Proceedings of the International Multiconference on Computer Science and Information Technology pp. 125–130
ISBN 978-83-60810-14-9
ISSN 1896-7094

Coalition formation in multi-agent systems—an evolutionary approach

Wojciech Gruszczyk
Wrocław University of Technology
Computer Engineering
Email: 141044@student.pwr.wroc.pl

Halina Kwaśnicka
Wrocław University of Technology
Computer Engineering
Email: halina.kwasnicka@pwr.wroc.pl

***Abstract*—The paper introduces solution of Coalition Formation Problem (CFP) in Multi-Agents Systems (MAS) based on evolutionary algorithm. The main aim of our study is to develop an evolutionary based algorithm for creation of coalitions of agent for solving assumed tasks. We describe the coding schema and genetic operators such as mutation, crossover and selection that occurred to be efficient in solution of CFP. Last part of the document provides a brief comment on our research results.**

***Index Terms*—Coalition formation, evolutionary algorithm, multi–agent systems**

I. Introduction

CONCEPT of agents (term agent will be used without distinction between software and hardware agents) is strongly connected with artificial intelligence (AI). Because the term "agent" is defined in literature in many different ways we will use definition first proposed in [7]:

Agent is software or hardware computer system that is/has:

1) Autonomous: agent takes actions without interference of a human and has control over taken actions,
2) Social ability: agents communicate (between themselves and/or with people),
3) Reactivity: agents have some perception of environment that they are part of and may react to changes in the environment,
4) Activity: agents may take actions to change their environment in order to achieve their goals.

When the environment contains at least two agents we talk about multi-agent system (MAS). Very often MAS are distributed. Many aspects of such systems have been widely discussed in literature. In this paper we present a new solution of the problem of coalition formation (CFP).

A. Coalition Formation

Many tasks cannot be completed by a single agent because of limited resources or capabilities of agents. Very often, even if a task may be completed by a single agent, his performance may occur too low to be acceptable. In such a situation agents may form groups to solve the problem by cooperation. Many organizational paradigms have been distinguished in the context of MAS (see [9]). This work is focused on coalitions—groups of cooperative agents, working together on a given task, short-lived and goal-directed, being a flat structure. Initially agents are independent and do not cooperate. When they cannot complete their tasks individually agents may exchange information and try to form coalitions which gives them best efficiency (of course the efficiency must be defined in terms of solved problem).

Evaluation of all possible shapes of coalitions depends exponentially on the number of agents. For example having m agents and n tasks to solve all n^m coalitions (with each coalition delegated to a particular task) must be evaluated to guarantee that the best coalition shape was found. Finding the optimal partition of agents set by checking the whole space may occur too expensive (in terms of time). Short lifetime of coalition may lead to a situation when time of computation is far longer than the time of existence of a particular coalition. Therefore it may create a bottleneck of a MAS. Many methods have been proposed to solve CFP. Further part of the introductory chapter will give a short summary of them.

B. CFP—a short overview

1) Any-time solution with worst case guarantee: Original work given by [8] presents any-time solution with worst case guarantee. The idea is based on a remark, that searching subset of possible coalition shapes that contains all possible subsets of the set of agents may guarantee quality of the solution. That is why the best solution must consist of already searched and evaluated subsets (best solution is not worse than the best solution found so far). To make the idea clear [8] suggested representing all partitions as a graph of coalition structures (Fig. 1):

Level LV_i denotes that the structures belonging to this level consist of i coalitions. Searching LV_1 and LV_2 assures that all subsets have been checked and may provide guarantee on the result. Then [8] suggests searching bottom up (in the picture levels LV_4, LV_3). Checking all coalition structures leads to brute force search and worst case guarantees are low before huge amount of the space has been searched (as shown in [5]).

2) Distributed algorithm: Distributed algorithms for solving CFP are proposed (among others) in [2] and [5]. The solution proposed in [5] (compared to other distributed methods) significantly minimizes efforts on communication between agents.

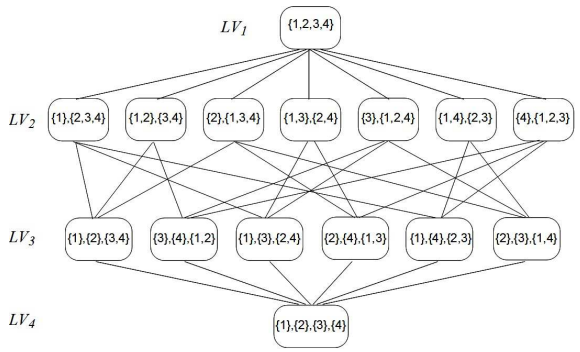

Fig. 1. Coalition structure graph for four agents

3) Genetic algorithm based method: In [1] the authors presents a genetic algorithm where two-dimensional, binary chromosome coding is used. The method uses so called *two dimensional "or" crossover operator* that seems to be troublesome, because after using the operator we have to review child chromosome in order to "repair" it when the operator breaks the rules of representation assumed in the method. Inspired by [1] we decided to propose an alternative chromosome coding and operators to solve CFP.

Good introduction to MAS provides [3]. Brief descriptions of organizational paradigms other than coalition of agents are presented in [9]. Environments of self-oriented, conflicting agents are described in [6]. Comprehensive introduction to evolutionary programming is provided in [4].

The main aim of our study is to develop a method of coalitions formation in multi-agent systems using evolutionary approach. The method should combine good efficiency in solving CFP with natural chromosome coding and genetic operators.The paper is structured as follows. Next section will introduce formal representation of agents and coalitions. Third chapter will cover implementation issue and experimental results. The last part of the paper concludes the results and point further research directions.

II. The proposed method

In this section we present a formal model of our problem and the proposed evolutionary algorithm.

A. Model

At the beginning we assumed as follows:

1) Agents solve a finite number of tasks,
2) Each agent has some ability (resources) to solve each task,
3) Each task requires some abilities (resources) to be solved,
4) Particular agent may have insufficient abilities (resources) to solve a task,
5) Agents are cooperative,
6) Agents are altruistic—their own good is less important than the good of the system as a whole,
7) Each agent must contribute to exactly one coalition,
8) We want to solve as many tasks as possible with the given set of agents.

According to these assumptions the following model of the problem has been proposed:

Set of tasks is represented as a vector T:

$$T =< T_1, \ldots, T_n >, T_i \in (\mathbb{N}_+ \setminus \{1\}), i > 1$$

Value t at position T_i means that to solve task i t units of resource (ability) are needed. We assume that $i > 1$ because for $i = 1$ the problem is trivial.

Let A be a set of agents:

$$A = \{A_1, \ldots, A_k\}$$

where each agent A_i is represented as:

$$A_i =< U_1, \ldots, U_n >, U_i \in \mathbb{N}, i > 1$$

Value p at position U_j means that agent i has abilities (resources) of p to solve task j.

Set K representing partition of set A with properties given below will be the result of proposed algorithm:

- Each element $C_i \in K$ will be a coalition,
- Each element C_i will be a set of agents such that $C_i \subseteq A$ (in particular C_i may be empty),
- For each $C_i, C_k \in K$: $C_i \cap C_k = \emptyset, i \neq k$
- $\bigcup_{i=1}^{n} C_i = A, n = card(K)$
- Membership of an agent in a coalition $i, 1 \leq i \leq n$ means that the agent contributes to solution of task i.
- K represents partition that achieved the best (the highest) value of fitness function (described further).

We defined total ability of a coalition i to solve its task as a simple sum of abilities (resources) of its (coalition's) members' abilities to solve the task. Formally total ability δ_i is given as:

$$\delta_i = \sum_{A_t \in C_i} U_i[A_t]$$

We say that task $i, 1 \leq i \leq n$ is solved in a given partition when (for given i) $\delta_i \geq T_i$.

Evolutionary algorithm operates on a set of individuals—V—each of them representing some partition of set A. Each individual is represented as a vector v_i:

$$v_i =< d_1, \ldots, d_q >, \text{ where:}$$
$$d_i \in \mathbb{N} \wedge d_i \in [1, card(T)], q = card(A)$$

All genetic operations are conducted on elements (individuals) from set V. Both operators used in our method (crossover and mutation) are presented in next subsection.

B. Evolutionary algorithm

Evolutionary algorithm that we developed uses $(\mu + \lambda)$ strategy (for detailed information about evolutionary strategies see [4]). We use $\mu = \lambda$. To preserve the best (so far) solution individual with the highest value of fitness function always survives (is added to child population).

Figure 2 presents block diagram presenting main loop of the algorithm.

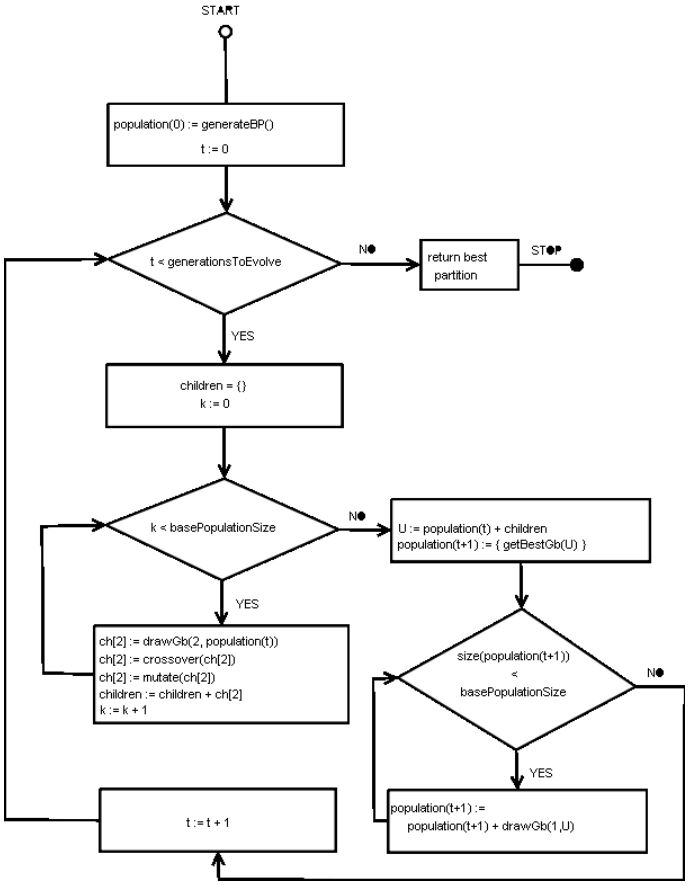

Fig. 2. Evolutionary algorithm block diagram

A few elements on the diagram (Fig. 2) need comment:

- Function drawGb(2, population(t)) draws two individuals from population(t) with returning,
- Function getBestGb(U) gets individual with the highest fitness function value, suffix Gb means that this operation is conducted with returning.

C. Operators, fitness function, base population

1) Crossover: We use standard two-point crossover operator. Chromosomes to cross are drawn (chosen stochastically) with giving back with probability proportional to their fitness (fitness and its function are discussed later). Both produced chromosomes are added to child population. Proposed crossover operator preserves the property that each agent (in a child chromosome) belongs to exactly one coalition. Therefore we do not need to repair the chromosome. Two-point crossover (and many more) are discussed in [4].

2) Mutation: Mutation is carried out with a given probability p on a single gene. The operator randomly chooses whether to mutate the gene. Mutation randomly changes the coalition to which the agent represented by the gene is assigned. Alike crossover, mutation does not break the chromosome therefore no repairs are needed.

3) The initial population: Initial population (the first generation) is generated randomly. Its size is given arbitrarily and all created chromosomes are correct in terms of definitions given in previous chapters.

4) Fitness function: A fitness function is essential for the method. As we want to maximize the number of tasks solved by coalitions of agents following assumptions about the function have been made:

- An individual should be punished for non-completion of a task,
- And individual should be punished for exceeding the required abilities (resources) by a coalition ($\delta_i > T_i$),
- Punishment for non-completion of a task should be more harmful than the punishment for exceeding the boundary,
- Fitness function must be non-negative.

Based on the above remarks we designed the following way of fitness calculation (Algorithm 1):

Algorithm 1 Calculating fitness function value

$fitVal \leftarrow numberOfTasks$
for AllCoalitions : Ci **do**
  if $\delta_i < T_i$ **then**
    $fitVal \leftarrow fitVal - 1.2 + max(0.2, \delta_i/T_i)$
  else
    if $T_i \neq 0$ **then**
      $fitVal \leftarrow fitVal - min(0.05, (\delta_i - T_i)/T_i)$
    else
      $fitVal \leftarrow fitVal - 0.05$
    end if
  end if
end for
return fitVal

All coefficients in the above algorithm were chosen by manual testing of various values. Given coefficients occurred to be satisfactory in solved problem but should be treated as a hint not as a rule.

At this point it is worth mentioning that the given fitness function is not perfect. It may happen in some situations that some coalition's structure A solving more tasks than coalition B has lower fitness value. It is caused by the punishment for exceeding task's boundary of required resources (abilities). Such a situation usually does not spoil algorithm's results, nevertheless, it is discussed further together with experimental results.

III. Implementation, results of experiments

Our implementation language was Java. Neither evolutionary algorithm nor MAS frameworks were used. Usage of Strategy pattern occurred to be useful while trying different operators, population generators, etc. See [10] for comprehensive information about design patterns.

Different types of test data have been used. We prepared:

- 2 types of agent populations (random and predefined),
- 2 sizes of agent population,
- 2 sizes of task set,
- 2 sizes of base population.

Brief comment of test data is provided in the next section. After a series of initial test with different values of mutation probability and number of generations we decided to use:

- Probability of mutation: 1%,
- Number of generations: 500.

Given values provided good results, nevertheless, (like for fitness function) they should be treated as a hint.

5) Agent populations, tasks requirements: We used two sizes of agent populations containing respectively: 20 agents (solving 5 tasks) and 30 agents (solving 10 tasks). Each population size was tested in two variants: predefined and generated. Predefined population of 20 agents was defined as:

$$\begin{gathered} A_{20} = \{ \\ <2,1,1,1,1>,<2,1,1,1,1>, \\ <2,1,1,1,1>,<2,1,1,1,1>, \\ <1,2,1,1,1>,<1,2,1,1,1>, \\ <1,2,1,1,1>,<1,2,1,1,1>, \\ <1,1,2,1,1>,<1,1,2,1,1>, \\ <1,1,2,1,1>,<1,1,2,1,1>, \\ <1,1,1,2,1>,<1,1,1,2,1>, \\ <1,1,1,2,1>,<1,1,1,2,1>, \\ <1,1,1,1,2>,<1,1,1,1,2>, \\ <1,1,1,1,2>,<1,1,1,1,2> \\ \} \end{gathered}$$

Set of 30 agents was defined as:

$$\begin{gathered} A_{30} = \{ \\ <2,1,1,1,1,1,1,1,1,1>,<2,1,1,1,1,1,1,1,1,1>, \\ <2,1,1,1,1,1,1,1,1,1>,<2,1,1,1,1,1,1,1,1,1>, \\ <2,1,1,1,1,1,1,1,1,1>,<2,1,1,1,1,1,1,1,1,1>, \\ <2,1,1,1,1,1,1,1,1,1>,<1,2,1,1,1,1,1,1,1,1>, \\ <1,2,1,1,1,1,1,1,1,1>,<1,2,1,1,1,1,1,1,1,1>, \\ <1,2,1,1,1,1,1,1,1,1>,<1,2,1,1,1,1,1,1,1,1>, \\ <1,2,1,1,1,1,1,1,1,1>,<1,2,1,1,1,1,1,1,1,1>, \\ <1,1,4,1,1,1,1,1,1,1>,<1,1,4,1,1,1,1,1,1,1>, \\ <1,1,1,4,1,1,1,1,1,1>,<1,1,1,4,1,1,1,1,1,1>, \\ <1,1,1,1,4,1,1,1,1,1>,<1,1,1,1,4,1,1,1,1,1>, \\ <1,1,1,1,1,4,1,1,1,1>,<1,1,1,1,1,4,1,1,1,1>, \\ <1,1,1,1,1,1,4,1,1,1>,<1,1,1,1,1,1,4,1,1,1>, \\ <1,1,1,1,1,1,1,4,1,1>,<1,1,1,1,1,1,1,4,1,1>, \\ <1,1,1,1,1,1,1,1,4,1>,<1,1,1,1,1,1,1,1,4,1>, \\ <1,1,1,1,1,1,1,1,1,4>,<1,1,1,1,1,1,1,1,1,4> \\ \} \end{gathered}$$

T vectors for 20 and 30 predefined agents were given as (respectively):

$$T_{20} =< 8,8,8,8,8 >$$

$$T_{30} =< 14,14,8,8,8,8,8,8,8,8 >$$

In the case of generated sets of agents we used a generator that (having predefined T vector) prepared such a set that could solve all tasks without a punishment (in terms of fitness function) for exceeding the threshold of needed resources. Each agent in a generated set has positive value of ability to solve any task so it is not obvious to which coalition it should be assigned. Of course the generator does not guarantee that only one optimal solution exists, but it ensures existence of at least one.

We decided to use test data that ensured existence of a solution that solved all tasks. Resources were limited (in fact limits were very rigid) and search spaces were huge (for 10 tasks and 30 agents the whole space consisted of 10^{30} possible coalitions!). The choice was suggested by used method which is worth applying to extremely hard combinatorial problems that cannot be solved in acceptable amount of time by deterministic search of the space (e.g. dynamic programming).

6) The initial population: We tried two sizes of the initial population for the evolutionary algorithm: 25 and 50 individuals.

7) Experimental results: To find optimal parameters for the algorithm we compared four cases:

- Generated test data, base population size: 25/50, number of tasks: 5, number of agents: 20,
- Generated test data, base population size: 25/50, number of tasks: 10, number of agents: 30,
- Predefined test data, base population size: 25/50, number of tasks: 5, number of agents: 20,
- Predefined test data, base population size: 25/50, number of tasks: 10, number of agents: 30.

Results of our experiments are listed in tables I, II, III and IV:

TABLE I
EXPERIMENT I: GENERATED TEST DATA, BASE POPULATION SIZE: 25/50, NUMBER OF TASKS: 5, NUMBER OF AGENTS: 20

No	Population size	Tasks solved (best)	Fitness function (best)	In generation
1	25	5	4.95	16
2		5	4.95	35
3		5	5.00	22
4		5	4.95	17
5		5	5.00	17
6		5	5.00 (4.95)[1]	146 (61)[2]
7		5	5.00 (4.90)	269 (13)
8		5	5.00 (4.90)	46 (8)
9		5	5.00 (4.95)	14 (4)
10		5	5.00 (4.85)	371 (7)
11	50	5	5.00	18
12		5	5.00 (4.85)	55 (6)
13		5	5.00 (4.85)	127 (12)
14		5	5.00 (4.80)	25 (11)
15		5	5.00 (4.85)	54 (13)
16		5	5.00 (4.95)	24 (10)
17		5	5.00 (4.90)	388 (7)
18		5	5.00	16
19		5	5.00	18
20		5	5.00 (4.80)	35 (8)

TABLE II
EXPERIMENT II: GENERATED TEST DATA, BASE POPULATION SIZE: 25/50, NUMBER OF TASKS: 10, NUMBER OF AGENTS: 30

No	Population size	Tasks solved (best)	Fitness function (best)	In generation
1	25	10	9.85 (9.80)	272 (207)
2		10	9.90 (9.80)	298 (94)
3		9	9.65	55
4		9	9.70 (9.15)	429 (37)
5		10	9.95 (9.85)	481 (113)
6		10	10.00	49
7		9	9.55 (9.45)	142 (107)
8		10	10.00 (9.90)	325 (269)
9		10	9.80 (9.75)	447 (245)
10		9	9.45	167
11	50	10	9.90 (9.85)	203 (184)
12		10	9.90	89
13		10	9.90	356
14		10	10.00 (9.95)	210 (139)
15		10	9.95 (9.90)	382 (132)
16		10	9.95 (9.85)	190 (96)
17		10	9.95 (9.65)	90 (34)
18		9	9.65 (9.35)	350 (196)
19		10	9.90	313
20		10	9.85	129

TABLE III
EXPERIMENT III: PREDEFINED TEST DATA, BASE POPULATION SIZE: 25/50, NUMBER OF TASKS: 5, NUMBER OF AGENTS: 20

No	Population size	Tasks solved (best)	Fitness function (best)	In generation
1	25	4	4.05 (4.00)	101 (18)
2		5	5.00	98
3		5	5.00	178
4		4	4.00	38
5		5	5.00	124
6		5	5.00	80
7		5	5.00	73
8		5	5.00	126
9		5	5.00	86
10		5	5.00	82
11	50	5	5.00	189
12		5	5.00	94
13		4	4.00	27
14		5	5.00	42
15		5	5.00	315
16		5	5.00	107
17		5	5.00	37
18		5	5.00	114
19		5	5.00	33
20		5	5.00	41

As we can see after 500 generations of algorithm's work in every test case I-IV we obtained solution no worse than 80% of optimal one. Tests I and III in most cases were solved returning optimal partition. For generated test data we obtained 100% accuracy. Average quality of solution in these cases exceeded 90% Two factors had impact on our results:

1) Search space was relatively small (5^{20} compared to 10^{30} makes $2^{20} \cdot 10^{10} \approx 10^{16}$ times smaller search space),
2) In the case of generated data it was not so rigid as predefined so more than one optimal solution might have existed.

In both cases the amount of 500 generations was to big number. For generated data optimal solutions were found in less than 100 generations. For predefined data in most cases 200 generations were enough to obtain comparable results.

In test cases number II and IV we achieved accuracy no worse than 80% of optimal solution. Again generated data occurred to be less rigid and gave better results (exceeding 90% of the best solution). For generated data 250 generations of evolutionary algorithm's work occurred to be enough to provide comparable results. Even after 500 generations our algorithm in most cases did not exceeded 8 solved tasks (only 3 times we achieved 9 tasks solved and it took over 270 generations).

[1]Value in brackets shows the value of fitness function which provided the same number of solved tasks. The value was achieved in a generation which number is shown in brackets in column "In generation". See [2]. The value is given only when the difference between generations is significant.

[2]Situation symmetric to described in [1].

Figures 3 and 4 show algorithm's progress. In both cases we put the number of generation on OX and the value of a given function on OY (3 functions are presented and described on the plots). Both plots show a case where despite the growth of fitness of the best individual the number of tasks solved by (current) best solution is lower (on figure 3 we see this situation between 20-th and 200-th generation; on figure 4 about 50-th generation). As mentioned earlier, this situation is caused by punishment used in fitness function. Despite the fact that such a situation is not frequent it may happen that solution with the highest value of fitness function does not provide partition that solves most tasks (even if such partitions were checked).

IV. CONCLUSIONS AND FURTHER RESEARCH

Our approach occurred to be efficient in solving CFP. Its power is remarkable especially in huge spaces where brute force search and other regular and deterministic search methods cannot be applied because of limited resources (especially time).

Comparing to [1], implementation of GA presented in this paper uses natural coding of chromosome (simple vector rather than complex 2D structure) which makes it easier to use and preserves classic flow of genetic (evolutionary) algorithm. Simple chromosome coding made it possible to use standard mutation and crossover operators. Both of them create individuals that do not break domain constraints (one agent belongs to exactly one coalition) while *two dimensional "or" crossover operator* proposed in [1] enforced checking of

TABLE IV
EXPERIMENT IV: PREDEFINED TEST DATA, BASE POPULATION SIZE: 25/50, NUMBER OF TASKS: 10, NUMBER OF AGENTS: 30

No	Population size	Tasks solved (best)	Fitness function (best)	In generation
1	25	8	8.45	215
2		8	8.09 (7.95)	154 (68)
3		9	9.00	273
4		8	8.09 (7.90)	260 (154)
5		8	8.73 (8.30)	463 (87)
6		8	8.44 (7.94)	87 (24)
7		8	8.51 (7.96)	252 (76)
8		8	8.09 (7.95)	260 (47)
9		8	8.09 (7.95)	391 (57)
10		8	8.09 (7.95)	140 (37)
11	50	8	8.09 (7.90)	314 (68)
12		8	8.44 (7.95)	252 (77)
13		8	8.09 (7.95)	162 (61)
14		8	8.01 (7.85)	495 (393)
15		8	8.44 (7.95)	383 (44)
16		8	8.09	405
17		8	8.09 (7.95)	190 (72)
18		9	9.00	275
19		8	8.09 (7.90)	402 (87)
20		9	9.00	407

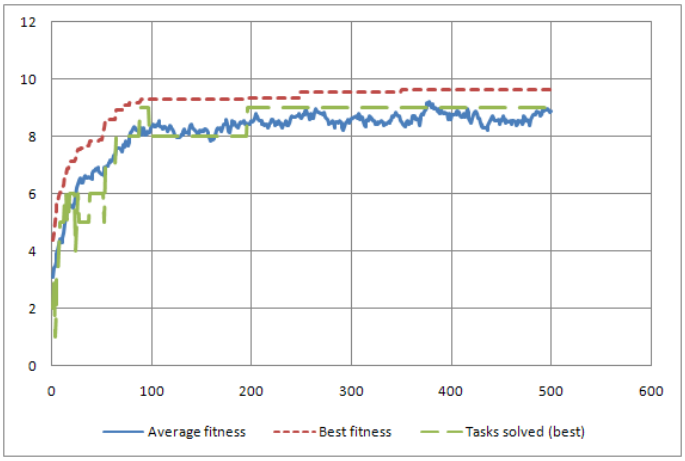

Fig. 3. Algorithm's progress: plot for test II (row number 18)

child chromosomes because they might break the constraint of one-to-one agent to coalition assignment.

Another thing worth mentioning is the initial population. In [1] every coalition structure that does not solve all tasks is assigned fitness of 0. Initial population is generated with revisions whether all tasks are solved and only total efficiency of a whole system (solving all tasks) is being optimized. Such assumption enforces low bounds on available resources (agents may solve all tasks). Our approach does not assume that all tasks must be solved (we do not know whether even a single one can be solved at all). Therefore our initial population is generated randomly, without further revisions.

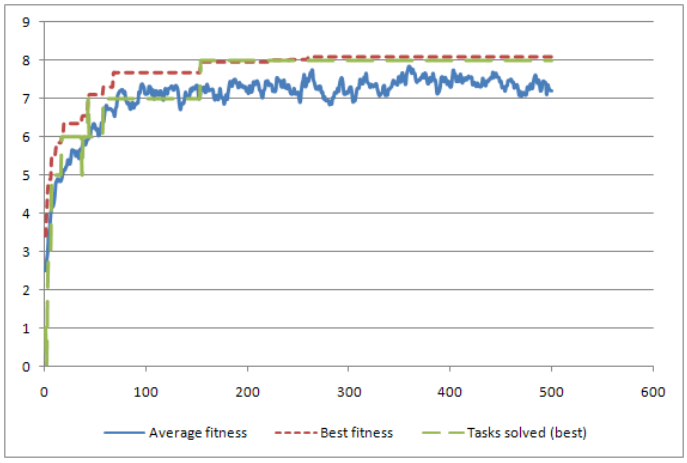

Fig. 4. Algorithm's progress: plot for test IV (row number 4)

Main optimization target of our method is the number of solved tasks which seems to be very important in real-time, strongly constrained domains.

The main limitation of our method is its centralization. As we mentioned in the first section, MAS are often distributed (or are at least independent processes or threads). Centralized control must assume that agents are cooperative not competitive. Such an assumption may be acceptable in MAS environments composed of agents belonging to one company. It may be, however, too strong in the case of distributed environments accessible for agents belonging to different companies and having contradictory aims. Therefore we are going to develop a method of coalition formation based on the search of consensus (equilibrium) through negotiation. In such a system the only requirement for agents would be to understand the protocol of communication. Therefore there are no obstacles to make the system fully decentralized and distributed.

REFERENCES

[1] Jingan Yang and Zhenghu Luo, *Colaition formation mechanism in multi-agent systems based on genetic algorithms*, Applied Computing Soft (Elsevier), 2006.
[2] O. Sheory and S. Kraus, *Task allocation via coalition formation among autonomous agents*, Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, pp. 655–661.
[3] G. Weiss—editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, 2000.
[4] J. Arabas, *Wykłady z algorytmw ewolucyjnych*, WNT, 2004.
[5] T. Rahwan, *Algorithms for Coalition Formation in Multi-Agent Systems*, University Of Southampton (PhD), 2007.
[6] C. Tessier—editor, *Conflicting Agents: Conflict Management in Multi-Agent Systems*, Kluwer Academic Publishers, 2001.
[7] M. Wooldridge and N.R. Jennings, *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, 1995, vol. 10/2, pp. 115–152.
[8] T. Sandholm et al, *Coalition Structure Generation with Worst Case Guarantees*, Artificial Intelligence, vol. 111/(1-2), pp. 209-238, 1999.
[9] B. Horling and V. Lesser, *A survey of multi-agent organizational paradigms*, The Knowledge Engineering Review, vol. 19/4. pp. 281–316, 2005
[10] E. Gamma and J. Vlissides and R. Johnson and R. Helm, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman Publishing Co., 1995.