Constrained Coalition Formation among Heterogeneous Agents for the Multi-Agent Programming Contest

Tabajara Krausburg and Rafael H. Bordini
School of Technology, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil

Keywords: Coalition Formation, Multi-Agent Systems, JaCaMo, Multi-Agent Programming Contest.

Abstract: This work focuses on coalition formation among heterogeneous agents for a simulated scenario involving logistic and coordination problems. We investigate whether organising a team of agents into a number of coalitions, in which agents collaborate with each other to achieve particular goals, can increase the effectiveness of the team. We apply coalition structure generation specifically to the 2017 multi-agent programming contest, where the agents controlling various autonomous vehicles form a competing team that has to solve logistic problems simulated on the map of a real city. We experiment on three approaches with different configurations. The first uses only a task-allocation mechanism, while the other approaches use either an optimal or a heuristic coalition formation algorithm. Our results show that coalition formation can improve the performance of a participating team under some circumstances. In particular, coalition formation can indeed play an important role when we aim to balance the skills in groups of agents selected to accomplish some given set of tasks given a larger team of cooperating agents in the presence of dynamically created tasks.

1 INTRODUCTION

In this work, we focus on Coalition Formation (CF) among heterogeneous agents applied to a simulated scenario involving logistic and coordination problems. In order to increase the effectiveness of agents, we can organise them into groups, in which the agents collaborate with each other in order to achieve individual, common, or global (i.e., system-level) goals. Such organisation of agents is often called *coalition*. A coalition is a short-lived and goal-directed structure, in which the agents join forces to achieve a goal (Horling and Lesser, 2004). Coalitions are also used in the context of cooperative game theory (Wooldridge, 2009). From this point of view, a coalition is a set of agents who may work together. Each coalition obtains a certain utility represented by a numeric value. We try to maximise the overall value of all coalitions in the environment.

Although many applications have used CF to address real-world problems, they often do not evaluate their techniques against other approaches in realistic problems. This is important in order to assess whether CF is a suitable technique for a particular problem. In this work, we evaluate Coalition Structure Generation (CSG) on the 2017 Multi-Agent Programming Contest (MAPC) (Ahlbrecht et al., 2018). The Multi-Agent Programming Contest (MAPC) is an annual international event that aims to stimulate research in the field of programming multi-agent systems. We selected one of its participating teams; the chosen one is the SMART-JaCaMo team (Cardoso et al., 2018). We studied how to apply coalition formation techniques to its implementation and we evaluate the performance of the CF techniques on the 2017 MAPC scenario. In this domain, we have heterogeneous agents (controlling different types of autonomous vehicles) that must deliver jobs posted by a server; to do so they need to collect items in various locations on a real city map and assemble the parts before delivery. The chosen team uses a task allocation approach to address this problem and we introduce the formation of coalitions as a preceding step to task allocation.

The coalition formation process is integrated into the JaCaMo platform (Boissier et al., 2016) in a generic way, so it can be used in other projects too. We use two algorithms for coalition formation designed for different purposes. The first one is an optimal coalition formation algorithm that takes into account constraints; it is named the *DC algorithm* and was proposed by Rahwan et al. (Rahwan et al., 2011). The second algorithm was proposed by Farinelli et. al. (Bistaffa et al., 2017a) and is a heuristic algorithm for coalition formation based on clustering al-

Krausburg, T. and Bordini, R.
Constrained Coalition Formation among Heterogeneous Agents for the Multi-Agent Programming Contest.
DOI: 10.5220/0007374501620169
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 162-169
ISBN: 978-989-758-350-6

gorithms; it is named the *C-Link algorithm*. This approach adds great value to the JaCaMo platform, in the sense that coalition formation algorithms can be applied in various different JaCaMo domain applications. As such, it can benefit researchers, lecturers, and students in their activities.

We perform several experiments focusing on the performance of the team while working on "jobs" issued by the competition server. We run the Coalition Structure Generation (CSG) algorithms in order to get the best partition of agents to work on the jobs; this way we are able to balance the agents for each job. When we take into account a set of jobs, we reason about all the jobs and how we can accomplish all of them instead of allocating the best agents to only one job. This is particularly important when jobs sporadically appear in the environment, so concentrating all the best-skilled agents in one job can be a bad strategy.

This paper is organised as follows. Section 2 introduces some relevant concepts related to the CF approach. In Section 3, we explain the basis of our work; then Section 4 details our approach in integrating CF into Multi-Agent System (MAS). Section 5 describes our experiments and results regarding the applied coalition formation techniques. We discuss some related work in Section 6 and Section 7 concludes the paper.

2 BACKGROUND

The coalition formation process could be divided into three phases: (i) the coalition structure generation; (ii) solving optisation problem of each coalition; and (iii) the division of the solution value between the coalition members[1] (Sandholm et al., 1999).

We formally describe the coalition formation process through Characteristic Function Games (CFG). In such games, we have a pair $\mathcal{G} = \langle A, v \rangle$, where A is the set of agents and v is a function that assigns a real value to every coalition, $v : 2^A \to \mathbb{R}$, which is called the *characteristic function* of the game. A coalition C is a subset of A ($C \subseteq A$). A coalition structure CS is a partition of A into mutually disjoint coalitions, i.e., for all $C, C' \in CS$, $C \cap C' = \emptyset$, $C \neq C'$ and $\bigcup_{C \in CS} C = A$. The value of a coalition structure CS is defined as $V(CS) = \sum_{C \in CS} v(C)$. This is the classic form of the game; alternatively, some extensions have been proposed (Rahwan et al., 2015).

[1]In order to divide the reward among the coalition's members, we can use *game theory* concepts such as the *shapley value* (Chalkiadakis et al., 2011).

An extension of CFG was proposed in (Rahwan et al., 2011) to take into account constaints during the coalition structure generation. The basic form of a Constrained Coalition Formation (CCF) game is defined as $\mathcal{G} = \langle A, \mathcal{P}, \mathcal{N}, \mathcal{S}, v \rangle$ in which $\mathcal{P}$, $\mathcal{N}$, and $\mathcal{S}$ are, respectively, the sets of positive, negative, and size constraints. In that approach, the feasibility of a coalition structure is determined by the coalitions that are member of it. In other words, the constraints are on each of the coalitions rather than the overall coalition structure. A coalition is considered feasible if it has at least one positive constraint, no negative constraint, and the size of the coalition is permitted. A feasible coalition structure has only feasible coalitions.

3 DOMAIN

Here, we explain the two basis of our work. The first is the 2017 MAPC (Ahlbrecht et al., 2018) simulation in which we perform our experiments on CF. The second is the SMART-JaCaMo (Cardoso et al., 2018), the publicly available code of one of the contestants, to which we integrate CF algorithms to enable its agents to group together in order to improve their joint performance. Note that we will not describe the contest scenario and the SMART-JaCaMo implementation in details; we will only discuss what is needed for comprehension of the work reported here (the reader can refer to the references above for further details).

3.1 The 2017 MAPC

The 2017 MAPC[2] is based on EIS (Behrens et al., 2012), a proposed standard for agent-environment interaction. Based on this standard, agents remotely connect to the contest server in order to receive the game state from it and to act on the game. In the contest domain, a team of agents try to earn as much "virtual money" as possible by delivering jobs announced by the game through the contest server. Almost all features of the simulation have *random variation* (except the ones related to the agents) and controlled by some configuration parameters. For instance, we can configure the number of agents, the number of steps, the number of different items, the rate of how often jobs are created and so on.

In the simulation, agents have 1000 discrete game steps to delivery jobs. A step is a cycle in which the server waits for the agents' action, executes them, and deliveries the perceptions of the new environment

[2]https://multiagentcontest.org/2017/

state to the agents[3]. Four types of agents are available: (i) car; (ii) drone; (iii) motorcycle; and (iv) truck. Each type of agent a contains different levels of skill in *speed*, *load*, *battery*, and *locomotion*. In our experiments each team has three agents of each type. The contest takes place on a map of a realist city (see Figure 1). The map is partitioned in a number of cells with sizes of 200 meters, which means an agent with speed 1 travels 200 meters in one step.

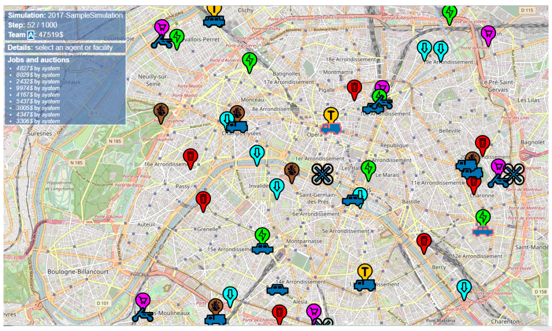


Figure 1: Map of Paris for the 2017 MAPC.

The jobs are generated randomly by the contest server. Each job requires a number of items to be assembled and the assembled product must be delivered at the storage indicated by the server. They also have a start and an end time, which means after the job's deadline the server does not accept the delivery of that job anymore. In this sense, the agents need to coordinate themselves to assemble and deliver all the items required by it. Items are identified by a name, a volume, a price[4], and requirements. These requirements might be other items and/or tools, in which case we call them *compound items*. The items that have no requirements are called *base items*. The compound items are assembled at *workshops* (yellow marks in Figure 1); for those that require tools, an agent of the specific type must be at the workshop carrying the required tool and performing an action to assist or assemble. Jobs only require compound items.

In summary, we choose the 2017 MAPC as a target application scenario for the following reasons:

- it has a stable/robust simulator;
- the domain is complex because it addresses many important issues in MAS and simulates a stochastic environment (virtually all variables and actions have a random component);
- it allows us to modify the number of agents, so we can develop our approach using a small number of agents before we work on scaling it;
- Once we have a stable version of our approach for this simulation, we can apply it to other complex domain (e.g., disaster response).

[3]The server can also throw away an action sent by an agent to simulate an action failure.

[4]The price of an item is determined according to the *shops*, where agents buy base items; each shop might have a different value for each item.

3.2 SMART-JaCaMo Team

We started our development on top of the SMART-JaCaMo team's implementation[5] for the contest, and we integrated CF techniques into it. It was implemented using JaCaMo as the MAS development platform (Cardoso et al., 2018). SMART-JaCaMo agents use the GraphHopper[6] API to calculate their routes to get to the desired positions. These routes are evaluated according the speed of each vehicle (e.g., a motorcycle is faster than a truck). When moving around the city, the vehicles consume battery that is recharged at *charging stations* (green marks in Fig. 1).

In order to finish jobs, agents must reach an agreement on what each agent should do to help deliver the job. This is done by means of task a allocation process implemented using Contract Net Protocols (CNPs) (Smith, 1980) in CArtAgO artefacts. When a job is perceived by the CNP initiator, it decomposes the main tasks into several subtasks:

- **deliver:** only trucks are allowed to place bids to deliver all items from the job (because they have the largest cargo capacity;
- **buying tools:** task to buy the tools required to assemble compound items.
- **buying base items:** task for buying the base items.

Bid values are based on the total route length to complete a task, thus lower is better (with the exception of bids with value -1, which are invalid bids; e.g., agent is not skilled to use a required tool). For tasks to buy tools and items, the agent also sends the *shop* where it expects to be able to buy them; this information is later used in bid selection.

After the bidding phase is completed, the initiator uses several bid selection rules to determine the winner, starting with the assemble tasks. These priority rules favour awarding tasks based on a few characteristics, such as (in order): (i) if the agent has already won another task to go to the same shop; (ii) if the agent won the assemble task; or (iii) if the agent has not won any tasks yet. In the last two (ii and iii), the route length has to be lower than any other. In all three the agent needs to have enough load to carry any item

[5]https://github.com/smart-pucrs/mapc2017-pucrs/

[6]https://github.com/graphhopper/graphhopper

related to the task. Each time a task is awarded, the expected load of the winner is updated in the team's artefact, which is used in any further calculations.

The coordination of the job execution is done by $\mathcal{M}$OISE (Hübner et al., 2007). An agent can commit to two types of missions: *assemble* and *assist*. The assemble mission consists of two goals: to go to a workshop and to assemble the items. The assist mission has three goals: to buy required bases, to assist in assembling, and to stop assisting (the agent has to repeatedly execute *assist* action until the assembling is completed). Using a $\mathcal{M}$OISE scheme, the agents know exactly when they must send the actions to the contest server (assemble and assist actions, in that case).

4 COALITIONS OF VEHICLES FOR THE MAPC SCENARIO

In this section we describe our approach to enable experiments on CF in the 2017 MAPC scenario. We will describe the algorithms, the characteristic function we created for that scenario, and how we integrate all of it into MAS.

4.1 The CF Algorithms

We use two algorithms for CF created with different design purposes. The first one is an optimal coalition-formation algorithm that takes into account constraints (Rahwan et al., 2011). We use the implementation of the DC algorithm developed by T.Rahwan. The second algorithm we use for this work is the algorithm proposed by Farinelli et al., named C-Link (Bistaffa et al., 2017a), which was implemented from scratch. It consists of a heuristic algorithm for coalition formation based on hierarchical clustering algorithms. It does not search through the entire search space so it can run in reasonable time even for relatively large instances.

We choose to work with these two algorithms because they have distinct purposes. One searches for an optimal coalition structure which increases the confidence in the outcome. The second algorithm is based on a heuristic approach which delivers a solution quickly but it might not be the optimal one. We could handle our problem better by using coalition-formation algorithms for tasks, but to the best of our knowledge there are no (publicly available) implementations for such algorithms. As we already have an algorithm that outputs the optimal solution and another that runs quickly, we conducted the experiments using the algorithms described above.

4.2 Integration of CF into MAS

In order to start using CF into MAS, the agents must be aware of the algorithms and must know how to pass their private information to the algorithms and then get the results (through perception). For this purpose, they are able to use plans that perform operations on a CArtAgO artefact named CFArtefact[7], which also informs the agents about the outcome of the algorithms (in other words, the resulting coalition structure). The idea behind this artefact is to gather all the necessary information to partition the set of agents into disjoint coalitions regardless of the choice of CSG algorithm to be used in any particular application.

A key part of CF is the coalition value. It measures how productive the agents are when acting together. Almost all algorithms that form coalitions will require some information from the agents in order to partition the set of agents. Our approach is based on MCNets (Ieong and Shoham, 2005). We borrow the rule formalisation of their approach and use it for the agents to express their marginal contributions. From the format $pattern \rightarrow value$, where $pattern$ is a conjunction of positive and negative literals, we use the same idea to allow agents to establish rules such as: $mc_P \wedge mc_N \rightarrow value$, where $mc_P \subseteq A^{CF}$ and similarly $mc_N \subseteq A^{CF}$ (A^{CF} is the set of agents available on the CFArtefact). The basic idea is that mc_P represents the agents that must be on the coalition for it to receive the rule value and mc_N is the set of agents that must not be in the coalition for that to happen.

We argue that CFArtefact is a suitable tool for integrating coalition formation algorithms in any JaCaMo MAS (or more generally any artifact-based multi-agent development platform). The CFArtefact allows a number of features to be specified by the agents: (i) CCF constraints (positive, negative, and size); (ii) types of agents; (iii) agent skills; (iv) the tasks that must be performed; (v) and agents' contribution; the contribution value could be the agent's marginal contribution to coalitions or the synergy between agents. How this is done in practice will be made clearer later in the examples below.

4.3 The Characteristic Function

The algorithms we are experimenting with were not specifically designed to deal with tasks, they are general-purpose algorithms for coalition formation. In this sense, we have to insert the notion of tasks into these algorithms, because in our application scenario an agent's contribution depends on the particular task

[7]This source code is available at https://github.com/TabajaraKrausburg/CoalitionFormationForMAS.

in question. For that purpose, we use a set of "fake agents" to represent jobs. For instance, if we have ten free agents and a set of three jobs, then the set we want to partition will have thirteen elements, where each formed coalition will have the constraint to contain at most one job-representing agent. This way we will know for which job each coalition was formed.

We have to take into account some constraints while calculating a value for a coalition:

1. How many agents of a given type will be required for a coalition to accomplish a job (e.g., agents that can use a required tool).
2. At least one truck must be in each coalition.
3. Two fake agents representing jobs cannot belong to the same coalition. A coalition can only work on one job at time.

When the algorithm generates a coalition that does not pass on these constraints, we do not consider the marginal contribution of its individual members, but we only apply some punishments depending on which constraint was not satisfied.

More specifically, when the coalition satisfies the constraints, we calculate positive values for the characteristic function as follows. Our equation is based on the class of *m+a functions* which is the sum of a monotonic function and an anti-monotonic function (Bistaffa et al., 2017a). We calculate the positive side of adding agents to the coalition and the negative side of such addition. They are called *subadditive* and *superadditive* functions, respectivelly. Our *subadditive* function establishes a discount based on the coalition size[8]. In our domain, it is important to use as fewer agents as possible so that we have more free agents to accomplish more jobs. We use the following equation[9] to calculate such value:

$$subadd(C) = (|C|-1)^3 \qquad (1)$$

For the *superadditive* function, we apply to the coalition all the rules of marginal contribution, and for the rules that fit the positive and negative constraints, we sum up its value into the coalition value as given in Equation 2.

$$superadd(C) = \sum_{mc \in MC} value(C,mc) \qquad (2)$$

$$value(C,mc) = \begin{cases} mc_V, & \begin{array}{l}\text{if } |mc_P \setminus C| = 0 \wedge \\ (|mc_N| = 0) \vee \\ \quad |mc_N \setminus C| > 0)\end{array} \\ 0, & \text{otherwise} \end{cases}$$

[8]This idea is similar to the *coalition management cost* in (Bistaffa et al., 2017a).

[9]We have determined the power value used in that equation experimentally.

where

- MC represents the set of agents' marginal contributions to the coalitions;
- mc represents one marginal contribution rule;
- mc_V represents the value of a marginal contribution rule;
- $|mc_P \setminus C| = 0$ states that all the agents in the positive rule of the marginal contribution belong to coalition C;
- $|mc_N| = 0$ states that there is no agent that constrains the application of such rule; only the positive rule of the marginal contribution was set;
- $|mc_N \setminus C| > 0$ states that coalition C does not have all the agents in the negative part of the marginal contribution rule.

Now, we explain how the agents calculate their marginal contribution. Each agent receives from the initiator a set of jobs and inserts in the CFArtefact a contribution value to each of them (i.e., it inserts a marginal contribution conditioned on it being on the same coalition as a fake agent representing a job). For this calculation we take into account the agent's distance to the *shops*, *workshops*, and *storage* relevant to that job. The normal course of an agent's action is to go buy the desired items, then it goes to a workshop to assemble the compound items, and finally goes to the storage to deliver the job. Once we have defined which facilities are in the agent's route, we calculate how many steps are necessary to complete the route. We subtract the agent's route length from 100 to get higher values to smaller routes[10].

5 EXPERIMENT RESULTS AND ANALYSIS

We now introduce our experiments to evaluate the use of coalition formation in the 2017 MAPC. It is a complex and stochastic environment and we aim to evaluate how much coalition formation can improve the performance of a team of agents.

5.1 Preliminaries

We mainly experiment with three distinct approaches, and for the sake of clarity we use short names for each of them:

- **O-CF:** it runs the DC algorithm to split the set of agents before the task allocation takes place;

[10]We empirically evaluated that an agent's route never takes more than 100 steps.

- **H-CF:** it runs the C-Link algorithm to split the set of agents before the task allocation takes place;
- **TA-3:** it does not run any CF algorithm previously; the task allocation process considers the set of all agents in the team;

All approaches make use of the task allocation technique implemented by the SMART-JaCaMo team. We remind the reader that our experimentation is built on top of the SMART-JaCaMo's code, so we obey their implementation constraints. We always wait until three jobs have been announced to start the whole process. The task allocation algorithm requires at least one truck to perform a job and we have three trucks in our set of agents; it thus makes no sense to work with sets greater than three jobs. Agents that belong to a coalition which is about to start the task allocation process do not have any guarantees that they will be assigned a task for that job; it is up to the task allocation algorithm.

5.2 Experiments on a Set of Jobs

In this experiment we are particularly interested in the *job rate configuration*. With this value, we can experiment with the frequency of jobs being announced by the contest server. We specifically use the values of 10%, 25%, 50%, 75% and 100% of the maximum capacity. In our experiments, the maximum capacity is the maximum frequency in which we would like to have a job announced. Apart from the maximum capacity, the job probability to announce a job also takes into account the remaining steps of the simulation; as we show in the following equation:

$$P(J) = e^{(-1\times(currentStep/totalSteps))} \times rate \qquad (3)$$

- *currentStep* represents the current step of the simulation;
- *totalSteps* represents the total number of steps defined in the server configuration file for the simulation to have;
- *rate* represents the job rate configured in the server configuration file;

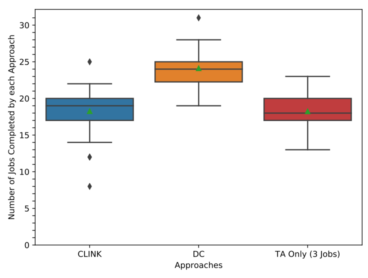
At the beginning of the simulation we have the highest chance of a job being announced by the server; as the number of the current simulation steps progresses, we have lower chances to get a new job. For each approach, and varying the job rate, we executed 50 repetitions of the simulation.

When the job rate is set to 10% of the maximum capacity, O-CF has better performance than others approaches (Figure 2a). This is due to the search over the entire search space (after the constraints being applied). It can evaluate the combinations of all feasible coalitions and detect the best coalition structure. In other words, it can evaluate each set of agents to each job in order to find out the most suitable agents to a particular job. This does not happen with H-CF because the C-Link algorithm tends to join the best pair of agents based on the linkage function. Once these agents are put together they will never be split again. In such cases, if we have two jobs that require one drone each and we only have two available drones, H-CF can put these two drones in the same coalition which makes one of the jobs unfeasible.

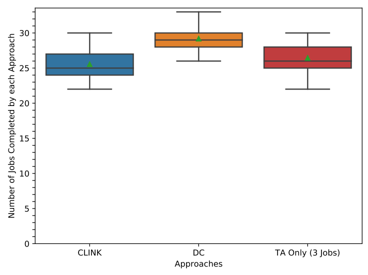
Regarding TA-3 with 10% of the maximum capacity, the SMART-JaCaMo gets the first job from the set and starts the allocation of its tasks. Then, it picks the second job and only when that task allocation is over it works on the the third job. From the set of jobs, agents can be allocated to one, two, or three jobs depending on the agents' location on the map when the jobs were announced. In that approach, if two agents are the best option for the first job (think about the case of drones which are the fastest agents), they will be allocated to that job, but one of them might be necessary for the second job, making it unfeasible. Considering the three approaches, optimal coalition formation can balance the agents to the jobs, and then allocate more jobs as the agents will have shorter idle periods.

As we start increasing the job rate, the number of completed jobs for O-CF remains in first place, but we can notice the difference between it and TA-3 decreasing (see Figure 2b). When we move to 50% of maximum job rate, we can see that TA-3 is leading (Figure 2c). This is due to the interval between one job and another. When the number of jobs sent by the contest server is high, as it can only announce at most one job per step, the interval between jobs decreases. Even though TA-3 cannot allocate all the jobs of the set to the agents, in few steps new jobs will be sent by the server and a new execution of the allocation can be run then. In the long run, the frequency of new jobs arriving compensates the failures in the allocation process. For this reason, TA-3 has the best performance, as it allocates the best agents to accomplish a job, in other words, they have the smallest time-step to delivery the job.

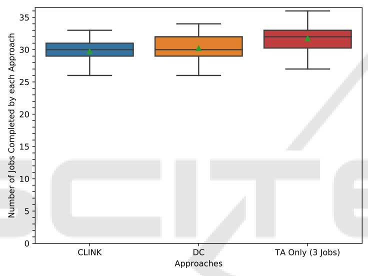
For job rates of 75% and 100%, TA-3 remains leading for the same reasons. Furthermore, another interesting fact can be observed in these experiments. The H-CF's performance is approximately equal to O-CF's performance. This is also due to the job frequency. Again, as H-CF lets some jobs and agents behind, with higher job rates this loss is compensated. The available agents that could not be allocated in the previous set of jobs may be allocated in the next set



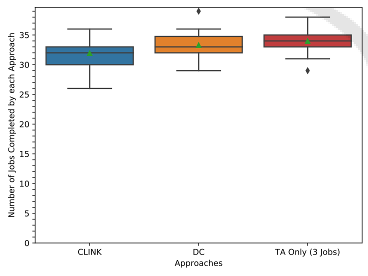
(a) Job rate at 10% of maximum capacity.



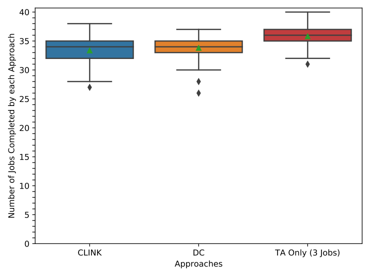
(b) Job rate at 25% of maximum capacity.



(c) Job rate at 50% of maximum capacity.



(d) Job rate at 75% of maximum capacity.



(e) Job rate at 100% of maximum capacity.

Figure 2: Number of completed jobs after 1000 steps considering different job rates. We depict the average of completed jobs for 50 simulations.

of jobs. In this sense, the team has fewer steps with idle agents, which result in more jobs being delivered. As H-CF also outputs only coalitions to perform the job, which means it does not decide which agent will perform each part of the work when forming the coalitions, it also stays behind TA-3.

6 RELATED WORK

Coalition formation has been applied to various different areas, and in many of them it aims to reduce operational costs. We emphasize the work developed for ride-sharing problem (Bistaffa et al., 2017b) and collective electricity consumption shifting (Akasiadis and Chalkiadakis, 2017). Our work differs from all those mentioned above in experimenting with an already implemented system to evaluate how CF approach can improve the overall performance. We analyse under what circumstances forming coalitions may benefit a team of agents. The aim of the various projects reported here is to assess the performance of coalition formation for particular domains. They do not evaluate if the coalition formation approach is the best approach for that scenario. They assume it is and experiment on the parameters and algorithms to solve efficiently the problem. Most of the related work focuses on coalition structure formation and in the payment calculation after achieving the coalition goal.

Apart from the traditional problem of coalition formation (splitting the set of agent into disjoint and exhaustive coalitions), much work reported in the literature proposes to form one coalition to perform one task (Irfan and Farooq, 2016; Ayari et al., 2017). In those approaches the coalitions are implicitly formed as a consequence of the task allocation process, whilst they differ on how a coalition is formed. As can be noted, such approaches output a team of agents to achieve a task[11]. They do not address the coalition formation problem, in which we must partition a set of agents into disjoint coalitions. In the case of our experiments, all those approaches would not be efficient on low job rates. However, there are approaches that aim to find the best coalition structure to accomplish a set of tasks, for example (Rauniyar and Muhuri, 2016). Their work is the closest to ours because it aims to balance the agents between the set of task that must be accomplished. Our aim is to integrate coalition formation techniques into systems that have already been developed. For that purpose, we maintain the task allocation algorithm implemented by the SMART-JaCaMo team, and we analyse what

[11]Recall that in a *team*, the agents are aware of the tasks they will be responsible for.

are the benefits of integrating coalition formation on an already implemented MAS.

7 CONCLUSIONS

We integrated coalition formation techniques into the 2017 MAPC (Ahlbrecht et al., 2018). In this domain, we have heterogeneous agents that work together in order to deliver jobs announced over time. We started on top of the SMART-JaCaMo team's implementation (Cardoso et al., 2018) for the contest. We experimented with two different algorithms for CSG; one provides an optimal solution while the other is a heuristic algorithm. We integrated such algorithms for use in the JaCaMo platform using a CArtAgO artefact named CFArtefact. It is a suitable tool to use CF algorithms in any JaCaMo MAS.

Our experiments showed that CF is important for low job rates; however, when we increase job rate, in that particular application domain, the effectiveness of coalition formation is worse than the original approach. It cannot beat the approach that uses only task allocation because its purpose is not to decide which agent will accomplish each task at the time the coalitions are formed; it only specifies who will work together, and with many jobs being announced, that is not too important (in that specific domain). As a future work, we aim to investigate how self-interested behaviour impacts on a team's performance for the MAPC. Also, in order to show the generality of our approach, we aim to apply CF techniques in other domains using our CFArtefact, for instance disaster response, in which robots and humans will form coalitions to work together in damaged areas.

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

REFERENCES

Ahlbrecht, T., Dix, J., and Fiekas, N. (2018). Multi-agent programming contest 2017. *Ann. Math. Artif. Intell.*, 84(1):1–16.

Akasiadis, C. and Chalkiadakis, G. (2017). Cooperative electricity consumption shifting. *SEGN*, 9:38–58.

Ayari, E., Hadouaj, S., and Ghedira, K. (2017). A dynamic decentralised coalition formation approach for task allocation under tasks priority constraints. In *Proc. of 18th ICAR*, pages 250–255.

Behrens, T., Hindriks, K. V., Bordini, R. H., Braubach, L., Dastani, M., Dix, J., Hübner, J. F., and Pokahr, A. (2012). *An Interface for Agent-Environment Interaction*, chapter 8, pages 139–158. Springer.

Bistaffa, F., Farinelli, A., Cerquides, J., Rodríguez-Aguilar, J., and Ramchurn, S. D. (2017a). Algorithms for graph-constrained coalition formation in the real world. *ACM TIST*, 8:1–24.

Bistaffa, F., Farinelli, A., Chalkiadakis, G., and Ramchurn, S. D. (2017b). A cooperative game-theoretic approach to the social ridesharing problem. *Art. Intell.*, 246:86–117.

Boissier, O., Hübner, J. F., and Ricci, A. (2016). *The JaCaMo Framework*, chapter 7, pages 125–151. Springer.

Cardoso, R. C., Krausburg, T., Baségio, T., Engelmann, D. C., Hübner, J. F., and Bordini, R. H. (2018). SMART-JaCaMo: an organization-based team for the multi-agent programming contest. *Ann. Math. Artif. Intell.*, 84(1):75–93.

Chalkiadakis, G., Elkind, E., and Wooldridge, M. J. (2011). *Computational Aspects of Cooperative Game Theory*. Online access: IEEE (Institute of Electrical and Electronics Engineers) IEEE Morgan & Claypool Synthesis eBooks Library. Morgan & Claypool Publishers.

Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *KER*, 19:281–316.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multi-agent systems using the Moise+ model: programming issues at the system and agent levels. *IJAOSE*, 1:370–395.

Ieong, S. and Shoham, Y. (2005). Marginal contribution nets: A compact representation scheme for coalitional games. In *Proc. of 6th CEC*, pages 193–202.

Irfan, M. and Farooq, A. (2016). Auction-based task allocation scheme for dynamic coalition formations in limited robotic swarms with heterogeneous capabilities. In *Proc. of 7th ICISE*, pages 210–215.

Rahwan, T., Michalak, T. P., Elkind, E., Faliszewski, P., Sroka, J., Wooldridge, M., and Jennings, N. R. (2011). Constrained coalition formation. In *Proc. of 25th ICAI*, pages 719–725.

Rahwan, T., Michalak, T. P., Wooldridge, M., and Jennings, N. R. (2015). Coalition structure generation: a survey. *Art. Intell.*, 229:139–174.

Rauniyar, A. and Muhuri, P. K. (2016). Multi-robot coalition formation problem: task allocation with adaptive immigrants based genetic algorithms. In *Proc. of 9th IEEE ICSMC*, pages 137–142.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Art. Intell.*, 111:209–238.

Smith, R. G. (1980). The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE TC*, 29:1104–1113.

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing.