

Multiagent Resource Gathering in RTS

Sergio S. Sette Geber L. Ramalho

Universidade Federal de Pernambuco, Centro de Informática



Figure 1: A resource gathering map. Screenshot taken from a RTSCup simulation map. Resource (yellow), Worker (blue), Obstacle (grey)

Abstract

Most of the RTS (Real-Time Strategy) games have the same basic formula. You gather resources, spend them building an army, creating structures and researching new technologies, and finally, send the army to conquer the adversary. Each of these stages is really complex and could be a topic of research. This paper tries to focus at the resource gathering stage and propose a cooperative multiagent solution.

Keywords: real time strategy, multiagent, cooperative, path-finding

Author's contact:

{Sergio Sette} sergiosette@gmail.com
{Geber Ramalho} glr@cin.ufpe.br

1. Introduction

Consider a worker's job in a RTS game. When the game begins, it has an apparently simple task, find a suitable resource source, gather as much as he can carry, and return to a resource delivering spot. It seems really simple, but each of these stages can be a very complex task, since we want to get the most resources in the shortest amount of time.

The first task a worker have, is to choose where he wants to go. Ideally, he wants to shorten the distance he has to cross, so he needs to find the closer resource deposit, meaning it also needs to know how to measure the distance to the resource deposit. The Euclidean Distance and the Manhattan Distance (City Blocks Distance) are the most commonly used, but they do not consider the actual path the worker is taking. As we can see at Figure 2, the resource 1 has the shorter Euclidean Distance from the worker, but it takes longer to reach than resource 2, due to the resource 1 being surrounded by impassable obstacles.

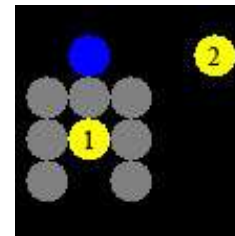


Figure 2: Resource 1 (yellow), Resource 2 (yellow), Worker (blue), Obstacle (grey)

The second task is to find the best path from the worker current location, and his target, being a resource spot when he can still carry more resources, or a resource deposit location, when he's full of resources. Usually, this is done applying the A* algorithm, or one of its derivatives which assumes that the world is static. In some cases it may be possible to use these assumptions in a world with multiple agents. If the world is sufficiently large, the costs of making this assumption will not be large. When the world is sufficiently constrained, cooperative path-finding is needed. [JANSEN AND STURTEVANT 2008]

The third task is doing the two first tasks while coordinating with the other workers in the environment, trying to maximize the overall team performance. For example: assigning a worker to a suboptimal resource, avoiding bottleneck regions.

2. Related Work

2.1 Optimal Foraging Theory

The Optimal Foraging Theory [MACARTHUR AND PIANKA 1966] was created to explain foraging animals, like ants, behavior. This theory states that the animals try to maximize the formula $E/(h+s)$ where "E" is the amount of energy that a food provides, "h" is the time

to capture the prey, or the food, and “s” is the time needed to search for the food.

This theory can be applied in multiagent resource gathering, by making sure the workers search the nearest resources, thus minimizing the “h”, carrying the maximum amount of resources possible, maximizing the “E”, and deciding when to explore, minimizing the “s”.

2.2 jcmjWorker

Studying foraging techniques, Moura [2006] created a gathering strategy based in the Optimal Foraging Theory [MACARTHUR AND PIANKA 1966] entitled jcmjWorker. In this strategy, the workers searched for the nearest resources. He also considered the amount of time a possible gathering worker was gathering the resource at the time. In his work, he applied the basic A* algorithm for path-finding, and in case of collision, the workers waited five turns. If the collision were to persist, new paths would be calculated.

The jcmjWorker competed in the Autonomous Agents class competition. His worker did well when the resources spots were near the resource delivering spot, but it did badly when there were complex obstacles like corridors, colliding a lot.

2.3 CA*

As we have seen, for complex ambient we need a cooperative path-finding solution. Silver [2005] explained how using time as a new dimension, can avoid a lot of collisions, since the workers will be sharing their plans, making everyone knowing everyone else’s position at any time. It works as a simple reservation table with three dimensions: x, y, and t, representing the time. Only a small proportion of grid locations will be touched, and so the grid can be efficiently implemented as a hash table, hashing on a randomly distributed function of the (x, y, t) key [Silver 2005].

3. Solution

This chapter will explain how a simple learning mechanism together with a cooperative path-finding algorithm can produce a nice and simple strategy with good results.

3.1 Choosing the destination

In chapter one, we have concluded that the Euclidean and Manhattan Distances are not good enough to know which of the resource will take the shortest time to reach. We need to know which one has the lowest cost of movement.

We define the cost of movement as the time required for a worker to reach a resource and deliver it

to the delivering spot. The cost of movement of each resource is initially the Manhattan Distance, and is updated each time a worker reaches it, with a learning rate α used to determine the speed of the learning process.

Taking this definition of cost of movement, we can not only have a better estimation of the shortest resource, but it also takes in account the worker’s traffic. If the path to a resource is complex like a narrow corridor and most of our workers go for the same resource, they will take longer to reach it, increasing the resource’s cost of movement, making this resource less desirable.

Another problem to consider is the update rate of a resource in a cluster of resources. When a worker find the resource with the lowest initial cost of movement, but later find out that it took too long to reach it, only this resource will get updated. As result, the worker will probably end up gathering from a nearby resource from the same cluster, until all of that cluster’s resources get updated one by one. To solve this, our resource cost of movement updates are propagated to all the resources in the same cluster, updating all of them.

One problem still remains: it could take several trips to update a bad cluster, and several bad clusters until all the workers get to the best clusters. To fix this, we assign one of the workers to be a scout worker. The scout worker is a simple worker, but it chooses the best cost of movement resource that was not already updated. When all of the clusters are updated, the scout worker becomes a normal worker. Also, the scout has a higher learning rate α during its scouting stage.

3.2 Path-finding

Using CA* (Cooperative A*), our workers will not be colliding against each other, solving our worker’s second task dilemma.

It’s costly to keep calculating a path, and most of the time, the workers are moving to a resource a worker has already been. To solve this, we can store our previously calculated paths in a hash table, using the resource delivering and resource deposit locations as the key. If we have only one deposit delivering spot, we can use only the resource deposit location as the key.

To improve the overall system robustness, we also keep track of how many turns a worker takes to find its path. If it’s more than one turn, it needs to start planning ahead of time, since it couldn’t keep up with the plan, which is now an invalid plan. This often happens when dealing with a huge ambient, with many agents.

4. Experimental Results

To evaluate the performance of the algorithm, the RTSCup [RTSCup] simulator was used. It's written in Java and uses a socket protocol for the client-server communication, allowing a client written in any language to work.

The tests were done with the bundled java client, and this paper's solution's worker, called CoopWorker was compared to a SimpleWorker, a simple non-cooperative worker based in jcmjWorker choosing the lowest Manhattan Distance resources, but with the improvement of using CA* as path-finding.

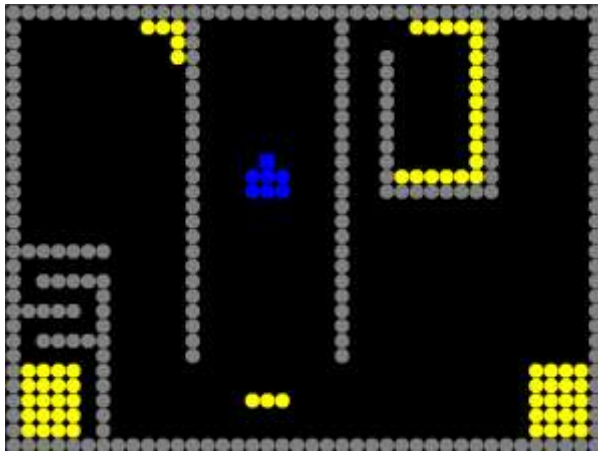


Figure 3: Simulation map. Blue circles are the workers, yellow circles the resources, grey circles the obstacles and the blue square is the delivering spot.

At the simulation, the workers were given 3000 turns to gather as much resources as possible. The table shows the amount of resources gathered for each implementation of the agents. This map was created with some of the most common problems in this area, like a narrow maze corridor, and a cave-like area with a small entrance.

Table 1: Experiment results

Worker	Resources gathered
CoopWorker - no scouting worker	1800
CoopWorker - one scouting worker	1820
CoopWorker - no cluster propagation	1610
SimpleWorker	1487

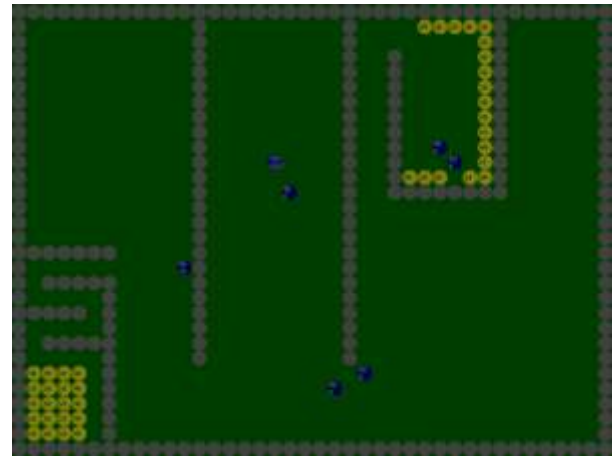


Figure 4: CoopWorker end of simulation screen.

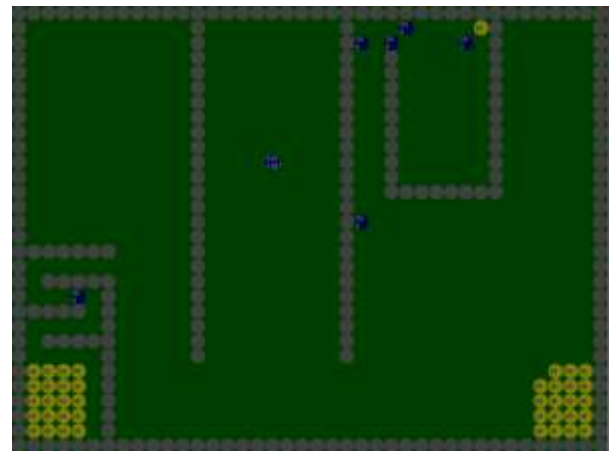


Figure 5: SimpleWorker end of simulation screen.

As we can see at Figure 4 and Figure 5, the CoopWorker gathers all the bottom right area, which is an open area free of possible collisions, while the SimpleWorker prefer the cave-like area, since its resources have the best Manhattan Distance.

5. Conclusion and Future Work

This paper shows how static monoagent techniques are not ideal for dynamic cooperative multiagents scenarios. We need a cooperative approach for a cooperative problem. The cooperative path-finding CA* has its flaws but it was enough to our needs. Some experiments with the WCHA* [Silver 2005] may prove it is superior in this problem, improving not only the agents capability, but also their computational cost.

Some of the concepts can be improved. For instance, choosing the number of scout works to be used, making a scout/worker ratio, since in big maps, more scouting is needed. A progressive learning rate could improve the learning process, making α high at the first iterations, and lower it gradually to make it less noise susceptible.

There's also the need to make some standard metrics like the number of collisions and computational cost.

References

- [JANSEN AND STURTEVANT 2008] JANSEN, R., AND STURTEVANT, N. 2008B. A NEW APPROACH TO COOPERATIVE PATHFINDING. IN AAMAS, 1401–1404. RICHLAND, SC: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS.
- [SILVER 2005] SILVER, D. 2005. COOPERATIVE PATHFINDING. IN YOUNG, R. M., AND LAIRD, J. E., EDS., AIIDE, 117–122. AAAI PRESS.
- [MACARTHUR AND PIANKA 1966] MACARTHUR, R. H. AND PIANKA, E. R. (1966). ON THE OPTIMAL USE OF A PATCHY ENVIRONMENT. AMERICAN NATURALIST, 100
- [MOURA 2006] MOURA, C. J. UMA ESTRATEGIA EFICIENTE DE COLETA MULTIAGENTE PARA JOGOS RTS. UNIVERSIDADE FEDERAL DE PERNAMBUCO COURSE COMPLETION ASSIGNMENT.
- [RTSCUP] [HTTP://WWW.RTSCUP.ORG](http://www.rtscup.org)