Multi-Agent Programming Contest 2016

Tobias Ahlbrecht, Jürgen Dix, Niklas Fiekas

Department of Informatics, Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
E-mail: {tobias.ahlbrecht, dix, niklas.fiekas}@tu-clausthal.de

Abstract: We present the eleventh edition of the Multi-Agent Programming Contest[1], an annual, community-serving competition that attracts groups from all over the world. Our contest enables head-to-head comparison of multi-agent systems and supports educational efforts in their design and implementation.

Keywords: Multi-Agent Systems; MAS Programming; Contest.

Biographical notes: Jürgen Dix is professor for Artificial Intelligence and Dean of the Faculty at Clausthal University of Technology. In the past 30 years he worked on basic research in knowledge representation and reasoning, deductive databases and multi-agent systems. He co-authored or co-edited more than 20 books, over 70 journal publications and organized/chaired more than 40 conferences and workshops. He is on the editorial boards of seven journals and several steering committees.

Tobias Ahlbrecht is a Ph.D. student in the Computational Intelligence Group at the Department of Informatics of Clausthal University of Technology. He earned his master's degree in Computer Science from the same department in 2016. He is a co-organizer of the Multi-Agent Programming Contest since 2013 and his current research interests include multi-agent systems modeling and simulation.

Niklas Fiekas is a graduate student in Clausthal, Germany. He earned his bachelor's degree in Computer Science from Clausthal University of Technology in 2016. Previously, he was working on scalable multi-agent simulation in the "Decentralized Simulation" project within the Simulation Science Center Clausthal/Göttingen.

1 Introduction

In this preface to the special issue we (1) briefly introduce our Contest and its development in the last 10 years, (2) elaborate on the brand new 2016 scenario, (3) introduce the five teams that took part in the tournament, (4) analyze and interpret interesting matches, and (5) evaluate the performance and strategies of the teams.

The Multi-Agent Programming Contest (MAPC) is an annual international event that started in 2005, initiated by Jürgen Dix and Mehdi Dastani. It is an attempt to stimulate research in the field of programming multi-agent systems by 1) identifying key problems, 2) collecting suitable benchmarks, and 3) gathering test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. In 2016 the competition was organized and held for the tenth time.

[1] `http://multiagentcontest.org`

Detailed information about the strategies of the teams can be found in the subsequent papers in this volume.

1.1 Related work

For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to [2, 4, 6, 7, 1]. A quick non-technical overview appeared in [3].

Similar contests, competitions and challenges have taken place in the past few years. Among them we mention *Google's AI challenge*[a], the *AI-MAS Winter Olympics*[b], the *Starcraft AI Competition*[c], the *Mario AI Championship*[d], the *ORTS competition*[e], the *Planning Competition*[f], and the *General Game Playing*[g]. Each of these rests in its own research niche. Originally, our Contest has been designed for problem solving techniques that are based on formal approaches and computational logics. But this is not a requirement to enter the competition. In fact, there are almost always approaches that are not even based on agent technology (eg. the runner-up in this years contest).

1.2 History: The contest from 2005 to 2014

Through the history of the Contest, changes to the scenarios were introduced with every new edition including **four** major redesigns.

From 2005 to 2007, a classical *gold miners* scenario was used. We introduced the *MASSim* software: A platform for executing the Contest tournaments.

From 2008 to 2010 we developed the *cows and cowboys* scenario, which was designed to enforce cooperative behavior among agents [5]. The topology of the environment was represented by a grid that contained, besides various obstacles, a population of simulated cows. The goal was to arrange agents in a manner that scared cows into special areas, called corrals, in order to get points. While still maintaining the core tasks of environment exploration and path planning, the use of cooperative strategies was a requirement of this scenario.

In 2011, the *agents on Mars* scenario [6] was newly introduced. In short, the environment topology was generalized to a weighted graph. Agents were expected to cooperatively establish a graph covering while standing their ground in an adversarial setting and reaching certain achievements. The basics of the *agents on Mars* scenario remained until the 2014 edition[h], although several modifications were introduced to keep the Contest challenging.

In 2015, work on the current scenario, which is presented later in this paper, began. This new scenario was first used in the contest's 2016 edition.

1.3 History II: Participants and their origins

In its 11 editions, the Contest has seen 70 teams from all over the world. Figure 1 shows the development of team numbers over the years.

[a] `http://aichallenge.org/`
[b] `http://www.aiolympics.ro/`
[c] http://eis.ucsc.edu/StarCraftAICompetition
[d] http://www.marioai.org/
[e] http://skatgame.net/mburo/orts/
[f] http://ipc.icaps-conference.org/
[g] http://games.stanford.edu/
[h] The 2014 contest was an "unofficial" edition (i.e. no publications and prizes, only glory) with no changes to 2013.

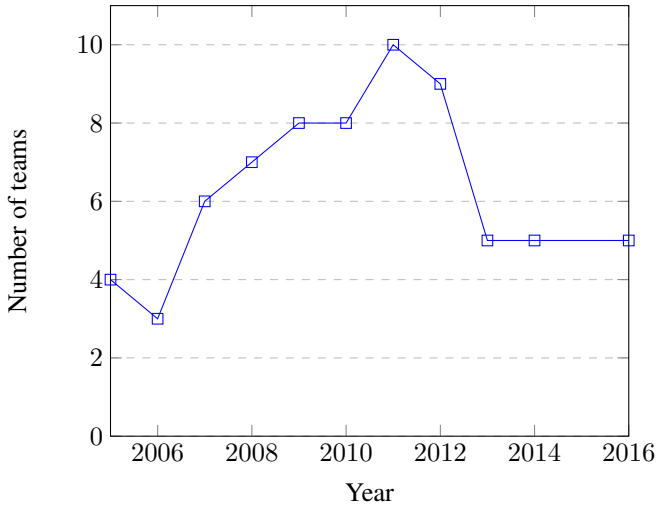

Figure 1: Participants of the Multi-Agent Programming Contest through the years

In its humble beginnings, the Contest started off with 4 teams in 2005 and 3 the year after. From then on, the number of teams monotonically increased until 2011, the best year yet with 10 participating teams. The number has gone down to 5 teams since and has remained there for 3 editions now.
The majority of teams originated from academia, while only 2 contestants competed without affiliation.
As of today, we are counting 19 different countries. The most frequently participating country, Germany, denotes a total of 22 attempts to win. This is mostly due to the tireless efforts from the Technical University of Berlin, who started in 2007 and have not missed a contest since, sometimes even contributing two teams in the same year. This has already fetched them the first place in 4 consecutive editions, starting right in 2007.
Closely following is Brazil, with 12 attempts and leading the list of winning countries with 5 first placements from 3 universities. The team from Federal University of Santa Catarina participated in every instance of the Mars scenario and won all 3 contests.
Brazil is immediately followed by 8 participations of Danish teams, all from Technical University of Denmark. They started participating in 2009, also not having missed a single contest since, while having contributed two teams in 2014.
Starting in 2008, the Contest also saw a team from University College Dublin for 5 consecutive years. In the same order of magnitude, 7 teams from Iran participated already, fully 4 teams just in 2011, 3 of which originated from Arak Univeristy, Iran.
Other sporadically participating countries, sorted by their first appearance, include the UK, Japan, Chile, Spain, Switzerland, the Netherlands, Australia, Poland, France, Turkey, Argentina, China, the U.S. and Greece.

2 MAPC 2016: The new scenario

The new scenario consists of two teams of agents moving through the streets of a realistic city. The goal for each team is to earn as much money as possible, which is rewarded for

completing certain jobs. Jobs comprise the acquisition, assembling, and transportation of goods. These jobs can be created by either the system (environment) or one of the agent teams.There are two kind of jobs: priced and auctioned. A team can accept an auctioned job by bidding on it. The bid amount of money is the reward. If both teams bid, naturally the lowest bid wins. If a job is not completed in time, the corresponding team is fined. Priced jobs have a reward defined upfront, that is given to the first team to complete that job. The teams have to decide which jobs to complete and how to do that, i.e. where to get the resources and how to navigate the map considering targets like shops, warehouses, charging stations, storage facilities.

A *team* consists of different types of agents. The agents differ in their speed, how they move around the city, battery charge, the volume of goods they can carry, and which tools they can employ to craft other items. Currently we have 4 agent roles: cars, trucks, motorcycles, and drones.

Items can be bought, crafted, given to a teammate, stored, delivered as part of a job completion, recovered from a storage facility, and dumped. These actions may happen at their respective specific locations/facilities. The crafting of an item requires the use of other products, some of which serve as prime matter and some as tools. Since each kind of agent can only handle a subset of the tools, the crafting of some items requires the explicit collaboration of 2 or more agents.

Agents feature a battery charge that gets decreased as they move around from one place to another. They need to make sure they never run out of charge, and therefore should plan their visits to the charging stations accordingly. Moving from one place to another, as well as recharging the battery at a charging station, are actions that are carried on only partially on each step, an may require several steps for completion.

Tournament points are distributed according to the amount of money a team owns at the end of the simulation. To get the most points, a team has to beat the other, as well as surpass a certain threshold.

3 The tournament

Following the tradition, a *qualification round* was held prior to the tournament, in which teams were required to show that they were able to maintain good stability (i.e. timeout-rates below 5%) during a round of test matches. Only then were they allowed to take part in the tournament. The qualification rounds showed extremely positive results: each and every team encountered not only a single timeout.

3.1 Participants and results

Five teams from around the world registered for the Contest and were able to pass the qualification round, thus taking part in the tournament (see Table 1).

BathTUB: The team *BathTUB* [**?**] from Technical University Berlin, Germany, is a regular contender of the Multi-Agent Programming Contest. Their agents are once again developed with the JIAC V platform (which won the contest several times in previous years). This time, six students and their supervisor have spent around 1000 man-hours developing their agent team. The approach is partly centralized, each agent being coordinated by a central instance. To complete jobs, agents request proposals from each other and reason locally how to retrieve the necessary items.

Team	Affiliation	Platform/Language
BathTUB	Technical University of Berlin	JIAC V
Flisvos 2016	none	Python
lampe	Clausthal University of Technology	C++
PUCRS	Pontifical Catholic University of Rio Grande do Sul	Jason, CArtAgO, Moise
Python-DTU	Technical University of Denmark	Python

Table 1 Participants of the 2016 Edition.

Flisvos 2016: The team *Flisvos 2016* [**?**], consisting only of a single person, participated for the first time in the Contest and promptly made the second place, losing only two simulations against this year's winners. The agents were implemented in Python, having invested roughly 250 hours. The centralized approach relies on no special agent-related concepts.

lampe: The two people of team *lampe* [**?**] from Clausthal University of Technology developed their agents in C++, spending about 150 man-hours. They also rely on a centralized approach together with a heuristic for choosing profitable jobs.

PUCRS: The team *PUCRS* [**?**] from Pontifical Catholic University of Rio Grande do Sul won this year's contest, only losing a single simulation against the runner-up. Approximately 230 man-hours were invested by the eleven members into developing the agent team in JaCaMo (Jason, CArtAgO and Moise). The agents divide jobs into tasks and distribute them with the well-known Contract Net Protocol, thus realizing a decentralized solution.

Python-DTU: The team *Python-DTU* [**?**] from the Technical University of Denmark is another regular contender of the Multi-Agent Programming Contest. After having tried GOAL in the 2013 (and 2014) edition, as the name suggests, the team changed back to using Python for this Contest. The four members spent around 400 man-hours developing their agents, not using any existing multi-agent technique or framework but plain Python instead. As most of the other teams, they chose a centralized approach.

The tournament took place on the 12th and 13th of September, 2016. Each day each team played against two other teams so that in the end all teams played against each other. We started the tournament each day at 13pm and finished around 6pm. A match between two teams consisted of 3 simulations differing in the map that was used and the properties of jobs that were offered by the system.

The teams were awarded 3 points for winning a simulation (minus 1 point if they did not make a profit) and 1 point in case of a draw. The results of this year's Contest are shown in Table 2.

PUCRS secured an almost flawless victory, only losing a single simulation against runner-up *Flisvos 2016*, thus scoring 33 out of 36 possible points. *Flisvos 2016* in turn only lost two of the simulations against *PUCRS*, resulting in 30 points total. Nevertheless, *Flisvos 2016* had a better score difference, which may be attributed to variations in the simulation instances (see Subsection 3.2 for details).

Pos.	Team	Score			Difference	Points
1	PUCRS	6,503,165	:	994,109	5,509,056	33
2	Flisvos 2016	7,197,893	:	941,069	6,256,824	30
3	lampe	1,320,153	:	1,601,549	-281,396	12
4	Python-DTU	532,714	:	7,764,645	-7,231,931	6
5	BathTUB	-625,240	:	3,627,313	-4,252,553	5

Table 2 Results.

Following with a rather significant margin, team *lampe* made the third place with 12 points, still one out of each three possible points. The points were gained from two victories each against *BathTUB* and *Python-DTU*.
Making for a close battle for the fourth place, *Python-DTU* was able to take it with a one point lead over *BathTUB*. Having the edge in their match against each other, *Python-DTU* won 2 out of the 3 simulations. However, no team was able to make a profit, resulting in 4 and 2 points respectively. Both teams were able to win one simulation against team *lampe*, resulting in 2 additional points for *Python-DTU*, and even 3 points for *BathTUB*, since they made a profit in that simulation.

3.2 Simulation definition

As already mentioned, a match between two teams consisted of three simulations comprising 1000 steps each. These three simulations differed in the map that was used and the form of jobs that were generated. However, each match featured the same three sets of simulation parameters.
The first of each set of simulations was played on the street graph of (a part of) London. Jobs had to be completed in 250 to 350 steps. The jobs' rewards were comparatively the lowest of all three simulations.
For the second simulation, played on a map of the German city Hannover, the job completion time bounds were decreased by 25 steps each and the potential job rewards increased.
The same adjustments were made for the third simulation, played on the San Francisco street graph, again decreasing completion times and increasing job rewards.

To summarize, in one match it became (on average) more difficult but at the same time more rewarding to complete a job.

Note that, while the simulation sets featured the same parameters, of course the concrete simulation instances that were played by the teams were (with a very high probability) never the same due to the random generation mechanism. The goal was, as in previous editions of the Contest, to generate a set of similar simulation instances, so that one could compare the simulations afterwards without risking to give the agents the possibility to learn the exact simulation from match to match (e.g. the prices of items or what job will be generated in which step).

3.3 Strategies

In this section we collect a few more facts about the participants and their agent team strategies. For more detailed information, we refer to the team description articles [**? ? ? ? ?**] in these proceedings.

PUCRS: The *PUCRS* agents start each simulation by exploring nearby shops to get information about the prices of items. The exploration is coordinated with a token ring communication, where each agent places its route to each facility if it is the shortest one yet.
After that, they start evaluating the incoming jobs and estimate their costs (for recharging the agents and buying or assembling the items). If a job's reward surpasses the cost, the agents try to complete it.
This is achieved by splitting the job into tasks which are distributed among the team members using the Contract Net Protocol.
In addition, the agents try to deceive the opponent agents by posting their own jobs, rather to distract them than to outsource a task.
Nearing the end of the simulation, the agents adjust their strategies

Flisvos 2016: The team uses a centralized approach, employing a global planning technique together with known optimization heuristics. The agents only communicate by updating a shared data structure in order to exchange percept information.

lampe: This team also uses a centralized approach for their agents, which are controlled by a "mother-ship". Jobs are evaluated with a heuristic method.

Python-DTU: The team also chose to employ a centralized approach.
Simulations have shown the agents to always spend a certain amount of money (a few hundred up to 15,000) and then stopping any noticeable behavior. In one particular match, the team tried a new strategy and posted up to 16 new (nonsensical) jobs in each step[i].

BathTUB: Regarding centralization, the team used a hybrid approach for coordination, whereas information was handled completely decentralized.
It was planned to complete jobs as fast as possible. To achieve that, a similar approach to *PUCRS* was chosen: Any agent may initiate the planning of a job and receives proposals from all the agents, i.e. which items they can procure at which cost. The initiating agent then decides on the optimal course of action and informs the other agents.
To prevent idling, a measure for proactiveness was implemented: the agents explore the map, plan a new job or keep watch on the opponent agents.

4 Performance of the teams

In this section, we will look at how the teams performed in the Contest regarding scores, completed jobs, stability and how they used certain aspects of the scenario.

4.1 Score and completed jobs

The score or the team's money is mostly depending on how many jobs the team completed and how economical the agents did it, as completing jobs is the only source of income in the scenario. Money is decreased through buying items for the jobs or recharging the agents

[i] Fortunately, only the web monitor had to be quickly patched to accommodate for the increased data volume. The *MASSim* server and the opponent team were able to handle the situation well.

when their energy gets low.
The scores and the number of completed jobs allow to assess a team's overall performance and compare it to other teams on the simulation level.[j]

PUCRS: The Contest winner completed 343 jobs in total, earning a money value of $18,221,172$, or $53,122$ on average per job.
Comparing by simulation, the biggest difference in completed jobs was to *Python-DTU*, the smallest to *Flisvos 2016*. Somewhere in between, we see similar numbers for the simulations against *BathTUB* and *lampe*. *PUCRS* completed more jobs than the opponent team in all but one simulation against *Flisvos 2016* (also being the one simulation they lost). However, in two simulations, *Flisvos 2016* earned more money, which means *PUCRS* was a little more efficient in choosing and completing jobs, as they won one of these simulations anyways. Namely, in their second simulation, *PUCRS* completed 13 jobs more, earning on average $11,404$ (vs. $21,910$ for *Flisvos 2016*), showing that *PUCRS* focused on smaller jobs.

Flisvos 2016: The runner-up completed 402 jobs, earning a total sum of $23,247,534$.
Of course, the smallest difference in completed jobs shows in the simulations against *PUCRS*. However, the numbers against all other teams look kind of similar. *Flisvos 2016* was able to complete more jobs than any opponent team in all but two simulations. Correspondingly, there was only one simulation where *Flisvos 2016* earned less money than the opponent.
Against *lampe* and *BathTUB*, the team completed smaller jobs in comparison in two out of three simulations each. In contrast, against *PUCRS* the average reward of jobs completed by *Flisvos 2016* was higher in all 3 simulations.
This leaves us with two possible conclusions. Either, *Flisvos 2016* was second best in decomposing the agents into smaller teams, thus completing more jobs in parallel, or the smaller size of jobs can be attributed to efficiency, being able to finish jobs with a smaller reward and still making appropriate profit of it (but most likely a combination of both).

lampe: The team completed rather few jobs compared to the previous two. Also, it finished less jobs than *BathTUB* in all of their three simulations, winning two of them regardless. In their first simulation, they earned less money on average per job (and in total) than *BathTUB*, resulting in the lost simulation. Interestingly, in their second simulation, they still earned less money in total, but more on average per job. Having spent way less money than *BathTUB*, this win went to *lampe*.
In two out of twelve simulations, one against *PUCRS* and one against *Python-DTU*, *lampe* did not complete any job. In the first case, they finished with their starting capital, indicating some kind of one-time bug.
Looking at the averages again, *lampe* aimed for comparatively high job rewards, pointing to a rather conservative heuristic. Only in the first simulation against each team the average reward for *lampe* was smaller compared to the opponent's.

Python-DTU: Weirdly enough, the fourth-placed team did not complete a single job. They won two simulations against *BathTUB* in which the opponent team finished with a big negative score and one against *lampe* with a monetary lead of roughly $2,000$.

[j] Due to the random generation of simulations, score (i.e. money) comparisons on the Contest level can only serve as estimations.

BathTUB: The team completed more jobs than *lampe* or *Python-DTU*, in total 54. Unfortunately, the agents could not make a profit in any simulation on the first day of the Contest. On the second day, they showed a varying performance. For example, against *Flisvos 2016*, *BathTUB* was able to make a good profit ($+134,763$) followed by a rather big loss ($-558,096$) and only a small loss ($-12,742$). In all of these simulations, the team completed a similar amount of 4-6 jobs, pointing to rather unpredictable investments. Their average reward against *Flisvos 2016* was higher in two out of three cases, however, they completed far less jobs.
Against *lampe*, they won the first simulation, in which they completed jobs with a higher average reward. In the following two simulations, they still completed more jobs but with a lower average reward than their opponent and lost both of those, again with varying final scores: some profit in the first simulation, some loss in the second and a good (but not big enough) profit in the third.

Having a final look at the completed jobs overall, none of them has been posted by any competing team, i.e. unfortunately the teams did not try or succeed in fooling their opponents into doing some work for them.

4.2 Agents' behavior

In this section, we will have a look at what actions the agents used to which extent and how that possibly affected the outcome of single simulations and the Contest as a whole.
As each team consisted of 16 agents, there were $16,000$ actions in each simulation per team. A big part of the actions of all teams is naturally made up of the actions `skip`, `continue` and `abort`, as those are necessary to maintain the also frequently used actions `goto` and `charge`.
Also, the actions `store`, `retrieve`, `give` and `retrieve` were not used at all, indicating the direction in which the scenario needs to be improved. Total action counts for all teams can be found in Table 3.

4.2.1 PUCRS

The team used the actions corresponding to job activities according to previous observations. Comparing these job related actions, as depicted in Figure 2 and 3 for the first and second simulation against *Flisvos 2016*, the numbers go in line with *PUCRS* having completed the most jobs in the Contest. Looking at the first simulation (where *PUCRS* lost), we see that *PUCRS* used more `deliver` but less `buy` actions than the opponent. However, around 50% of *PUCRS*' `deliver` actions against *Flisvos 2016* failed, many of them due to the job not being active (possibly already completed by *Flisvos 2016*) or because the delivering agent did not carry any items that were still missing for the particular job. Thus we can conclude that *PUCRS* employed less partial deliveries than *Flisvos 2016*, as they needed less (successful) actions to complete more (or comparable amounts of) jobs.

From the same Figures, we can see that *PUCRS* did not use any actions for assembling products (making it another action to promote in future editions of the Contest). Indeed, *PUCRS* did not use the actions `assemble`, `assist_assemble`, `retrieve_delivered` and `bid_for_job` for the entirety of the Contest (in addition to the actions that were never used by any team).

The `buy` action was used the most by *PUCRS* which also agrees with them having completed the most jobs. They were also the only team to use the `dump` action, however,

	Flisvos 2016	lampe	BathTUB	PUCRS	Python-DTU
goto	5224	3721	26860	7134	4892
noAction	0	35824	7823	1855	4010
skip	169034	192	153588	137936	142011
abort	0	108491	32	0	0
continue	0	19245	0	35017	3932
charge	2042	2951	1885	2326	3910
retrieve	0	0	0	0	0
retrieve_delivered	0	406	0	0	0
store	0	0	0	0	0
dump	0	0	0	36	0
deliver_job	1219	274	406	1764	0
buy	1695	342	1166	2044	164
assemble	0	10010	0	0	17734
assist_assemble	0	10269	0	0	12483
give	0	0	0	0	0
receive	0	0	0	0	0
call_b.	12824	467	0	3338	0
post_job	154	0	0	742	3056
bid_for_job	0	0	432	0	0

Table 3 Total action counts

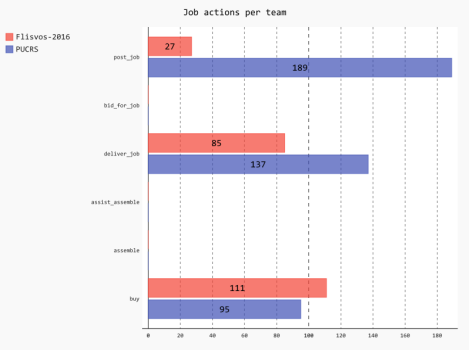

Figure 2: Job actions - *Flisvos 2016* vs. *PUCRS*- simulation 1 of 3

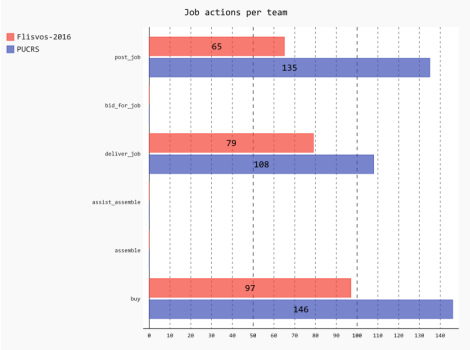

Figure 3: Job actions - *Flisvos 2016* vs. *PUCRS*- simulation 2 of 3

there were only 36 cases where they saw the need to dispose of some items, 20 of them in the match against *Flisvos 2016*. The `post_job` action was used sparingly in order to distract the opponent teams by adding nonsensical jobs and therefore consuming some of their processing time.

Finally, the team was one of the few to use the `call_breakdown_service` action regularly, counts reaching from 100 to 500 times per simulation. However, as the action has to be called repeatedly (25 times if it does not fail randomly) to have an effect, this amounts roughly to each agent being left without charge 0 to 2 times per simulation.

4.2.2 *Flisvos 2016*

The action counts of team *Flisvos 2016* are overall similar to *PUCRS*. The actions `abort` and `continue` were not used, the latter being replaced by more `skip` actions. The team did not make use of the storage actions `retrieve`, `store` and `dump`. Also, `bid_for_job` and `retrieve_delivered` were not used, thus ignoring auction jobs and the possibility to retrieve (unsuccessful) partial deliveries.
Unfortunately, the explicitly cooperative actions `give`, `receive`, `assemble`, `assist_assemble` were also not used at all by *Flisvos 2016* (again, possibly not being pushed enough by the concrete simulation instances).

Flisvos 2016 was one of three teams to use the `call_breakdown_service` action, even three times more often than *PUCRS*, indicating either a tiny routing/planning problem or that the team accepted to use the action in order to complete some otherwise unachievable jobs.

The `post_job` action was also used, however, more moderately than by *PUCRS*. As no team completed a job posted by an opponent team, the team did not need to use the `retrieve_delivered` action, as noted above.

Comparing the job related action counts to those of *BathTUB*, e.g. for their first simulation as given in Figure 4, again the difference in `buy` and `deliver` actions becomes obvious.

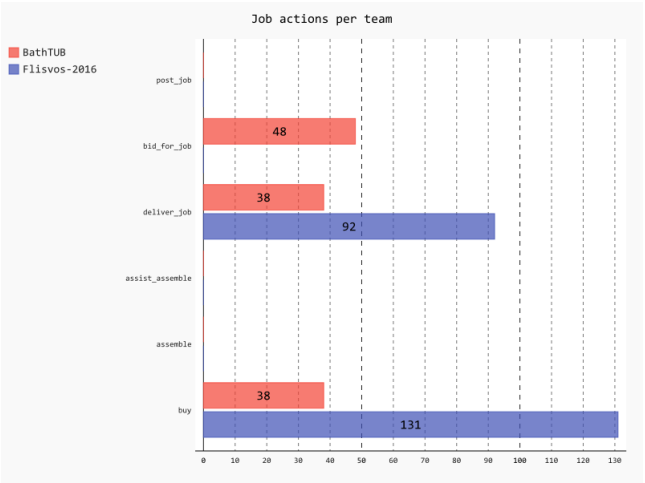

Figure 4: Job actions - *BathTUB* vs. *Flisvos 2016*- simulation 1 of 3

4.2.3 *lampe*

The *lampe* team decided to use the `continue` action in combination with `abort` in place of `skip`, thus being the only team to make sure that ongoing actions are explicitly stopped.

Interestingly, the team was the only one attempting to use the `retrieve_delivered` action (however, only making up ca. 0.2% of their actions). Unfortunately, the action only succeeded 20 times out of 406 for them.

The *lampe* agents did not use the `give` and `receive` actions, but they performed the most successful `assemble` (and accordingly `assist_assemble`) actions, which allowed them access to more valuable jobs. The ratio of these actions was almost $1:1$ suggesting that most often exactly two agents cooperated.

Of the teams to use the `deliver_job` action, this team used it the least, again highlighting their conservative heuristic which made the agents only work on the most rewarding jobs while forgoing a lot of smaller ones.
For example, much of this can be seen in their third match against *BathTUB*, with both teams' actions compared in Figure 5 and the failed actions of team *lampe* in Figure 6.

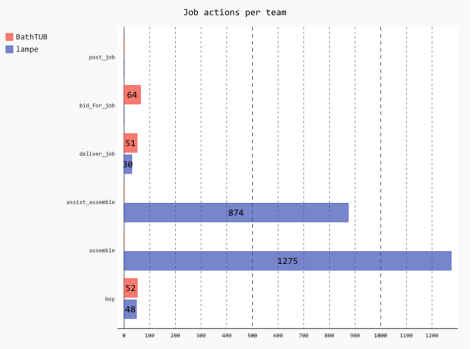

Figure 5: Job actions - *BathTUB* vs. *lampe*- simulation 3 of 3

Figure 6: Failed job actions *lampe*- *BathTUB* vs. *lampe*- simulation 3 of 3

We see that *lampe* used fewer delivery actions but had some successful assemblies. Comparing this to the money development in that simulation, as charted in Figure 7, one can see again how it paid off (here) to complete jobs with higher rewards on average, especially halfway through the simulation, where *BathTUB* only little more than broke even. Counting the increments, *BathTUB* indeed completed one job more than *lampe* in this simulation.

4.2.4 Python-DTU

The team *Python-DTU* used all the common actions to a "normal" degree, featuring a relatively high amount of `skip` actions.

Regarding storage actions, the team did not use any of them (e.g. `retrieve` or `deliver`). Also, as with any other team, `give` and `receive` where not used. As for the remaining actions, the team did not make use of `bid_for_job` and - surprisingly - `deliver_job`.

The team used even less `buy` actions than *lampe*: only 164 during the whole Contest. However, surprising us again, the team used the most (`assist-`)`assemble` actions out of all teams, even surpassing team *lampe* by roughly 77%. Recall that those were the only two teams to use these actions at all.

Probably as an attempt to completely occupy the opponent agents, the team used a comparatively high amount of `post_job` actions. Fortunately, only the simulation web monitor seemed to struggle with that much new information and could be quickly repaired. This behavior was only observed in certain simulations, e.g. the first one against *lampe*, as depicted in Figure 8. In simulations where the *Python-DTU* team used less `post_job`

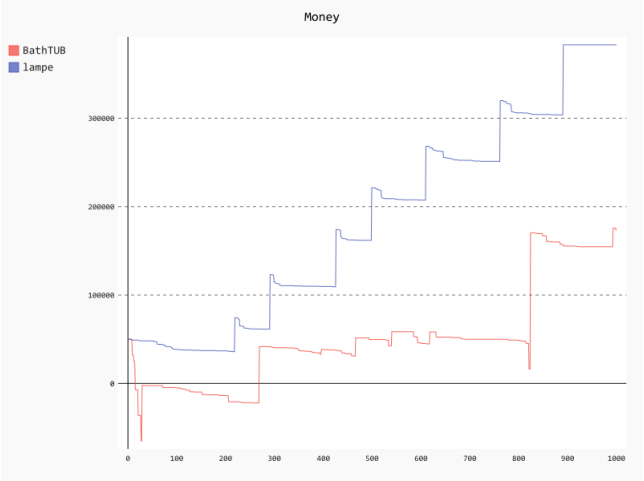

Figure 7: Money - *BathTUB* vs. *lampe*- simulation 3 of 3

actions, the amount of `assemble` actions was noticeably higher. In the accompanying Figure 9 we can also observe that the majority of `assemble` actions failed.

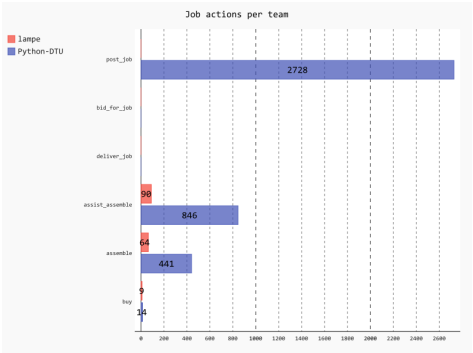

Figure 8: Job actions - *lampe* vs. *Python-DTU*- simulation 1 of 3

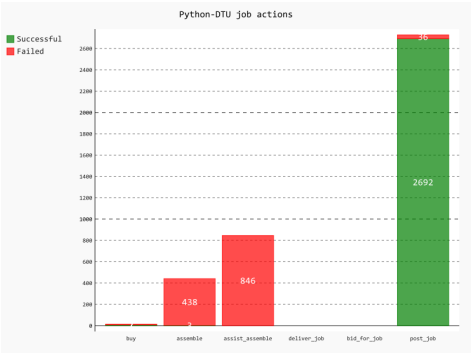

Figure 9: Failed job actions *Python-DTU-lampe* vs. *Python-DTU*- simulation 1 of 3

4.2.5 BathTUB

The *BathTUB* team also used no common action to an unusual degree. As the others, the team did not use the storage actions, nor any of the cooperative assembly and item exchange actions.

Like *Python-DTU*, the team did not need the `call_breakdown_service` action. Also, the *BathTUB* agents did not make use of the `post_job` feature. On the other hand, they were the only ones to use the `bid_for_job` action, which did not fail in most of the cases. Unfortunately, none of these auction jobs was completed successfully.

The amount of `buy` actions for team *BathTUB* lies somewhere beneath that of *PUCRS* and *Flisvos 2016*, but considerably above that of *lampe* and *Python-DTU*. The effect of

these spendings can e.g. be observed in the second simulation of *BathTUB* against *lampe*, as shown in Figure 10.

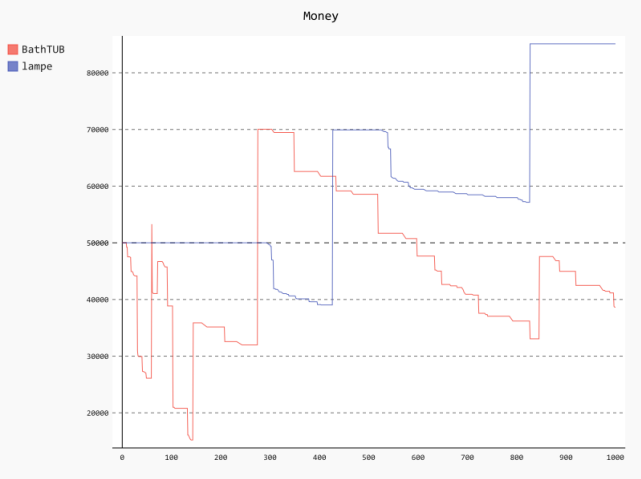

Figure 10: Money - *BathTUB* vs. *lampe*- simulation 2 of 3

Shortly after step 400, the teams are almost neck-and-neck, however, big investments of *BathTUB* are not translated into greater rewards from then on.

4.3 Agents' reliability and stability

In the last section, we mostly analyzed what the agents *tried* to do. Now, we will have a look at the extent to which the agents submitted correct and *sensible* actions to the system. For example, it does not make sense for an agent to perform a `charge` action if that agent is not currently located within a charging station.

First off, the amount of `noActions`, which are registered if an agent does not submit an action before the timeout, was considerably higher than in the qualification phase. In fact, *Flisvos 2016* was the only team to keep a flawless record of 0 `noActions`. The most timeouts were encountered by team *lampe*, a solid 18.66% of their total actions. On the first day of the Contest, their agents had a bug preventing them from reconnecting to a match once they lost connection for the first time. Thus, almost half of their 35,824 `noActions` originated from a single simulation, their third one against *PUCRS*.

The second most `noActions` were had by team *BathTUB*, roughly a fifth of those of team *lampe*. *Python-DTU* follows with half of that amount (i.e. ca. 4000) and *PUCRS* with a little less than half of that again.

The actions that were received by the system could potentially fail due to a number of reasons. Here, we will have a look at the causes of failed actions and how often they were experienced by the teams.

failed_location: This happens whenever an agent tries to perform a location-specific action outside of that location, i.e. charging, assembling or buying. It happened only a single time to *PUCRS* and about 150 times to *Python-DTU*.

failed_unknown_item: This error occurs if an agent specifies a non-existing item as parameter to an action. It happened only a single time to *PUCRS*.

failed_unknown_facility: This is the same as above, but for facilities. It also occurred only once for team *PUCRS*.

failed_item_amount: This occurs whenever an agent tries to use more items for an action than it currently carries. It happened 13 times to *PUCRS* and about $12,000$ and $18,000$ times for *lampe* and *Python-DTU* respectively, making it the most frequently occurring failure code. We can assume that both of the latter teams had a rather serious planning problem, as these numbers explain the previously seen counts of failed `assemble` actions.

failed_tools: Whenever a group of agents tries an `assemble` action without carrying the necessary tools, this failure occurs. Again, this occurred to both *lampe* and *Python-DTU*, about 7000 and $11,500$ times respectively, making it the other big cause of failed `assemble` actions. As "failed_tools" takes precedence over "failed_item_amount" and both cases could be satisfied at the same time, these numbers should be taken into account at the same time.

failed_capacity: This occurred whenever an agent did not have enough inventory space to obtain an item. It happened close to 600 times for each *BathTUB* and *PUCRS*.

failed_job_status: This code indicates that an agent tried to either bid for a job that's not up for auctioning, or deliver items to a job which has already been completed by the opponent team. Concerning *Flisvos 2016*, this was their only self-inflicted failure and on top only 2 times, probably due to *PUCRS* completing a job faster. Vice versa, the same thing happened to *PUCRS* 239 times, almost only against *Flisvos 2016*, indicating the same reason. The third and last team to experience the failure code was *BathTUB* with 85 actions.

failed_counterpart: This occurs whenever an agent tries to assist an assembling agent, but any requirement for assembling is not satisfied. Both teams who tried to assemble items, *Python-DTU* and *lampe*, experienced this failure on a minor scale, i.e. 146 and 328 times respectively.

failed_random: With a chance of 1%, any action might fail and get this result. By the nature of this failure, all teams had a similar rate of around 1% of their total actions.

useless: An agent causes this failure if it tries to deliver items towards the completion of job, but does not possess any item that is still needed therefor. Both *Flisvos 2016* and *Python-DTU* did not cause this failure at all. *BathTUB* tried this only 35 times, *lampe* 74, and *PUCRS* a surprising 438 times.

Finally, the failure codes for invalid agent and job parameters, for an invalid location passed to the `goto` action, for being in the wrong facility, for attempting to assemble an item that cannot be assembled, for bidding on a job that is not an auction, and for using wrong parameter types were not encountered by any agent during the Contest.

4.4 Feature usage

As already suggested, there are a number of features of the scenario which have been used only to a lesser degree or not at all. Since it was the first time the new scenario was played, it was not entirely clear how the participating teams would handle the different aspects.

For one, the teams mostly did not use the assemble action very much. In preparation of the Contest, we balanced job rewards so that only by assembling items themselves the teams would be able to net a notable profit. However, in previous tests, the testing agents were not reliably earning money, which led us to the decision to increase base rewards so that buying assembled items in shops still remained a viable though unfavorable option.

On the other hand, auction jobs were neglected as well, possibly due to an abundance of regular jobs, which do not have a potential penalty attached (though of course any regular job comes with the risk of the opponent team completing it faster).

The storage facilities were also not used. Probably, the teams were mostly able to deliver the items they bought for their jobs on demand, while stocking up on certain items up front would have been more risk than reward.

Regarding jobs, the teams have used the `post_job` action, yet only to divert the opponent team's attention and add to the information those agents have to process, instead of outsourcing some of their own work.

5 Interesting simulations

In this section, ...

6 Summary, conclusion, and going forward

In conclusion, we have seen an interesting Contest and a solid first run of the new scenario which we can use as a foundation for future improvements. Judging from the previous section, we need to put some effort into pushing the scenario into a more cooperative direction. One way of doing this is to put more emphasis on or even enforce the use of the `assemble` action and the related systems. A rework of this is already in progression and should become ready in early 2017.

Also, the scenario still leaves room for more contention among the opposing teams. This year, the teams could mostly just work alongside each other without having much influence on the other team's possibilities.

As we are already looking forward to the next edition of the Contest in 2017, we have to admit that our marketing is still capable of development. With a stagnating number of participants since 2013, which had its peak in 2011, this has also moved up on our agenda.

One thing that is also on our list is adding more than two teams to the same simulation. The current scenario would provide for this naturally, however, the underlying technical system has evolved with only two teams in mind, making this a rather big undertaking.

Acknowledgments

We would like to thank Alfred Hofmann from Springer for his support right from the beginning and for endowing the prices of 500 and 250 Euros in Springer books.

References

[1] Tobias Ahlbrecht, Jürgen Dix, Michael Köster, and Federico Schlesinger. Multi-agent programming contest 2013. In Massimo Cossentino, Amal El Fallah-Seghrouchni, and Michael Winikoff, editors, *Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers*, volume 8245 of *Lecture Notes in Computer Science*, pages 292–318. Springer, 2013.

[2] Tobias Ahlbrecht, Jürgen Dix, and Federico Schlesinger. From Testing Agent Systems to a Scalable Simulation Platform. In Thomas Eiter, Hannes Strass, Mirosław Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation. Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2014.

[3] T. Behrens, M. Dastani, J. Dix, J. Hübner, M. Köster, P. Novák, and F. Schlesinger. The multi-agent programming contest. *AI Magazine*, 33(4):111–113, 2012.

[4] Tristan Behrens, Mehdi Dastani, Jürgen Dix, Michael Köster, and Peter Novák, editors. *Special Issue about Multi-Agent-Contest I*, volume 59 of *Annals of Mathematics and Artificial Intelligence*. Springer, Netherlands, 2010.

[5] Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition: 4th edition. In Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors, *Programming Multi-Agent Systems, 6th International Workshop (ProMAS 2008)*, volume 5442 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2009.

[6] Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. Multi-agent programming contest 2011 edition documentation. Technical Report IfI-12-01, Clausthal University of Technology, April 2012.

[7] Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. Multi-agent programming contest 2011 edition evaluation and team descriptions. Technical Report IfI-12-02, Clausthal University of Technology, April 2012.