrified graph now includes directed

edges, because we are (hopefully) not dealing with a time traveling agent. To generate these graphs, we examine each shadow set. For each shadow set, we examine each "entrance state" connected by an edge to the shadow state. We then compute the shortest path between every two entrance states for that shadow state. By Corollary 2, we assume the length of the longest shortest path will be the maximum number of turns an optimal agent will spend within the shadow set. We then create create the same number of memory nodes with directed edges from one to the next (see states $M1, M2, M3$ from state S in Figure 2.4). Then we create an edge from each entrance state, to the memory state corresponding to the length of the shortest path between the entrance state, and the original entrance state from which we spawned the memory states. For example, because a adversary would have to two shadow states on their journey from state S to state 3 in Figure 2.1, we connect state $M2$ to state 3 in Figure 2.4. Note, that the post transmogrification graph in Figure 2.4 is actually incomplete, because memory states have only been spawned from the entrance state S and not from entrance states 3 or 8. Because an optimal agent has no reason to backtrack in this particular scenario, the memory nodes for 3 and 8 were omitted to keep the graph simple.

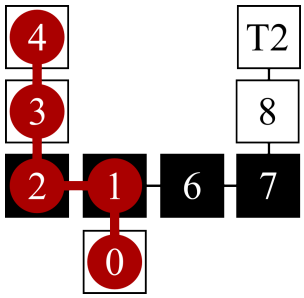

Figure 2.5: The adversary's path to target $T1$.

Next, let us examine how a game might play out from the adversary's perspective.

Using the environment from Figure 2.1, let us assume the adversary was assigned target $T1$. Because the graph is rather limiting, the adversary takes the shortest path from starting state S to the target $T1$. Figure 2.5 illustrates the adversary's journey, each turn market with a red circle. Note that on turns 1 and 2, the observer would lose sight of the attacker until turn 3. Now, let us examine what the same scenario would look like from the observer's prospective, when using the transmogrification method.

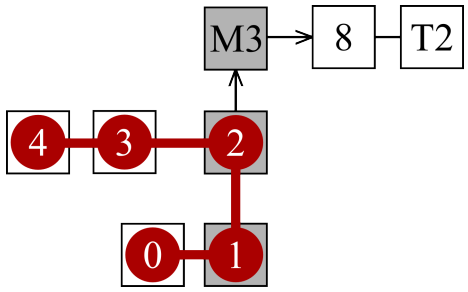

Figure 2.6: The adversary's path to $T1$ from the observer's perspective.

Because the adversary takes two turns within the shadow set, the observer sees the adversary as moving from state $M1$ to state $M2$, before exiting the shadow set on turn 3.

2.4 Experiments

As with previous experiments, all tests were run on a machine using OSX Yosemite version 10.10.5, with 16 GB of ram and a 2.3 GHz Intel Core i7 processor. First, we will use the simple example from Figure 2.1 to compare the performance of the Whale, and Transmogrification methods. Additionally, we will compare the performance against a best case solution, in which we turn all shadow states into standard states. The best case solutions measures how the observer would perform if their strategy effectively negated the shadow states. Graphs were displayed using the Python NetworkX library. Arrowheads on directed

edges were added manually for clarity. The starting state is displayed in green, target states are displayed in red, shadow states are displayed in purple, and default states are displayed in cyan.

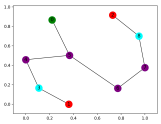

Figure 2.7: A simple scenario using NetworkX.

After transmogrifying the graph, the scenario becomes more complex, but allows the observer to play a fully observable game. There graph will no longer have any shadow states (purple). All states labeled with values greater than 1000 are nodes that have been added to represent turns hidden within the shadow set. Note that the graph in Figure 2.8 will have more nodes than the previously seen illustration of the transmogrified graph in Figure 2.4, because we spawn memory states for every entry state, and not just the starting state.

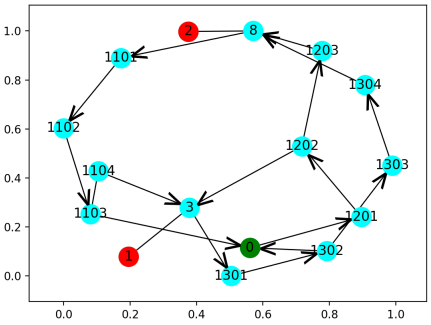

Figure 2.8: The transmogrified graph.

For this game, the adversary is not penalized for timesteps taken. The reward for

reaching the target is 0. The reward the observer receives for each correct guess was set to 1. The point value for each method at the end of the game equates to the number of correct guesses the observer can expect to make. At the start of the game, the adversary has a 75% chance of being assigned target 1, and a 25% of being assigned target 2.

No Shadow	3.75 Correct Guesses
Whale	3.25 Correct Guesses
Transmogrification	3.5 Correct Guesses

Table 2.1: Simple Performance Test Results.

Appendix A

Example appendix

Here is my code.

```
#include <iostream>
int main(void) {
    cout << "Hello, world!\n";
    return 0;
}
```