The Multi-Agent Programming Contest

Miriam Kölle

University of Koblenz-Landau

1 Introduction

The Multi-Agent Programming Contest[1] (MAPC) is an international online competition. Started in 2005 it takes place every year. The MAPC is meant to incite research in the area of multi-agent programming as stated in [BDH+11] by

- identifying key problems,
- collecting suitable benchmarks, and
- gathering test cases which require and enforce coordinated action

The tournament is an online event, that is, all participants run their systems locally and communicate with the contest server via the Internet. The winning team is determined by each team playing against all other teams. All participants are free to chose an implementation and agent communication technology by themselves. Usually there are about 5 to 10 participants as seen on the website.

The MAPC scenario is changing over the years. The current one is the 'Agents on Mars' scenario. However, it is unclear whether there will be a contest in 2014 at all, as there is a new scenario in the making and the organizers didn't decide yet whether to use this one, stick to the old one, or even skip the 2014 contest (April 2014). The previous scenarios were the following

- 2005: Food Gatherer
- 2006-07: Goldminers
- 2008-10: Cows and Cowboys
- since 2011: Agents on Mars

The focus of the MAPC changed with the different scenarios, which got more and more complex. The current one, Agents on Mars, demands for sophisticated agent communication and coordination in order to stand one's ground.

2 Agents on Mars

The Agents on Mars scenario is about occupying zones on the Mars. A detailed description can be found in [MAPC]. Abbreviated, mankind is to populate the

[1] https://multiagentcontest.org

planet and therefore water wells have to be found. The goal is to occupy the best zones and defend them against the competing team. Each team thereby consists of several agents which take on different roles, i.e., the team is consisting of heterogeneous agents.

2.1 Environment

The environment is given by a weighted graph. Each vertex has an unique identifier and is assigned with a value. Each edge is assigned with a weight, defining the costs of traversing this edge. The environment is unknown in the beginning thus the agents must explore it.
A team can occupy zones, i.e., subgraphs, and each zone value counts into the team's score. The vertices are exceptional in that only probed vertices count their full value. If not probed (this is an agent action, please see the next section) the vertices only count 'one'.

2.2 Occupying Zones

To define the zones a team occupies, a specific algorithm (graph coloring algorithm) is used. In general, zone building depends on all agents' current position at the current simulation step (i.e., it 'might' change). The algorithm works as follows:

- A vertex belongs to that team with the majority of agents standing on that vertex
- Neighbors dominated by at least two neighbors (neighbors occupied by one team) also belong to that team
- All vertices in an isolated zone belong to the team having established the border

It is obvious that one agent alone cannot establish a zone.
The graphics 1-4 are extracted from [**?**] to visualize the just described zone building algorithm. In (a) one can see the coloring of each vertex where an agent is currently standing on. In (b) the second step includes the coloring of all vertices with at least two colored neighbors. In © the third step is shown: the green team was successful in establishing a green zone. (D) finally shows the effect of a red team's agent moving one position in the next simulation step – destroying the green team's zone.

2.3 Agents

Each team consists of heterogeneous agents, differing in their internal attributes and their actions they are able to perform. There do exist five different agent roles: Explorer, Repairer, Saboteur, Sentinel, and Inspector. One could also call them 'experts' for different situations. In 2013, a team consisted of 28 agents – 6 Explorers, 6 Repairers, 4 Saboteurs, 6 Sentinels and 6 Inspectors.

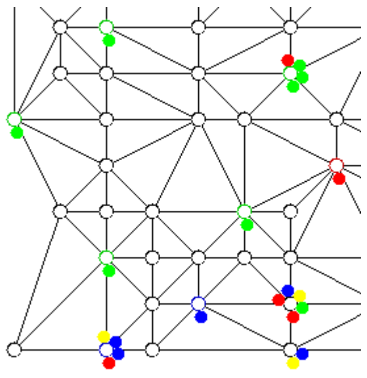

Fig. 1. (a)

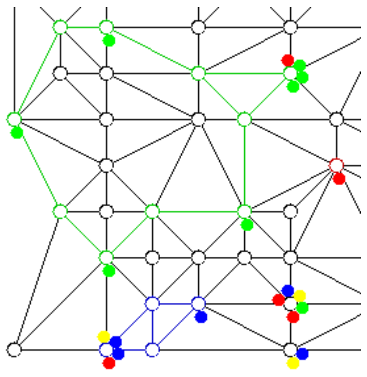

Fig. 2. (b)

The agent's internal attributes are energy, health, strength and visibility range. Energy is needed for almost all actions, i.e., actions cost energy and agents can only execute a specific action if its energy level is sufficient. The health of each agent can be decreased, resulting in less actions the agent can perform. If the health goes down to zero, the agent is called to be disabled. The strength defines the strengths of an attack and is of interest for the Saboteur. The visibility range defines the range an agent can see, i.e., the distance in which he can perceive vertices and other agents.
Internal attributes can vary during the simulations, that is, actions cost energy and attacks cost health. In addition, an agent can 'change' its attributes' values using the 'buy' function that will be described just in the following.
There exist 10 different actions in total:

- skip: do nothing
- recharge: increases the energy by 50
- goto: move from one vertex to the other one (edge weight ? energy loss)

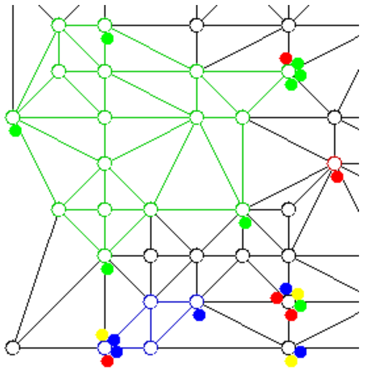

Fig. 3. (c)

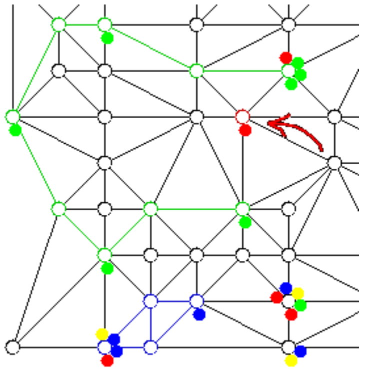

Fig. 4. (d)

- probe: get the full value of a vertex
- survey: get the weights of all edges in the visibility range
- inspect: inspect internal attributes of another agent
- attack: attack another agent
- parry: parry an attack
- repair: repair an agent – not itself
- buy: increase the agent's internal attributes

Some of these actions are ranged actions, i.e., agents do not have to be at the same vertex. However, the specific vertex has to be in the agent's visibility range and the energy costs increase while the effectiveness of an action decreases. Ranged actions are probe, inspect, attack and repair.
Only some of these actions are executable if an agent is disabled, more precisely these actions are: skip, recharge, goto, repair.
Actions can fail. Possible causes: not enough energy, being successfully attacked, an attack was parried by the opponent, out of visibility range, wrong role, . . . and any action can fail randomly with a probability of 0.01.

Role	Actions	Energy	Health	Strength	Visibility Range
Explorer	skip, goto, probe, survey, buy, recharge	12	4	0	2
Sabteur	skip, goto, parry, survey, buy, attack, recharge	7	3	4	1
Repairer	skip, goto, parry, survey, buy, repair, recharge	8	6	0	1
Sentinel	skip, goto, parry, survey, buy, recharge	10	1	0	3
Inspector	skip, goto, inspect, survey, buy, recharge	8	6	0	1

3 Simulation

The teams run their systems locally. The simulated environment runs on the MASSim Server. Agents communicate with the server by exchanging XML messages. Three phases are distinguished, namely the initial phase, the simulation phase and the final phase. While the first and last ones control connection establishment and termination, the simulation phase is quite of interest. The main steps are the following:

- collect all actions from the agents,
- let each action fail with a specific probability,
- execute all remaining attack and parry actions,
- determine disabled agents,
- execute all remaining actions,
- prepare percepts,
- deliver the percepts.

The percepts hereby are

- State of the simulation (step)
- State of the team (score and money)
- Internal attributes
- Visible vertices (+ dominating team)
- Visible edges
- Visible agents (vertex, identifier, team)
- Returned values of probe, survey, and inspect

4 Conclusion

The Agent on Mars scenario demands for a sophisticated communication and cooperation amongst the agents in order to win a game. The goal is to maximize the team's score, calculated by adding the zones' values and the money for each simulation step:

$$score = \sum_{s=1}^{steps} (zones_s + money_s)$$

In my talk I briefly talked about ELMS. ELMS is an environment description language for multiagent systems. Please see [OHC05] for detailed information.

References

[BDH+11] Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, Federico Schlesinger: Multi-Agent Programming Contest 2011 Edition - Evaluation and Team Descriptions, Technical Report IfI-12-02, Clausthal University of Technology, September 2012.

[MAPC] Multi-Agent Programming Contest. Scenario Description (2013 Edition). URL: https://multiagentcontest.org/downloads/func-startdown/1663/

[BDD+12] Tristan Behrens, Mehdi Dastani, Jürgen Dix, Jomi Hübner, Michael Köster, Peter Novák, Federico Schlesinger: The Multi-Agent Programming Contest, AI Magazine, Volume 33, Number 4, pg 111-113, 2012.

[OHC05] Fabio Y. Okuyama, Rafael H. Bordini, and Antônio Carlos da Rocha Costa. ELMS: An environment description language for multi-agent simulation. Environments for Multi-Agent Systems. Springer Berlin Heidelberg, 2005. 91-108.