NANYANG TECHNOLOGICAL UNIVERSITY

CZ4046 Intelligent Agents
AY 2012-13 Sem 2
Final Assignment

A Hybrid Architecture for a Multiagent System in TileWorld, Optimised using Genetic Algorithms
Technical Report

Author:

Prerna Chikersal

Teammates:

Shailesh Ahuja, Niklas Forsmark, Bhargav Sosale and Chetna Goyal

We are team "Agents Undercover"

Lecturers:

Prof. Bo An and Prof. Michael Harold Lees

1 Introduction To Tileworld

TileWorld[1] is a widely used testbed for evaluating agent architectures. It contains grid cells and every grid cell can either be empty or can contain an agent, a tile, a hole or an obstacle. Agents can move up, down, left or right. The goal of the agent is to collect and move tiles to fill the holes. The TileWorld environment is dynamic, that is, tiles, obstacles and holes appear and disappear randomly. It is also highly parameterised, that is, the object creation rate, object mean, lifetime of an object and few other parameters governing the dynamism (rate at which new holes appear) and hostility (rate at which obstacles appear) are in the form of parameters and can be controlled easily by tweaking the parameters. Upon receiving criticism regarding TileWorld being too simple, [2] added a fuel level parameter and a refuelling station. In this project, agents are given 1000 units of fuel, such that, the agent must return to refuel at the refuelling station, before he has moved 1000 times. Also, when an agent fills a hole with a tile, its score increases by 1. The total score of a run is the cumulative number of holes filled by both agents.

2 Agent Architectures Developed and Tested

Initially, a reactive agent was implemented to understand how the MASON agent toolkit works. After that, a simple practical reasoning agent was implemented. Reactive agents tend to perform better in environments with larger number of resources, while practical reasoning agents work better in environments with low dynamism and lesser number of resources. Since we want our agent to perform well in all three environment configurations provided to us, we add reactive capabilities to our practical reasoning agent, thus creating a hybrid agent architecture.

2.1 Reactive Agent Architecture

Purely reactive agents are agents which act without referring to their history. They have no internal memory and base their actions purely on the present state of the environment.
That is, action: E $\rightarrow$ Ac, where E is the set of discrete instantaneous states of the environment and Ac is the set of possible actions available to an agent.

Reactive agents perform well in an environment with a large number of resources. The dynamism of an environment is not very relevant to a reactive agent, since it never really remembers when an object was last seen. In sparse environments with very less number of resources, reactive agents tend to not perform well, since they do not deliberate or plan how to get to a resource they had seen some time back; they rely completely on chance and keep moving randomly till a resource appears and they are able to grab it.

The purely reactive agent we implemented in Tileworld, would move in any random direction, till it came across a tile it can grab. After grabbing the tile, it would once again resort to moving aimlessly, until it sees a hole to put the tile in or another tile to collect. Whenever the fuel units of the agent fell below a certain threshold, it would move towards the refuelling station in order to refuel.

2.2 Reactive and Practical Reasoning Hybrid Agent Architecture

Practical reasoning agents are based on (i) deliberation, which involves, deciding what states of affairs it wants to achieve, and (ii) means-end reasoning reasoning, which involves, deciding how to achieve those states of affairs.

Based on percepts from the environment, beliefs are generated. In Tileworld, beliefs can be location of tiles, holes, obstacles, other agents, etc. Based on the intentions and current beliefs, the agent will derive desires like wanting to pick up a tile, wanting to refuel, wanting to drop tile, etc. The beliefs, desires and intentions are then filtered to give new intentions. This is our major decision making stage, which is completed with the help of utility functions.(*Due to limited space, I am not describing the utility functions in my report.*) Then, the beliefs and new intentions are used to generate a plan. A plan is the set of actions the agent needs to perform to fulfil its intention. Finally the plan is executed and the agent will act according to the plan. However, there are a number of issues in this method, such as:

- Beliefs change constantly, even while we are executing a plan. Hence, after executing each action in a plan, it is necessary to fetch a new percept, derive new beliefs from it and check if the plan is sound, given the new beliefs and previous intentions. If it is found that the plan is no longer sound, the execution of the plan must stop and a new plan must be generated. **In Tileworld, the only situation where a plan becomes unsound is when our intentions change, in which case, we'll need to re-plan**.
- Even with the above, the agent still remains over-committed to its intentions. Hence, the agent must stop to determine whether its intention has succeeded or is impossible. **In Tileworld, the only situation where an intention becomes impossible is if obstacles block the path to our intention. If the agent is completely surrounded by obstacles, it will stay there and wait**.
- Also, it is possible for intentions to change. For example, lets consider an agent exploring a sparse environment. The agent will continue to explore unless it finds a tile. But, what if the agent runs out of fuel in between? Then, there must be a way to get the agent to reconsider and change its intentions. **In Tileworld, we reconsider our intentions and beliefs at every step, since it is a constantly changing environment**. However, reconsidering intentions is an expensive process as it requires fetching new beliefs, deriving desires and filtering those desires using utility functions, which can be expensive to compute. Hence, it is important to decide when to call reconsider() and when to not call reconsider(). **For this purpose we change our architecture to a hybrid architecture by adding high priority reactionary procedures**.

3 Communicating Agents and Novel Exploration Strategy

"There is no such thing as a single agent system". All, but the most trivial computer systems contain many sub-systems which work together. We implement communication between two agents with the hope that by working together, their cumulative score will increase. Although, we are only allowed to communicate 3 things at a time, we have identified a total of 5 things which when communicated can boost our score. Depending on the situation, we dynamically decide the 3 things to send, out of the following:

- **Request and Response for tiles and holes**: Agents can ask other for the location of the nearest tile to them. This is done in every time step.
- **Intentions**: Agents can communicate their intentions so as to avoid intention clashes. When an agent's intentions change, the new intentions are given a very high priority and almost always communicated. When Agent 0 sends Agent 1 an intention, Agent 1 always

replies with an obstacle in Agent 0's path. And if Agent 0's intentions clash with Agent 1's, Agent 1 changes its intention.

- **Updating out-dated memory**: If Agent 0 tells Agent 1 the location of some object, and Agent 1 approaches that object, only to find that that object has disappeared, Agent 1 will send a message to Agent 0, asking him to update his outdated memory.
- **Current Location**: The agents tell each other their current locations after every 5 time steps.
- **Update Nearest Obstacle**: Upon receiving an intention, the agent always responds with an obstacle in the path of the other agent's intention. This is done by creating a box around the other agent, using the its current location (sent every 5 time steps) and target location (location of intention), picking 10 random points in the box, finding obstacles closest to those 10 points and finally, returning the location of the obstacle closest to the other agent.

A novel **Exploration Strategy** is also implemented, which involves the agent keeping a track of its previous locations in the environment, called snap points. Since the location is communicated to the other agent, once in every 5 steps, both the agents can keep track of each other's snap points, so as to avoid exploring the same area.

4 Parameter Optimisation Using Genetic Algorithms

Utility functions (mentioned in teammates' reports) help filter out intentions from desires and are important for decision making. To calculate utilities, they use certain constants or thresholds, which can be changed to give better results for different environments. Usually, these parameters are hard to predict, hard to find and are often determined by hit and trial. The search space for sets of these parameters is very large. Therefore, we need to use a probabilistic optimisation algorithm to find optimal or good enough sets of parameters for agents in different environments. Genetic algorithm is one such algorithm which can help us do this.

4.1 What are Genetic Algorithms?

Genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic is routinely used to generate useful solutions to optimisation and search problems. The outline of a genetic algorithm is as follows:

1. **"Genesis" or initialisation**: Create an initial set (population) of n candidate solutions, which can be random or can be seeded by some hand-picked solutions.
2. **Evaluation**: Evaluate each member of the population using some fitness function.
3. **Survival of the Fittest**: Select a number of evaluated candidates with high fitness score to be the parents of the next generation.
4. **Evolution**: Generate a new population of offsprings by randomly altering and/or combining elements of the parent candidates. The evolution is performed by one or more evolutionary operators. The most common operators are cross-over (combining) and mutation (altering).
5. **Iteration**: Repeat steps 2-4 until a satisfactory solution is found or some other termination condition like max generation count, is met.

4.2 Implementation of Genetic Algorithms: An Overview

We use the Watchmaker Framework For Evolutionary Computation[3] to implement Genetic Algorithms.

There are 2 prerequisites to implement genetic algorithms:

- **Encode the parameters to be optimised into a separate data structure**: In our project, we take all the constants and threshold used in the utility functions and put them in a data structure called "Genome".
- **Have a fitness function which can completely or partially evaluate the effectiveness of solutions generated** : In our project, the cumulative scores achieved by 2 agents in a multiagent system, during 1 run, becomes the fitness function.

4.3 Defining Rules for Evolution

To extend GA to Tileworld, the following is defined:

- **"Genome" of an Agent**: Data Structure containing parameters used by utility functions of the hybrid agents.
- **Candidate in the Population**: Two homogenous agents (with same parameters) who communicate with each other.
- **Initialisation and Population Size**: 1 Generation = Population size of 10 candidates (10 pairs of agents), 5 of whom are initialised to default parameters chosen by hit and trial. While, the other 5 are initialised to parameters chosen randomly.
- **Number of Runs/ Termination Condition**: Run for 20 generations and evolved using GA. Termination Condition used is generation count, which is equal to 20.
- **Fitness Function**: We will train all candidates for the same environment. Hence, the fitness function of a candidate is equal to the number of holes filled or score of the run.
- **Selection of Parents for Reproduction**: A Selection Strategy called ?Stochastic Universal Sampling? is used. This ensures that even in a population where some candidates are much fitter than the rest, the less fitter candidates get a chance to reproduce. In a small population, this also prevents the algorithm from converging too fast. Lets say we place all candidates on a roulette wheel, giving them space proportional to their fitness, and a candidate x occupies 4.5% of the wheel, then Stochastic Universal Sampling ensures that candidate x is not allowed to reproduce more than 4 to 5 times.
- **Elitism**: Elitism specifies the % of the fittest parents which are moved unchanged into the next generation. For our training, we have set elitism to be 0, because elitism causes the algorithm to converge to a local optima too quickly. This is mainly because our population size is very small, due to insufficient computational power.
- **Genetic Operators**:
  - *Mutation* – Randomly changing some parameters in an agent to ensure diversity and exploration of a greater search space.
  - '*Crossover* (reproduction) – Randomly picking some number of parameters (6 in our training) in 2 agents and exchanging them.

5 Results and Analysis

All results below are shown with a sensor range of 2 and random seed of 9042014. Environments 1, 2 and 3 are configured according to the assignment manual.

5.1 Comparing Reactive, Practical Reasoning and Hybrid Agent Architecture

Figure 1 and 2 shows the comparison between Reactive, Practical Reasoning and Hybrid Architectures in Environments 1, 2 and 3 (as specified in the assignment manual), respectively.

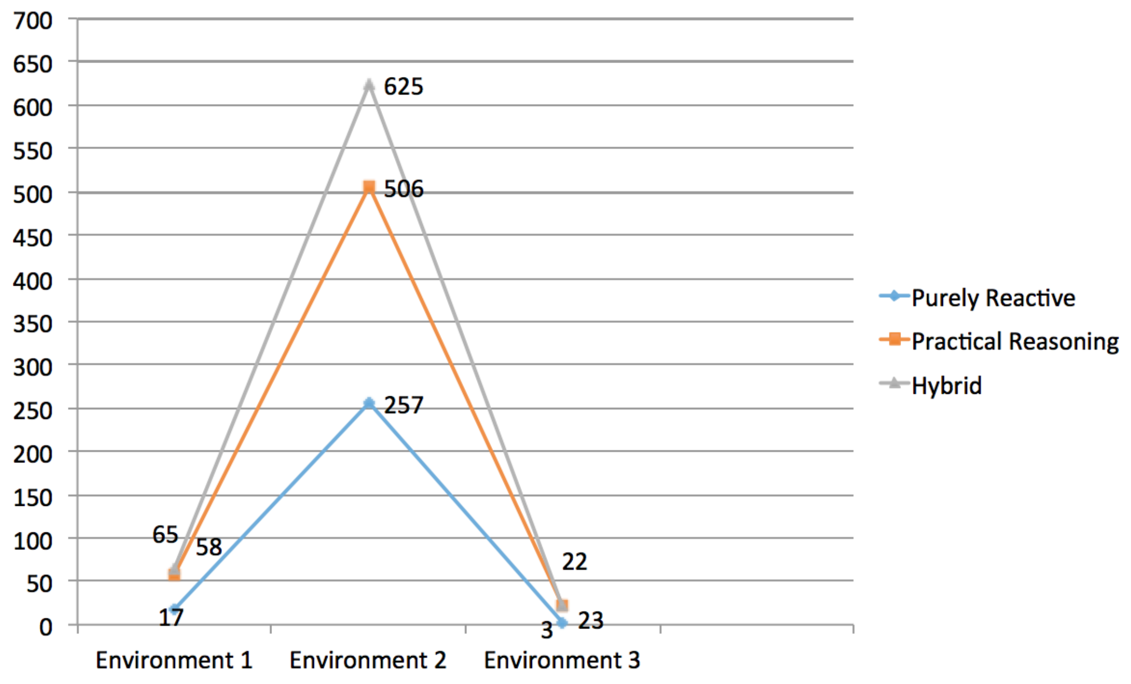
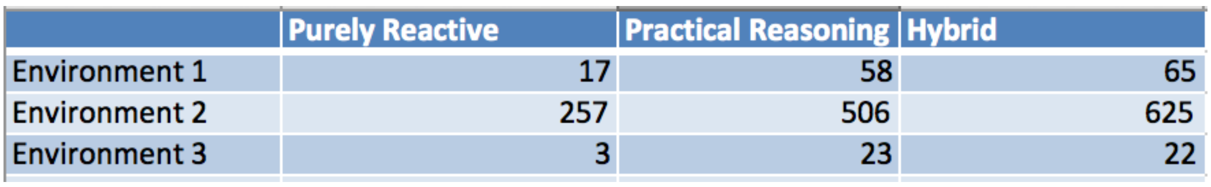


Figure 1: Comparing Reactive, Practical Reasoning and Hybrid Agent Architecture



	Purely Reactive	Practical Reasoning	Hybrid
Environment 1	17	58	65
Environment 2	257	506	625
Environment 3	3	23	22

Figure 2: Comparing Reactive, Practical Reasoning and Hybrid Agent Architecture

From the above, we can see that Hybrid performs best in all 3 environments. Reactive performs worst in all 3. While, Hybrid and Practical Reasoning perform equally in 1 and 3.

5.2 Visualising Improvement in Performance due to Genetic Algorithm Parameter Optimisation

Figure 3 and 4 shows the how the overall performance of the system improved significantly by using GA for parameter optimisation in Environments 1, 2 and 3 (as specified in the assignment manual), respectively.

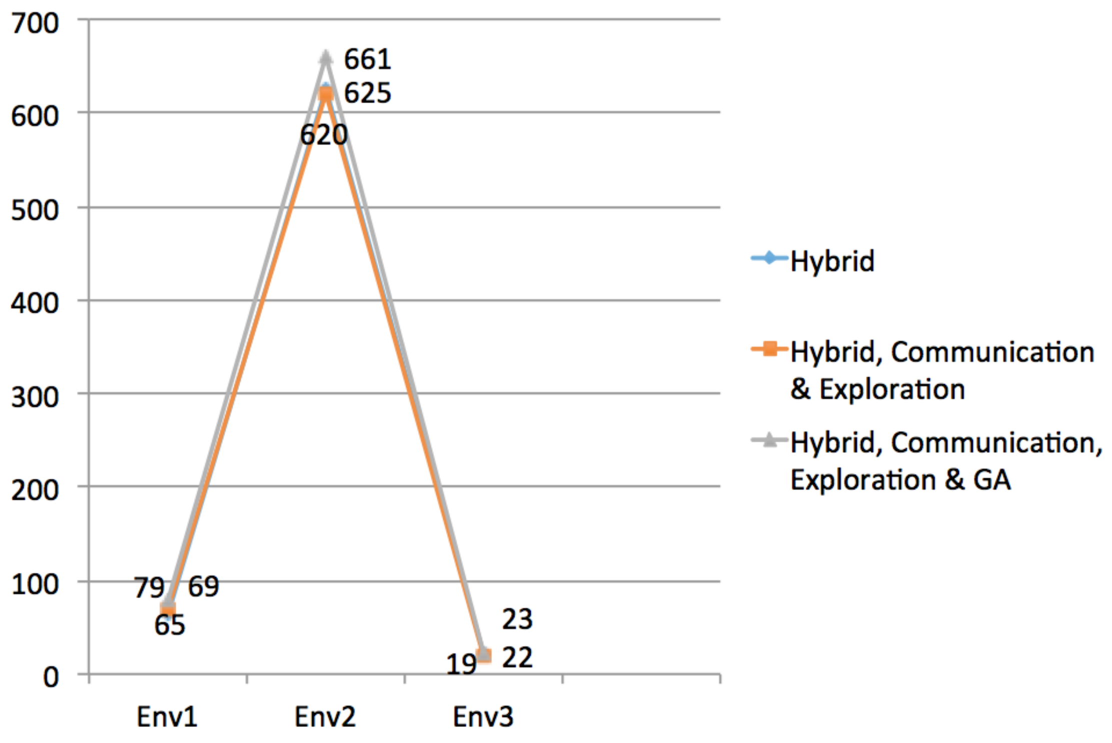
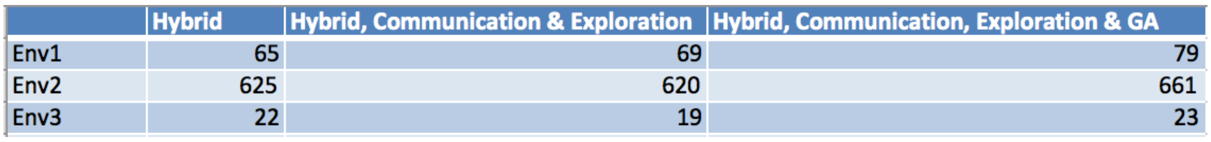


Figure 3:



	Hybrid	Hybrid, Communication & Exploration	Hybrid, Communication, Exploration & GA
Env1	65	69	79
Env2	625	620	661
Env3	22	19	23

Figure 4:

6 Conclusion

From the above graphs, we can see that out of the architectures implemented, Hybrid Agent Architecture performs the best in Tileworld. Also, additional features like Communication and Exploration Strategy help to increase the score till a certain extent. Finally, Genetic Algorithms, depending on the space they search, are almost always able to improve the performance of the multiagent system. GA combined with the Hybrid architecture, Communication and Exploration Strategy, seems to give the best possible score.

7 References

[1] http://www.cs.nott.ac.uk/WP/2002/2002-1.pdf

[2] Pollack, Martha E., et al. "Experimental investigation of an agent commitment strategy." Pittsburgh, PA 15260 (1994).

[3] http://watchmaker.uncommons.org/