An Agent-Based Planning Method for Distributed Task Allocation

Dhouha Ben Noureddine[1,2(✉)], Atef Gharbi[1], and Samir Ben Ahmed[2]

[1] LISI, National Institute of Applied Science and Technology, INSAT, University of Carthage, Tunis, Tunisia
dhouha.bennoureddine@gmail.com, atef.elgharbi@gmail.com
[2] FST, University of El Manar, Tunis, Tunisia
samir.benahmed@fst.utm.tn

Abstract. In multi-agent systems, agents should socially cooperate with their neighboring agents in order to solve task allocation problem in open and dynamic network environments. This paper proposes an agent-based architecture to handle different tasks; in particular, we focus on planning and distributed task allocation. In the proposed approach, each agent uses the fuzzy logic technique to select the alternative plans. We also propose an efficient task allocation algorithm that takes into consideration agent architectures and allows neighboring agents to help to perform a task as well as the indirectly related agents in the system. We illustrate our line of thought with a Benchmark Production System used as a running example in order to explain better our contribution. A set of experiments was conducted to demonstrate the efficiency of our planning approach and the performance of our distributed task allocation method.

Keywords: Multi-agent system · Software architecture · Distributed task allocation · Planning · Fuzzy logic

1 Introduction

Nowadays, task allocation in Multi-Agent System (MAS) is a noteworthy research issue. Task allocation problem can be defined as that when an agent has a task which it cannot attain independently, the agent then tries to find other agents which contain the proper resources, and assigns the task or part of the task, to those agents. That's why, they need to be cooperative with their neighboring agents to process tasks and accomplish their objectives. The social cooperation is a crucial challenge in the software engineering fields, especially in the distributed artificial intelligence and MAS [5]. This challenge developed with the progress of the applications, e.g. in wireless ad-hoc networks [6], service-oriented MAS [7], multi-robot system in healthcare facilities [8], file sharing in P2P systems [9], social networks [10], etc. So, cooperation can provide appreciable convenience for these applications by promoting joint goals.

M. van Sinderen and L. A. Maciaszek (Eds.): ICSOFT 2018, CCIS 1077, pp. 282–306, 2019.
https://doi.org/10.1007/978-3-030-29157-0_13

More and more attention has been paid to distributed task allocation approaches. Early researches used centralized approaches (such in [17]) to generate a plan for cooperating all the agents by using a central server able to gather the whole system information. This type of approaches can be a proficient solution in a small network because the central planner has a global view of the whole system and it affects the appropriate tasks to the agents. In such case, communication overhead could be decreased during allocation processes. On the flip side, it also has important disadvantages. First, in some systems, it is hard to have such a central controller. Second, when the central planner is out of order or cracked by some attackers task allocation will endure a major inconvenience in this system.

Other researches pointed out the distributed task allocation approaches (e.g. [15]) as a solution to avoid the risks of deficiencies of the centralized approaches. The distributed task allocation approaches are widely used for interactive MAS, semantic web and grid technologies. However, the decentralized approaches are more scalable and robust but the communication overhead rises.

In this paper, we propose an agent-based architecture to manage tasks and control embedded systems at run-time. We firstly, introduce multi-agent planning in which each agent uses the fuzzy logic technique to select plans. The originality in this approach is that our agents evaluate plans based on their goal achievement satisfaction, which is represented as degrees of membership for each individual agent, their aggregate then represents the satisfaction of the overall goal. Proving that our approach performs better than the central planning processes in other systems. We then propose the distributed task allocation solution which is allowing agents to request help from neighbors, this would be done by allocating tasks to different agents who may be able each, to perform different subsets of those tasks. We use to highlight the performance of our solution using the provision of a benchmarking scenario.

The remainder of this paper is organized as follows. Section 2 provides some current related research in this field. Section 3 introduces the benchmark production system used in our approach. After that, in Sect. 4 a software architecture of MAS will be depicted in detail. Section 5 defines our planning method and demonstrates the simulation and analysis about the quality and performance of our method. Then, a distributed task allocation approach is illustrated as well as its related experiments in Sect. 6. Finally, we discuss and conclude our work in Sect. 7.

2 Related Work

In the literature, the proposed architectures solving the task allocation can be classified into centralized or decentralized. [17] proposed a centralized approach, therefore, they supposed that there is a central planner to allocate tasks to agents. Their main goal was to find a solution with a small team cost and each objective to be allocated to the correct number of various agents. Despite the centralized approaches have the main advantage of computing a global plan

dependent on all accessible data, their main drawback, however, is the fact that being a single point of failure.

The decentralized architecture dodges this issue, there is no centralized controller and rather the task allocation process was contributed by all agents. The distributed task allocation approach additionally has the benefit of scalability and robustness. In the multi-agent network, the task allocation remains a complex problem. Many parameters have to be taken into accounts, such as communication protocols, resource sharing, synchronization or the evolution of the priorities assigned to each task, etc. These different parameters are positioned as strong constraints when we consider that they evolve as and when the missions unfold. This raises the question of the effectiveness of a planning or the relevance of a dynamic allocation solution without prior planning. We will send the interested reader back to [15] and [16] for a categorization around different axes such as self-organization, the formalization of coordination and the composition of teams of agents.

Some researchers [18–20] proposed other task allocation approaches in multi-agent network environments including the negotiation-based approaches. [20] introduced a method with uncertain negotiation deadlines. Thereafter, [22] proposed an approach based on negotiation for task allocation by taking into account the uncertain factors such as the deadline and the reserve price. Nevertheless, the negotiation for task allocation in most open, dynamic and distributed environments, practically, takes into consideration more than two uncertain factors. In [23], the authors expanded more uncertainty factors, like resource competition, deadline, reserve price and cost under the assumption of a global view of each resource consumer.

In [21], the authors developed a market-based approach for allocation tasks in the environment that is in reality an approach based on multi-resource negotiation. In their method, the consumer gets the required resources through negotiating with providers for each of the needed resources independently. Against our approach, the separate negotiations always result in a large number of Manager being selected to finish a task, and this may result in communication overload among the chosen managers.

[14] proposed a Greedy Distributed Task allocation Protocol (GDAP) in social networks. There are a few angles at which GDAP is like our approach, e.g., there is no central controller in GDAP, which implies every agent just has local view and agents are connected as a social network which is similar to the one proposed in our approach. However, this protocol just enables neighboring agents to help with a task which may result in a high probability of abandon of tasks when neighbors can't provide sufficient and adequate resources. In this paper, our approach [4] is proposed which allows agents to allocate tasks not only to their neighbors yet, in addition, to submit incomplete tasks to their neighbors for reallocation. Along these lines, the agents can have more opportunities to accomplish a solution to their tasks. Given the characteristics of existing multi-agent task allocation approaches, there remains an important opportunity to develop cost-effective and communication economical decentralized methods to

task allocation in the multi-agent systems. Although additional assumptions like partial observability, and heterogeneity, can additional make difficult this problem. In this paper, we propose a decentralized planning algorithm [4] to the following hypothesis: (i) there are no environmental uncertainties, and (ii) each agent has a full observation of all tasks and the state of other agents.

A combinatorial auction-based algorithm CBBA [24] proposed to solve task allocation problem. This algorithm used combinatorial auctions, where groups of tasks are produced. CBBA has displayed better execution than single-item auctions and has created good results against optimal centralized approaches [25]. A second combinatorial auction-based algorithm was developed by [27] similar to CBBA. However, the baseline of this algorithm performs, empirically, better than the baseline CBBA algorithm [26], with the approach of [27] showing a greatly improved achievement rate with different numbers of tasks and agents, and different network topologies. However, the papers mentioned do not examine handling of uncertainty of the method of [27], as well as CBBA's.

3 Benchmark Production System

We explain our approach using a simple current example called RARM [11] which is implemented in our previous work [1,2,4]. The RARM presented in Fig. 1 is composed of two inputs and one output conveyors, a servicing robotic agent and a processing-assembling center. Workpieces to be treated come irregularly one by one. The workpieces of type A are delivered via conveyor $C1$ and workpieces of type B via conveyor $C2$. Only one workpiece can be on the input conveyor. A robotic agent R transfers workpieces one after the other to the processing center. The next workpiece can be put on the input conveyor when it has been emptied by the robotic agent. The technology of production requires that firstly an A-workpiece is inserted into the center M and treated, then a B-workpiece is added to the center, and finally the two workpieces are assembled. Afterwards, the assembled product is taken by the robot and put above the $C3$ conveyor of output. The assembled product can be transferred on $C3$ only when the output conveyor is empty and ready to receive the next produced one.

3.1 Sensing Input

Formally, the statement of benchmark production system is defined like this: $RARM = \{$*position, A-workpiece, B-workpiece, AB-workpiece, conveyor, states, processing center, robotic agent*$\}$ where each variable is defined by his values as follows:

- A set of positions $\{p_1, p_2, \dots\}$: the variable **position** is used to localize the workpiece A, B or AB and p_1, $p_2, ..., p_i$ present the values of the variable **position**;
- A set of robotic agents $\{r_1, r_2, \dots\}$: the variable **robotic agent** transfers a workpiece one after one to be processed and r_1, $r_2, \dots r_i$ present the values of the variable **robotic agent**;

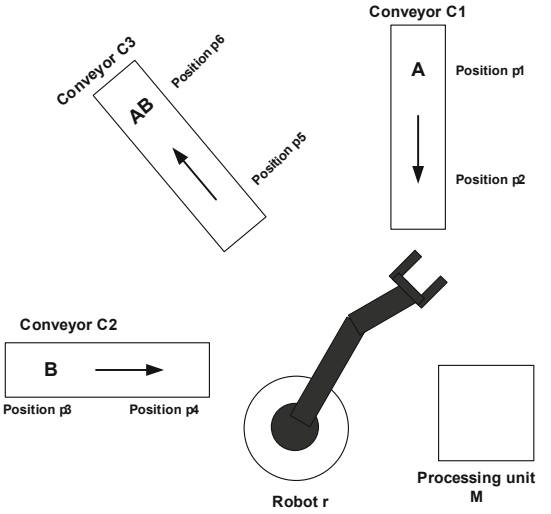

Fig. 1. The benchmark production system RARM.

- A set of workpieces of type A $\{a_1, a_2, \dots\}$: $a_1, a_2, ..., a_i$ present the values of the variable **A-workpiece**;
- A set of workpieces of type B $\{b_1, b_2, \dots\}$: $b_1, b_2, ..., b_i$ present the values of the variable **B-workpiece**;
- A set of workpieces of type AB $\{ab_1, ab_2, \dots\}$: $ab_1, ab_2, ..., ab_i$ present the values of the variable **AB-workpiece**;
- A set of conveyors $\{C_{1i}, C_{2i}, C_{3i}\}$: the variable **conveyor** and his values C_{1i} (resp. C_{2i}, C_{3i}) is responsible for transferring set of workpieces of type A (resp B, AB);
- A set of processing center M $\{M_1, M_2, \dots\}$: first one A-workpiece is inserted into the variable **processing center** M and processed, then one B-workpiece is added into the center M, and last both workpieces are assembled.

The set of the variable **states** is $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}\}$ where:

- s_0 (resp. s_6, s_{15}) is meaning a workpiece of type A (resp. B, AB) is at position $p1$ (resp. $p3$, $p5$);
- s_1 (resp. s_7, s_{16}) is meaning a workpiece of type A (resp. B, AB) is at position $p2$ (resp. $p4$, $p6$);
- s_2 (resp. s_8, s_{14}) is meaning a workpiece of type A (resp. B, AB) is taken by the robotic agent r near to the position $p2$ (resp. $p4$, $p5$) of the conveyor $C1$ (resp. $C2$, $C3$);
- s_3 (resp. s_9, s_{13}) is meaning a workpiece of type A (resp. B, AB) is taken by the robotic agent r near to the processing unit;
- s_4 (resp. s_{10}, s_{12}) is meaning a workpiece of type A (resp. B, AB) is put in the processing unit M.

The robot-like agent connects directly to the environment via sensors. So, these sensing inputs present the observations of RARM in our approach. It is defined as follows:

1. Is there an A-workpiece at the extreme end of the position $p1$? (sens1)
2. Is $C1$ in its extreme left position? (sens2)
3. Is $C1$ in its extreme right position? (sens3)
4. Is there an A-workpiece at the unit M? (sens4)
5. Is $C2$ in its extreme left position? (sens5)
6. Is $C2$ in its extreme right position? (sens6)
7. Is there a B-workpiece at the extreme end of the position $p3$? (sens7)
8. Is there a B-workpiece at the unit M? (sens8)
9. Is $C3$ in its extreme left position? (sens9)
10. Is $C3$ in its extreme right position? (sens10)
11. Is there a AB-workpiece at the unit M? (sens11)
12. Is the robotic agent arm in its lower position? (sens12)
13. Is the robotic agent arm in its higher position? (sens13)

3.2 Action Output

The system can be controlled using the following actuators:

1. move the conveyor $C1$ (act1);
2. move the conveyor $C2$ (act2);
3. move the conveyor $C3$ (act3);
4. rotate robotic agent (act4);
5. move elevating the robotic agent arm vertically (act5);
6. pick up and drop a piece with the robotic agent arm (act6);
7. treat the workpiece (act7);
8. assembly two pieces (act8).

The set of actions is $\{C1\_left,\ C1\_right,\ R1\_left,\ R1\_right,\ C2\_left,\ C2\_right,\ R2\_left,\ R2\_right,\ C3\_left,\ C3\_right,\ R3\_left,\ R3\_right,\ take_1,\ take_2,\ take_3,\ load_1,\ load_2,\ load_3,\ put_1,\ put_2,\ put_3,\ process_1,\ process_2\}$ where:

- $C1\_left$ (resp. $C1\_right$) is meaning a workpiece of type A is moving to the left of conveyor $C1$ from position p_1 (resp. p_2) to position p_2 (resp. p_1);
- $C2\_left$ (resp. $C2\_right$) is meaning a workpiece of type B is moving to the left of conveyor $C2$ from position p_3 (resp. p_4) to position p_4 (resp. p_3);
- $C3\_left$ (resp. $C3\_right$) is meaning a workpiece of type AB is moving to the left of conveyor $C3$ from position p_5 (resp. p_6) to position p_6 (resp. p_5);
- $R1\_left$ (resp. $R1\_right$) is meaning the robotic agent taking a workpiece of type A is moving to the left (resp. to the right) from the position p_2 of conveyor $C1$ (resp. the processing unit M) to the processing unit M (resp. the position p_2 of conveyor $C1$);

- $R2\_left$ (resp. $R2\_right$) is meaning the robotic agent taking a workpiece of type B is moving to the left (resp. to the right) from the position p_4 of conveyor $C2$ (resp. the processing unit M) to the processing unit M (resp. the position p_4 of conveyor $C2$);
- $R3\_left$ (resp. $R3\_right$) is meaning the robotic agent taking a workpiece of type AB is moving to the left (resp. to the right) from the the processing unit M (resp. position p_2 of conveyor $C3$) to the position p_2 of conveyor $C3$) (resp. the processing unit M);
- $take_1$ (resp. $take_2$, $take_3$) is meaning the operation of taking a workpiece of type A (resp. B, AB);
- $load_1$ (resp. $load_2$,$load_3$) is meaning the fact of loading a workpiece of type A (resp. B, AB);
- put_1 (resp. put_2, put_3) is meaning the operation of putting a workpiece of type A (resp. B, AB);
- $process_1$ (resp. $process_2$) is meaning the fact of processing a workpiece of type A (resp. B).

4 Agent Architecture

We propose an agent architecture to control embedded systems at runtime. The agent checks the evolution of the environment and reacts when new events occur.

4.1 Formal Specification

We use the state machine to define the dynamic behavior of an intelligent agent controlling the planning. In the state machine, states, inputs and outputs are enumerated. The state machine can be defined as a graph of states and transitions. It treats many events that may execute by detecting them and responding to each one appropriately. We describe a state machine SM_i as follows:

$$SM_i = (S_i, S_{i0}, I_i, O_i, Precond_i, Postcond_i, t_i)$$

- $S_i = \{s_{i1}, .., s_{ip}\}$: the set of states;
- S_{i0} the initial state;
- $I_i = \{I_{i1}, .., I_{im}\}$: the input events;
- $O_i = \{O_{i1}, .., O_{ik}\}$: the output events;
- $Precond_i$: the set of conditions to be verified before the activation of a state;
- $Postcond_i$: the set of conditions to be verified once a state is activated;
- $t_i : S_i \times I_i \rightarrow S_i$: the transition function.

Figure 2 shows a conceptual model for a state machine where we define the classes *State machine*, *State*, *Transition*, *Event* and *Condition*. The class *State Machine* composed by the classes *State* and *Transition*. The class *Transition* is doubly associated linked to the class *State* as long as the transition is considered as an association between two states. Each transition has an event that is considered as a trigger to fire it and a set of conditions to be verified. This association between the class *Transition* and the two classes *Event* and *Condition* exists and it's modeled by the aggregation relation.

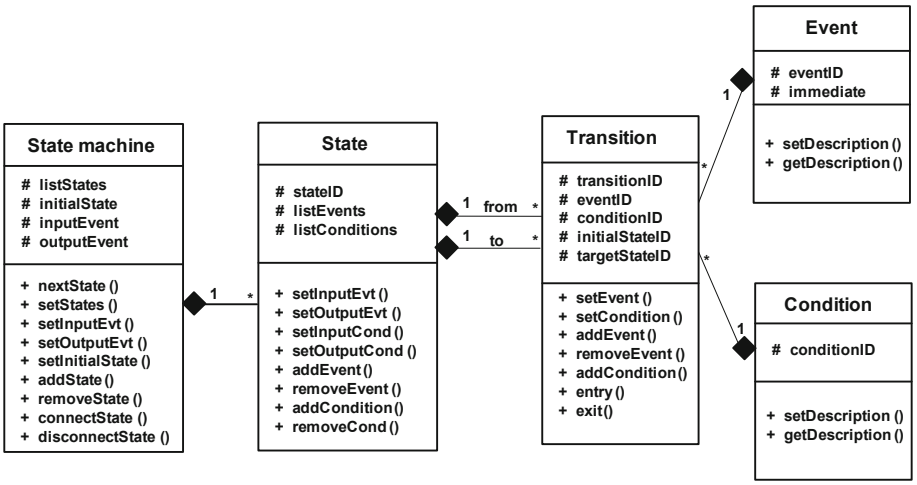

Fig. 2. The state machine model.

4.2 Conceptual Architecture for MAS

A conceptual architecture was proposed for MAS. It consists of four parts: *(i)* the Event Queue which saves different input events that may happen in the system, *(ii)* the Software Agent which reads an input from the event queue and reacts accordingly, *(iii)* the set of state machines, and *(iv)* each state represents a specific information about the system.

The agent using state machine determines the new state of the system to perform depending on the event inputs and the conditions to be satisfied. This approach provides coming characteristics: *(i)* The design of the agent is general enough to adapt to different sorts of embedded-software based application. Consequently, the agent is uncoupled from the application and from its components. *(ii)* The agent is independent of the state machine: it grants to change the structure of the state machine without changing the implementation of the agent. This guarantees the agent keeps on working accurately even if there should arise an occurrence of the modification of state machines.

In the following algorithm, the symbol Q is an event queue which holds incoming event instances, ev refers to an event input, S_i represents a State Machine, and $s_{i,j}$ a state related to a State Machine S_i. The internal behavior of the agent is defined as follow:

1. the agent reads the first event ev from the queue Q;
2. searches from the top to the bottom in the different state machines;
3. within the state machine SM_i, the agent verifies if ev is considered as an event input to the current state $s_{i,j}$ (i.e. ev $\in$ I related to $s_{i,j}$). In this case, the agent searches the states considered as successor for the state $s_{i,j}$ (states in the same state machine SM_i or in another state machine SM_l);
4. the agent executes the operations related to the different states;
5. repeats the same steps (1–4) until no more event exists in the queue to be treated.

First, the agent evaluates the pre-condition of the state $s_{i,j}$. If it is false, then the agent exits, else the agent determines the list of tasks to be executed. Then, it evaluates the post-condition of the state $s_{i,j}$ and generates errors whenever it is false.

5 Multi-agent Planning

We are interested in the use of reflection capabilities and multi-agent interaction protocols for the task allocation. If we place ourselves in an open and loosely coupled framework, it is difficult for a set of agents to combine their actions to reach a goal that it be explicitly described at the system level or implicitly from the local goals of agents.

We will propose an approach to deal with this problem. It suggests a decentralized algorithm for task allocation in complex MAS based on multi-agent planning in which each agent uses the fuzzy logic technique to select plans. The originality in this approach is that our agents evaluate plans based on their goal achievement satisfaction, which is represented as membership degrees for each individual agent, their aggregate then represents the satisfaction of the overall aim. This approach successes to make the collaboration and communication between agents superb and it performs better than the central planning processes in other systems. As it's known the task allocation problem in teams is one of optimally assigning tasks in a team plan to agents to maximize overall team utility.

In this approach, our agents use their introspection and intercession abilities to indicate in runtime what they can do, what they cannot do and why? And change their actions so that they are adaptable.

5.1 Policy

The most approaches concerning planning use a single-agent procedure, in which one agent executes the entire search process, developing the complete action plan to do the task at hand. We are inspired by these approaches to propose our contribution, in which every agent belonging to our system can make a decision by following a simple policy. This policy aids the agent to select a suitable plan of actions to carry out, which led us to solve many issues regarding the openness of the MASs and more precisely the task allocation problems.

We use conjunction operators for *fuzzy relationship* in [2,4], to make the agent expresses more objectively the decision about the evaluation of a plan (list of events). It is very feasible to apply a membership function in fuzzy mathematics to calculate and assess the *satisfaction degree* of the plan. So, we consider G the problem goal, it is the union of individual goals of all agents denoted by g_i, which are flexible propositions.

$$G = \cup_{i=1..n} g_i \tag{1}$$

These goals can be accomplished with a satisfaction degree. The form of a flexible proposition is $(\rho\ \phi_1, \phi_2, ..., \phi_j\ \kappa_i)$, where $\phi_i \in \phi$ and κ_i are elements of totally ordered set, K, which presents the truth degree of the proposition. K is composed of a finite number of membership degrees, $k_\uparrow, k_1, ..., k_\downarrow$, where $k_\uparrow \in K$ and $k_\downarrow \in K$, representing respectively total falsehood and total truth. When dealing with a flexible proposition with a truth value of $k_\uparrow$ or $k_\downarrow$, the boolean style $\neg\ (\rho\ \phi_1, \phi_2, ..., \phi_j)$ or $(\rho\ \phi_1, \phi_2, ..., \phi_j)$ is adopted. The flexible proposition [12] is described by a *fuzzy relation*, R, which is defined as a membership function $\mu_R(.)$: $\Phi_1{*}\Phi_2{*}...{*}\Phi_j \rightarrow K$, where $\Phi_1{*}\Phi_2{*}...{*}\Phi_j$ is the Cartesian product of the subsets of Φ in the current proposition state. Especially, if each agent accomplishes its individual objectives with a specific satisfaction degree, the public goals of the problem are achieved. The satisfaction degree of a multi-agent flexible planning problem is characterized as the conjunction of the satisfaction degrees of each action and goal.

$$\mu_G = \wedge_{i=1..n}\mu_R(i) \tag{2}$$

The function μ_G [4] signifies how well a given plan is satisfying and can be considered as a value between 0 and 1, 1 represents totally satisfied and 0 represents not satisfied by any means. In our approach, each plan alternative is related to a satisfaction degree. That implies each value is the metric that gives the way to choose a plan among various alternatives. Having the improved mean values calculated, the alternative plan alongside these values are sent to the current state machine. The alternative plan, the need, the objective, and the corresponding values reach the decision-making mechanism first. These values are used by the decision-making mechanism to compare the satisfaction degrees for each alternative plan to find the most satisfactory and acceptable one. The one with the highest satisfaction degree is considered as the most satisfactory plan alternative.

Running Example

Giving $(S,\ A,\ G_s)$ where $S = \{s_i|\mathrm{i}=1\ldots\mathrm{n}\}$ is a set of states, $A=\{Ci\_left,\ Ci\_right,\ Ri\_left,\ Ri\_right,\ take_i,\ load_i,\ put_i,\ process_i|\mathrm{i}=1\ldots\mathrm{n}\}$ is a set of actions, and G_s is the problem goal. *if* s_0 *and* $g = \{$*workpiece in the processing unit*$\}$. *Let:*

- π_0: $(C2\_left,\ take_2,\ load_2,\ process_2)$
- π_1: $(load_1,\ put_1,\ process_1,\ C1\_right)$
- π_2: $(C1\_left,\ take_1,\ load_1,\ put_1,\ process_1,\ C1\_right)$

We solve multi-agent planning problems by distributed flexible constraint satisfaction problem (CSP) technique [13] and make a trade-off between plan length and the compromise decisions made. The quality of a plan is measured by its satisfaction degree and its length, where the shorter of two plans is better under the same satisfaction degrees. In this example, the definitions of K and L are: $\mathrm{K}=\{k_\uparrow, k_1, k_2, k_\downarrow\}$, $\mathrm{L}=\{l_\uparrow, l_1, l_2, l_\downarrow\}$.

The multi-agent planning problem is helpful to robot-like agents like in this example. If any actions $\in \{Ci\_left,\ Ci\_right,\ Ri\_left,\ Ri\_right,\ take_i,\ load_i,$

put_i, $process_i|\mathrm{i}=2 \ldots \mathrm{n}\}$ will damage the plan, leading to a satisfaction degree l_2, any plan not beginning with $C1\_left$ will result in a satisfaction degree l_2 because it is not applicable to s_0, and when any action is applicable to s_0 and the resulting state is a goal state then the result will be a satisfaction l_1. We obtain π_0 which is a plan of 4 steps with satisfaction l_2 as follows:

- $C2\_left$ has a satisfaction degree l_2
- $take_2$ has a satisfaction degree l_2
- $load_2$ has a satisfaction degree l_2
- $process_2$ has a satisfaction degree l_2

In addition, we obtain π_2 which is a plan of 6 steps with satisfaction $l_\uparrow$ as follows:

- $C1\_left$ has a satisfaction degree $l_\uparrow$
- $take_1$ has a satisfaction degree $l_\uparrow$
- $load_1$ has a satisfaction degree $l_\uparrow$
- put_1 has a satisfaction degree $l_\uparrow$
- $process_1$ has a satisfaction degree $l_\uparrow$
- $C1\_right$ has a satisfaction degree $l_\uparrow$

Then π_0 is not a solution because although it is applicable to s_0, the resulting state is not a goal state; π_1 is not a solution because it's not applicable to s_0; π_2 is the most appropriate solution.

5.2 Experimental Evaluation

We have evaluated our approach to prove, on the one hand, how our software agent having distinctive capacities such as it can perform, simulate the behavior of the agent. On the other hand, to prove how the performance of the robotic agents was impacted by varying their satisfaction degree and the plan length. In order to show the feasibility of our approach, we have presented experimental results on preliminary tests focusing on the analysis of the planning performance using the satisfaction degree by simulating RARM.

The results obtained when running our architecture were shown in Fig. 3. Therefore, we have compared their performance on a set of plans for the RARM state-transitions. Since the second plan of 2 steps with maximum satisfaction degree l_2, the fifth plan of 15 steps with satisfaction $l_\uparrow$. So, it is often possible to find short, satisfactory plans quickly during the decision-making mechanism. The quality of a plan is its satisfaction degree combined with its length, where the shorter of two plans with equivalent satisfaction degrees is better.

These results are demonstrative of the capacity to dynamically treat working conditions among various conveyors, a service robot and a treating-assembling center after some time, assumes a basic role in the determination of actions during the planning. Additionally, the choice of process flexibility was affected by the making decisions. The breakdown of an individual robotic agent impacts in

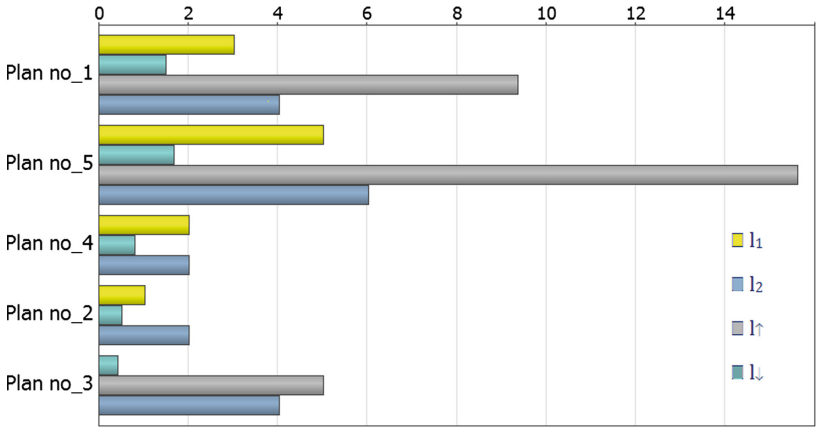

Fig. 3. The experimental results collected the length of the plan and his satisfaction degree [4].

the whole team as a result of the satisfaction degree of the plan which selects the most satisfactory alternative plan depending on the software agent architecture. At the point when the architecture is tested on the multi-robot system RARM, there are a few critical functions that have been performed. The related issues are planning and intelligent decision making.

6 Distributed Task Allocation Approach

To solve the task allocation problem, many researchers have proposed various methods of single-agent planning. Recently other researchers are interested in multi-agent planning (including our approach), they hence focus on a set of heterogeneous agents that work together to develop a course of action that satisfies the purpose of the team. Therefore, multi-agent planning defines a social approach to planning by which multiple intelligent entities work together to solve planning tasks that they are not able to solve by themselves, or to at least accomplish them better by cooperating [28].

6.1 Problem Definition

The social task allocation problem that will be depicted in this section, it can be defined as an agent not satisfactory to complete a task by itself and it needs the collaboration from other agents to achieve an action or service. We denote $A = \{a_1, ..., a_m\}$ a set of agents, that require resources to achieve tasks; and $R = \{r_1, ..., r_k\}$ a set of resources types available to A. Each agent $a \in A$ controls a fixed amount of resources for every resource type in R, which is defined by a resource function: rsc: $A \times R \rightarrow N$. Moreover, we suppose agents are connected by a *social network* as discussed before in [3,4].

We define $T = \{t_1, t_2, ..., t_n\}$ a set of needed tasks at such an agent social network. Each task $t \in T$ is then defined by a 3-tuple $\{u(t), rsc(t), loc(t)\}$, where $u(t)$ is the utility gained if task t is accomplished, rsc: $T \times R \rightarrow N$ is the resource function that specifies the amount of resources required for the accomplishment of task t and loc: $T \rightarrow A$ is the location function that defines the locations (i.e., agents) at which the tasks arrive in the social network. An agent a is the location of a task t, i.e. $loc(t) = a$, is called this task *manager*. Each task $t \in T$ needs some specific resources from the agents to complete the task. A task allocation is defined as the exact assignment of tasks to agents.

A task plan of agent consists of a list of actions to be taken in order. Each action is an attempt to acquire a particular resource, by asking the agent associated with that resource for permission to use the resource. A task agent builds a plan by maximizing the satisfaction degree described in the Sect. 5. At each timestep, a task agent performs the action presently prescribed by its plan. It does this by contacting the agent associated with the targeted resource, and asking it whether it may take the resource.

Now we formally define the important components of the problem:

- Social network: an agent social network $SN = (A, AE)$ is an undirected graph, where A is a set of agents and AE is a set of edges connecting two agents a_i and a_j significant that it exists a social connection between these two agents.
- Each agent $a \in A$ is composed of 4-tuple $\{$*AgentID(a), Neig(a), Resource(a), State(a)*$\}$, where $AgentID(a)$ is the identity of agent a, $Neig(a)$ is a set indicating the neighbors of agent a, $Resource(a)$ is the resource which agent a contains, and $State(a)$ demonstrates the state of agent.
- Multi-agent planning problem: we denote π a plan which is described by a 5-tuple $\{T, P(t), E(t), G, \mu_t\}$, where T is a set of tasks, $P(t)$ is the set of action (task) preconditions, $E(t)$ is the set of task effects, G is the problem goal and μ_t the satisfaction degree of a multi-agent flexible planning problem introduced in Sect. 5.
- Task allocation: we consider a set of tasks $T = \{t_1, t_2, ..., t_n\}$, a set of agents $A = \{a_1, ..., a_m\}$, a set of plans $\pi = \{\pi_1, ..., \pi_m\}$, and a set of resources $R = \{r_1, ..., r_k\}$ in a social network SN, a task allocation is a mapping ϕ: $T \times A \times R \times \pi \rightarrow SN$.
- Three types of agents to control system in our software agent architecture = $\{$*Manager, Participant, Mediator*$\}$. *Manager* is the agent which requests help for its task, *Participant* is the agent which accepts and performs the announced task and *Mediator* is the agent which receives another commitment of the agent for help to discover participants.
- Three states = $\{$*Busy, Committed, Idle*$\}$. In a complex system, an agent can be only in one of the three states at any timestep. When an agent is a Manager or Participant, the state of that agent is *Busy*. When an agent is a Mediator, the agent is in *Committed* state. An agent in *Idle* state is available and not assigned or committed to any task.
- *Resource Announce Message*: it is a message sent from agent a_i to agent a_j for building up neighborhood is 3-tuple described formally as $\{$*AgentID(a_i),*

TaskID(t_{a_i}), *Resource*(t_{a_i})}, where *AgentID*(a_i) represents the ID of the agent a_i, *TaskID*(t_{a_i}) is the ID of the task of the agent a_i and *Resource*(t_{a_i}) represents the resource required for the task t_{a_i}.

- *Propose Message*: it is sent from agent a_j to agent a_i for proposing a task to be achieving is 4-tuple described formally as {*AgentID*(a_j), *Resource*(a_j), *Execute*(a_j), *Utility*(a_j)}, where *AgentID*(a_j) represents the identity of the agent which proposes the resource type it contains which is defined as *Resource*(a_j), *Execute*(a_j) represents the execution time, and *Utility*(a_j) represents the utility.
- *Contract*: it is sent from agent a_i to the chosen agent a_j after satisfying with resource proposal of the neighbors a_j, the contract is defined as 4-tuple {*AgentID*(a_i), *AgentID*(a_j), *TaskID*(t_{a_i}), *Resource*(a_i)}, where *AgentID*(a_i) represents the identity of the agent which sends the contract to the agent with the identity *AgentID*(a_j), *TaskID*(t_i) represents the ID of the appropriate task to do to *Resource*(a_i).
- *Commitment*: it is sent from agent a_i to the agent a_j after selection a partial fulfilled task a_j, the commitment is defined as 4-tuple {*AgentID*(a_i), *AgentID*(a_j), *TaskID*(t_{a_i}), *rsc*(t_{a_i})^1}, where *rsc*(t_{a_i})^1 is a subset of *rsc*(t_{a_i}), which contains the unfulfilled required resources, *AgentID*(a_i) represents the identity of the agent which sends the commitment to the agent with the identity *AgentID*(a_j), *TaskID*(t_i) represents the ID of the partial fulfilled task to do.

6.2 The Principle of Distributed Task Allocation

As we indicated before, we propose a software agent architecture to manage the openness and to guarantee a coherent behavior of the MAS, in our case the multi-robot system. Accordingly, we propose a distributed task allocation approach, which is allowing agents to request help from neighbors, this would be done by allocating tasks to different agents who may be able each, to perform different subsets of tasks. Moreover, each neighbor selects the most appropriate tasks due to the single-agent planning described in the Sect. 5.

So, we illustrate the following idea: *we suppose that* $Neig(a_i)$ *stores only directly linked neighboring agents of agent* a_i *where at each timestep, these task neighboring agents perform the action presently prescribed by their most satisfying tasks. The task neighboring agents do this by contacting the agent associated with the targeted resource, and asking it whether it may take the resource.*

To make our task allocation approach efficient, it is supposed that only an *Idle* agent can be assigned to a new task as a *Manager* or a partial fulfilled task as a *Participant*, or *Committed* to a partial fulfilled task as a *Mediator*. A partial fulfilled task is a task, for which a full group is in formation procedure and has not yet formed. We present our approach which describes an interactive model between agents detailed as follows:

- When a *Manager* denoted by a_{Mn} ought to apply distributed task allocation, it then sends resource announce message to all its neighbors
  ResAnnounceMess = <AgentID(a_{Mn}), TaskID(t_{Mn}), Resource(t_{Mn})>;

– These neighboring agents receiving the *ResAnnounceMess* sent by a_{Mn},
  - **If** (*state(neighboring agent) = Idle*) **Then** the neighboring agent a_j applies the single-agent planning to select the most appropriate tasks and then sends a propose message
    ProposeMess = <AgentID(a_j), Resource(a_j), Execute(a_j), Utility(a_j)>.
  - **Else** (*state(neighboring agent) = Busy*) the neighboring agent a_j refuses and sends the following message *RefuseMess* = <AgentID(a_j)>.
– After answering the resource announce message sent by a_{Mn}
  - **If** (a_{Mn} is satisfied with many resource proposals of the neighbor) **Then** a_{Mn} will pick the agent having the highest utility, denoted by a_j, and the state of a_j will be changed to *Busy*. In case the a_{Mn} finds many agents having the highest utility then it chooses the agent a_j proposing the least execution time with a most appropriate task.
  - **Else** the a_{Mn} is satisfied with only one resource of the neighbor, then the a_{Mn} will choose this agent without any utility consideration.

  Manager a_{Mn} sends a contract to the chosen agent a_j composed of 4-tuple,
  Contract = <AgentID(a_{Mn}), AgentID(a_j), TaskID(t_{Mn}), Resource(a_{Mn})>.
– After obtaining the answer from its different cooperative neighbors, a_{Mn} then compares the available resources from its neighbors, i.e. Resoneig(a_{Mn}), with the resources required for its task t_{Mn}, namely rsc(t_{Mn})
  (Here, Resoneig(a_{Mn}) = $\bigcup_{a_j \in Neig(a_{Mn})} Resource(a_j)$). This comparison would result in one of the following two cases:
  1. **If** (rsc(t_{Mn}) $\subseteq$ Resoneig(a)) **Then** a_{Mn} can form a full group for task t_{Mn} directly with its neighboring agents which they apply the policy of single-agent planning.
  2. **Else** (Resoneig(a) $\subset$ rsc(t_{Mn})), in this condition, a_{Mn} can only form a partial group for task t_{Mn}. It then commits the task t_{Mn} to one of its neighbors. The commitment selection is based on the number of neighbors each neighbor of a_{Mn} maintaining. The more neighbors an agent has, the higher probability that agent could be selected as a *Mediator* agent to commit the task t_{Mn}.
– After selection, a_{Mn} commits its partial fulfilled task t_{Mn} to the *Mediator* agent, denoted as a_{Md}. A commitment consists of 4-tuple,
  Commitment = <AgentID(a_{Mn}), AgentID(a_{Md}), TaskID(t_{Mn}), rsc$(t_{Mn})^1$ >, where rsc$(t_{Mn})^1$ contains the unfulfilled required resources. Afterwards, a_{Md} subtracts 1 from N_{max} and attempts to discover the agents with available resources from its neighbors. If any agents satisfy resource requirement, a_{Md} will send a response message, *RespMess*, back to a_{Mn}. The agent a_{Mn} then directly makes contract with the agents which satisfy the resource requirement and have an appropriate plan of tasks. If the neighboring agents of a_{Md} cannot satisfy the resource requirement either, a_{Md} will commit the partial fulfilled task t_{Mn} to one of its neighbors again.
– This process will continue until all of the resource requirements of task t_{Mn} are satisfied, or the N_{max} reaches 0, or there is no more *Idle* agent among the neighbors. Both of the last two conditions, i.e. $N_{max} = 0$ and no more *Idle*

agent, demonstrates the failure of task allocation. In these two conditions, a_{Mn} disables the assigned contracts with the *Participant* s, and the states of these *Participant* are reset to *Idle*.
– When finishing an allocation for one task, the system is restored to its original status and each agent's state is reset to *Idle*.

Figure 4 illustrates briefly a simple example of interaction scenario between a *Manager*, *Mediator* and *Participants*.

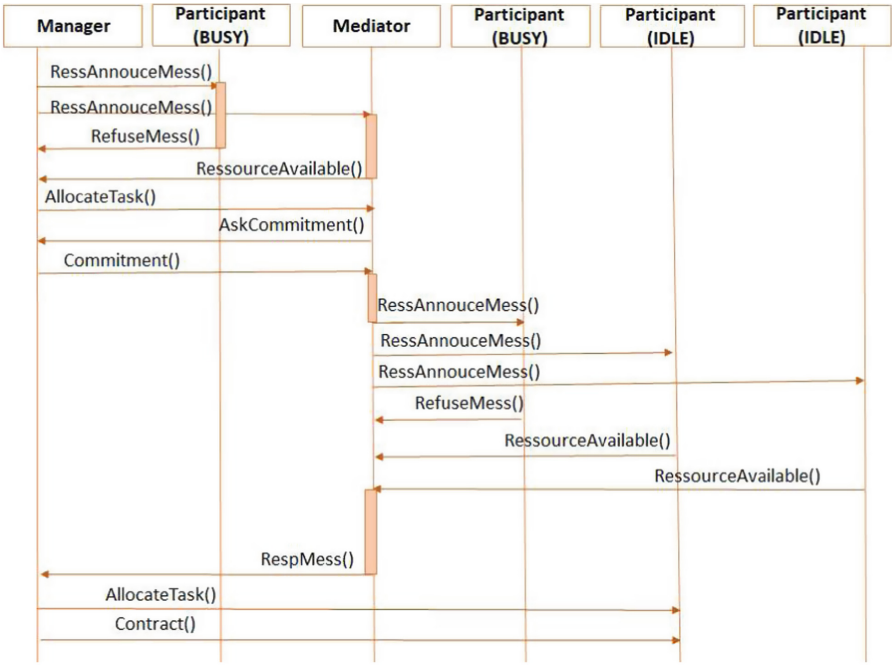

Fig. 4. The interaction process between Manager, Mediator and 4 Participants.

The following algorithm gives the pseudocode of the distributed task allocation algorithm employed by each Manager, Mediator and Participant.

Algorithm *Communicate*()
begin
switch (role)
case Manager:
  switch (step)
  case 0: // send a request to all neighbors Agents
    for j = 1 **to** NbA **do**
      send(ResAnnounceMes(Agents[j]));
    step++;

```
        break;
    case 1: // Receive accept/refusal from neighbors Agents        reply ← receive();
        if (reply = ProposeMess(Agents[j]))
            send(Contract(Agents[j]));
            Res=Res+Res(Agents[j]);
        Nb++;
        if (Nb = NbA )
            if (Res = Resource)
                step ← 4; (execute step)
            else
                step ++;
        break;
    case 2: // choose the Mediator Agent
        Max ← Neig(Agents[1])
        Mediator ← 1
        for j = 2 to NbA do
            if Neig(Agents[j]) > Max ;
                Mediator ← j
        send(Commitment(Agents[Mediator]));
        step++;
    case 3: // wait the response from the Mediator Agent
        reply ← receive();
        if (reply = ResMess(Agents[Mediator]))
            for j = 1 to list(Agents[Mediator]) do
                send(Contract(Agents[j]));
            Res += Res(list(Agents[Mediator]))
            if (Res = Resource)
                step ← 4; (execute step)
            else
                step ← 5; (cancel step)
        break;
    case 4:
     for j = 1 to length(list(Agents)) do
        send(Execute(list(Agents[j]));
     step ← 0;
     role ← participant;
     break;
    case 5:
     step ← 0;
     role ← participant;
     break;
     End switch
case Mediator:
     switch (step)
    case 0: // wait a message from the Manager Agent
```

```
    reply ← receive();
    if (reply = Commitment)
       step++;
    break;
   case 1: // send a request to all neighbors Agents
       for j = 1 to NbA do
          send(ResAnnounceMes(Agents[j]));
       step++;
       break;
   case 2: // Receive accept/refusal from neighbors Agents      reply ← receive();
    if (reply = ProposeMess(Agents[j]))
       Res=Res+Res(Agents[j]);
    Nb++;
    if (Nb = NbA )
       step ← 3; (inform the manager)
    break;
   case 3: // inform the manager Agent
    send(ResMess(Manager));
       break;
    End switch
case Participant:
    switch (step)
   case 0: // wait a message from the Manager Agent
    reply ← receive();
    if (reply = ResAnnounceMes(Manager))
       if (state = IDLE )
          send(ProposeMess(Manager));
          step++;
       else
          step ← 0;
    break;
   case 1: // wait a CONTRACT from the Manager Agent
    reply ← receive();
    if (reply = CONTRACT(Manager))
       state = BUSY
       step++;
    break;
   case 2: // Receive accept/refusal from neighbors Agents      reply ← receive();
    if (reply = Execute(Manager))
       ExcuteTask();
    state = IDLE
    step← 0;
    break;
    End switch
end
```

6.3 Experiments

We have simulated our distributed task allocation algorithm in different networks. To test the efficiency of our algorithm, we compare it with the Greedy Distributed Allocation Protocol (GDAP) [14]. In this subsection, we briefly define GDAP. Then, we introduce the experiment environment' settings. And we depict in the last sub-subsection the results and the relevant analysis.

GDAP is selected to manage task allocation problem in agent social networks. It's described briefly in [14] as follows: All *Manager* agents $a \in A$ try to find neighboring contractors (the same as *Participant* in this paper) to help them do their tasks $T_a = \{t_i \in T | loc(t_i) = a\}$. They start offering the most efficient task. Among all tasks offered, contractors select the one having the highest efficiency and send a bid to the related manager. A bid consists of all the resources the agent is able to supply for this task. If sufficient resources have been offered, the manager selects the required resources and informs all contractors of its choice. When a task is allocated, or when a manager has received offers from all neighbors but still cannot satisfy its task, the task is removed from its task list. And this is the main disadvantage of GDAP that it only relies on neighbors which may cause several unallocated tasks due to limited resources, that is exactly what our approach tries to solve.

Experimental Settings. We have been implementing our distributed task allocation algorithm and (GDAP) in JAVA and we have been testing them. There are two different settings used in our experiment. The first setup has been done in the *Small-world networks* in which most neighbors of an agent are also connected to each other. The second setup has been done in the *Scale free networks*.

Setting 1: is shown in Table 1. We assume that tasks are distributed uniformly on each *Idle* agent and resources are normally allocated to agents. The only changing variable in this setting is the average number of neighbors. This setting intends to represent the influence of neighbors' number on the performance of both our algorithm and GDAP.

Table 1. The details of Setting 1.

Setting	Quantity
Number of agents	40
Number of tasks	20
Number of different resource's types	5
Average number of resources required by each task	30

Setting 2: is shown in Table 2. We fix the average number of neighbors at 10, the ratio between the number of agents and tasks at 5/3 and the resource ratio at 1.2. The tasks are uniformly distributed. This setting is defined to show the

Table 2. The details of Setting 2.

Setting	Quantity
Number of agents	varies from 100 to 2000
Number of different resource types	20
Average number of resources required by each task	100

scalability of both our proposed algorithm and GDAP in a large scale networks with a fixed average number of neighbors.

The algorithms have been evaluated according to two criteria in this experiment; the *Utility Ratio* and the *Execution Time*, where:

$$UtilityRatio = \frac{\sum Successful - completed - tasks}{Total - of - tasks} \tag{3}$$

The unit of Execution Time is millisecond. For simplicity, we suppose that once a task has been allocated to a Participant, the Participant would successfully finish this task without failure.

Experiment Results and Analysis from Setting 1: We would like to test in this experiment the influence of different average number of neighbors on both algorithms. We notice in Fig. 5 that the Utility Ratio of our algorithm in different networks is more reliable than the GDAP algorithm. For the reason that the distribution of tasks in GDAP is only depending on the Manager neighbors, contrary to ours, in the case of need, other agents are allocated (i.e. not only the neighbors).

We can mention another factor to compare both approaches which is the network type. The results of GDAP in a small world network is higher than in a scale free network, and this could be explained by the fact that the most agents have a very few neighbors in the small network. Opposingly to that, in the scale free network when the average number of neighbors increases, the GDAP performance decreases. Which leads to say that this factor does not affect the performance of our algorithm as we take into consideration enough neighbors to obtain satisfactory resources for processing its tasks without reallocating tasks further.

Figure 6 presents the Execution Time of two algorithms in different networks depending on the average number of neighbors. The Execution Time of our algorithm is higher than that of GDAP since during execution, the agents in our algorithm reallocate tasks when resources from neighbors are unsatisfying. Furthermore, we note that the results of GDAP in a small world network is higher than in a scale free network, but compared to our algorithm are still lower and this is because it considers only neighbors which could decrease the time and communication cost during task allocation process.

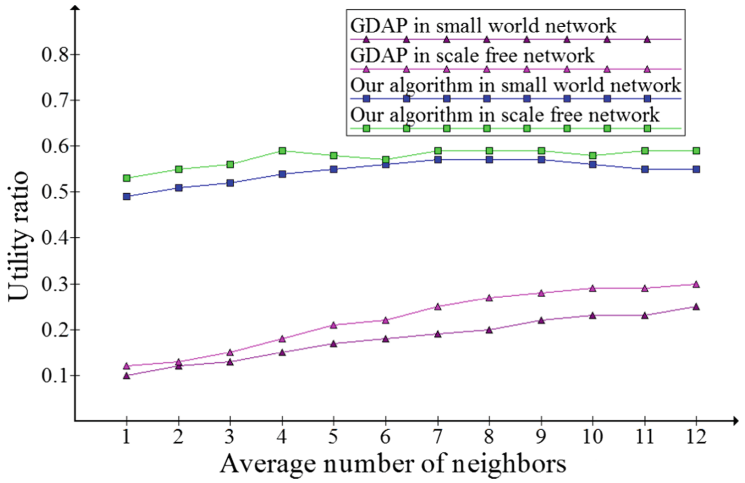

Fig. 5. The Utility ratio of the GDAP and our algorithm depending on the average number of neighbors in different type of networks [4].

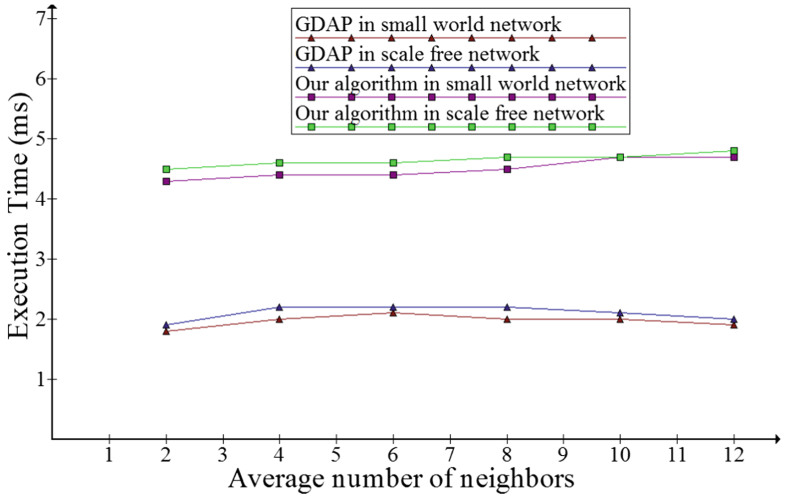

Fig. 6. The Execution time in millisecond of the GDAP and our algorithm depending on the average number of neighbors in different type of networks [4].

Experiment Results and Analysis from Setting 2: We would like to test the scalability of both GDAP and our algorithm in different large network scales like applications running on the internet. The Fig. 7 presents the Utility Ratio of GDAP which is constantly descending while that of our algorithm can save the stability and it is higher than GDAP with the increase of number of agents and simultaneously the number of tasks in a large network scale. In fact, we

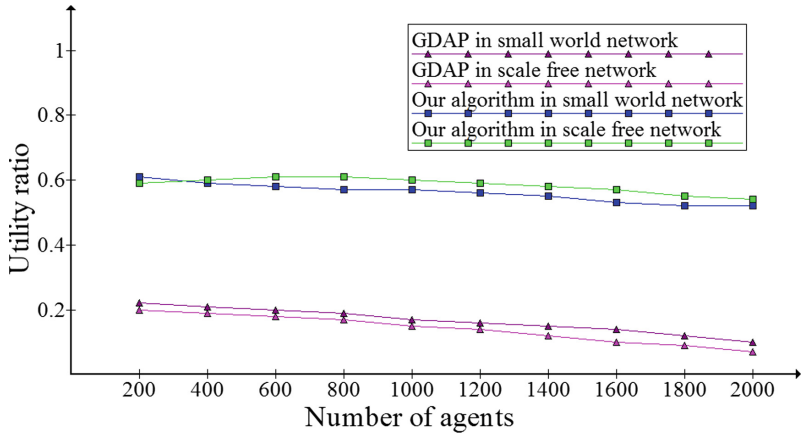

Fig. 7. The Utility ratio of the GDAP and our algorithm depending on the number of agents in different type of networks [4].

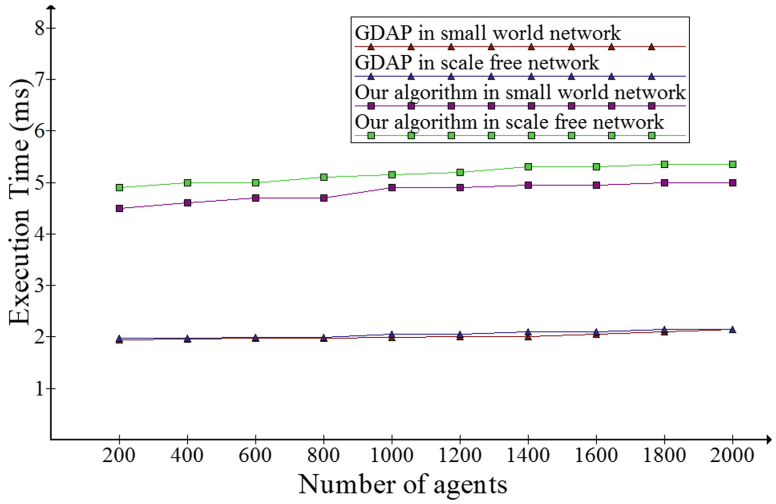

Fig. 8. The Execution time in millisecond of the GDAP and our algorithm depending on the number of agents in different type of networks [4].

can explain this by the proportional rise of the network scale, the tasks and the resource types.

Moreover the condition in small world network is better than that in scale free network. And this is justified by the same reason described above, that in scale free network, several agents only have a few neighbors which is not good for GDAP. Compared with GDAP, our algorithm is more competitive and it is favoured from task reallocation.

Figure 8 presents the Execution Time of our algorithm and GDAP in different network types. GDAP spends less time when there are more agents in the network. This is because there are more tasks despite the average number of neighbors is fixed. Accordingly, more reallocation steps cannot be avoided towards allocating these tasks, that leads to soaring in time and overhead communication. Furthermore, the graphs show that the GDAP and our algorithm almost behaves linearly and the time consumption of GDAP keeps a lower level than ours. This can be supposedly interpreted that GDAP only relies on neighboring agents.

7 Conclusion

We have proposed a software agent architecture to manage services and control embedded systems at run-time to perform self-adaptation. These architecture served us to propose a single-agent planning method that aids agent to make a convenient decision and compose better his services during the communication with others. An important originality of our method is the integration of the fuzzy logic technique to select good plans in the planning phase. This method leads us to spread more in the planning filled in dynamic MASs and we have introduced a distributed multi-task allocation approach based on a pertinent policy to solving problems of task allocation in dynamic environment. All the results are applied in this phase to a particular benchmark production system.

Although our approach overcomes many dilemmas, which exist in some current related works, due to its decentralization and reallocation features, it still has many deficiencies. They will be faced in near future work, that will focus, on the one hand, on assessing the mechanism's ability to deal with larger state action spaces than the one exemplified in this paper and review the performance benefits compared to the heavier-weight alternative solutions. On the other hand, we will focus on making the agent more intelligent and learner by utilizing some multi-agent learning methods. Indeed, the sending of the message can be expensive and it is not possible to broadcast all resource announce messages to all neighbors at each timestep. In fact, agents will be capable to autonomously determine which messages to send, to whom, and at what point in their execution is. In this context, communication should be seen as a task to do in the own decision of the agent that it will deal with both the actions and the possible interactions. We also plan, in the future, to enhance our approach by considering that each agent in the system has multiple resources instead of only one.

References

1. Gharbi, A., Ben Noureddine, D., Ben Halima, N.: Building multi-robot system based on five capabilities model. In: 12th International Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, Spain, pp. 270–275 (2015)

2. Ben Noureddine, D., Gharbi, A., Ben Ahmed, S.: An approach for multi-robot system based on agent layered architecture. Int. J. Manag. Appl. Sci. (IJMAS) **2**(12), 135–143 (2016)
3. Ben Noureddine, D., Gharbi, A., Ben Ahmed. S.: Multi-agent deep reinforcement learning for task allocation in dynamic environment. In: 12th International Conference on Software Technologies, Madrid, Spain, pp. 17–26 (2017)
4. Ben Noureddine, D., Gharbi, A., Ben Ahmed, S.: A social multi-agent cooperation system based on planning and distributed task allocation: real case study. In: 13th International Conference on Software Technologies, Porto, Portugal, pp. 483–493 (2018)
5. Jennings, N.R., Sycara, K., Wooldridgse, M.: A roadmap of agent research and development. Auton. Agents Multi-agent Syst. **1**, 7–38 (1998)
6. Mejia, M., Peña, N., Muñoz, J.L., Esparza, O., Alzate, M.: DECADE: distributed emergent cooperation through adaptive evolution in mobile ad hoc networks. Ad Hoc Netw. **10**, 1379–1398 (2012)
7. Del Val, E., Rebollo, M., Botti, V.: Promoting cooperation in service-oriented MAS through social plasticity and incentives. J. Syst. Softw. **86**, 520–537 (2013)
8. Das, G.P., McGinnity, T.M., Coleman, S.A., Behera, L.: A distributed task allocation algorithm for a multi-robot system in healthcare facilities. J. Intell. Robot. Syst. **84**, 1–26 (2014)
9. Sun, Q., Garcia-Molina, H.: SLIC: a selfish link based incentive mechanism for unstructured peer-to-peer networks. In: 24th International Conference in Distributed Computing Systems, Tokyo, Japan, pp. 506–515 (2004)
10. Wei, G., Zhu, P., Vasilakos, A.V., Mao, Y., Luo, J., Ling, Y.: Cooperation dynamics on collaborative social networks of heterogeneous population. IEEE J. Sel. Areas Commun. **31**, 1135–1146 (2013)
11. Hruz, B., Zhou, M.: Modeling and Control of Discrete-event Dynamic Systems with Petri Nets and Other Tools. Springer, London (2007). https://doi.org/10.1007/978-1-84628-877-7
12. Miguel, I., Jarvis, P., Shen, Q.: Flexible graphplan. In: 14th European Conference on Artificial Intelligence, Berlin, Germany, pp. 4506–4514 (2000)
13. Miguel, I., Giret, A.: Feasible distributed CSP models for scheduling problems. Eng. Appl. Artif. Intell. **21**(5), 723–732 (2008)
14. Weerdt, M.D., Zhang, Y., Klos, T.: Distributed task allocation in social networks. In: 6th International Conference on Autonomous Agents and Multi-agent Systems Distributed Computing Systems, Honolulu, Hawaii, USA, pp. 500–507 (2007)
15. Farinelli, A., Farinelli, R., Iocchi, L., Nardi, N.: Multi-robot systems: a classification focused on coordination. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(5), 2015–2028 (2004)
16. Dudek, G., Jenkin, M., Milios, E.: A taxonomy of multirobot systems. In: Robot Teams: From Diversity to Polymorphism, pp. 3–22 (2002)
17. Zheng, X., Koenig, S.: Reaction functions for task allocation to cooperative agents. In: 7th International Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, pp. 559–566 (2008)
18. Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Wooldridge, M.J., Sierra, C.: Automated negotiation: prospects, methods and challenges. Group Decis. Negot. **10**(2), 199–215 (2001)
19. Fatima, S.S., Wooldridge, M.: Adaptive task and resource allocation in multi-agent systems. In: 5th International Conference on Autonomous Agents, Montreal, QC, Canada, pp. 537–544 (2001)

20. Gatti, N., Giunta, D., Marino, S.: Alternating-offers bargaining with one-sided uncertain deadlines. An efficient algorithm. Artif. Intell. **172**(8), 1119–1157 (2008)
21. An, B., Lesser, V., Sim, K.M.: Strategic agents for multi-resource negotiation. Auton. Agent Multi Agent Syst. **23**(1), 114–153 (2011)
22. An, B., Gatti, N., Lesser, V.: Bilateral bargaining with one-sided two-type uncertainty. In: The International Joint Conference on Web Intelligence and Intelligent Agent Technology, DC, USA, pp. 403–410 (2009)
23. An, B., Lesser, V., Irwin, D., Zink, M.: Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In: 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, ON, Canada, pp. 981–988 (2010)
24. Choi, H.-L., Brunet, J., How, J.P.: Consensus-based decentralization auctions for robust task allocation. IEEE Trans. Robot. **25**(4), 912–926 (2009)
25. Cramton, P., Shoham, Y., Steinberg, R.: An overview of combinatorial auction. ACM SIGecom Exch. **7**(1), 3–14 (2007)
26. Whitbrook, A., Meng, Q., Chung, P.W.H.: A novel distributed scheduling algorithm for time-critical, multi-agent systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany, pp. 6451–6458 (2015)
27. Zhao, W., Meng, Q., Chung, P.W.H.: A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. IEEE Trans. Cybern. **46**(4), 902–915 (2016)
28. Weerdt, M.D., Clement, B.: Introduction to planning in multiagent systems. Multiagent Grid Syst. **5**(4), 345–355 (2009)