Agents Clustering within Multi-Agent System by means of Multiobjective Iterative Algorithm

Jiří Kubalík
Department of Cybernetics
CTU Prague Technická 2
166 27 Prague 6
Czech Republic
kubalik@labe.felk.cvut.cz

Pavel Tichý, Radek Šindelář
Rockwell Automation s.r.o.
Pekařská 10a
15500 Prague 5
Czech Republic
ptichy,rsindelar@ra.rockwell.com

Raymond J. Staron
Rockwell Automation
1 Allen-Bradley Drive
Mayfield Hts.
OH 44124-6118, USA
rjstaron@ra.rockwell.com

Abstract

This paper deals with static agent clustering analysis within multi-agent systems. The goal is to find clusters of agents within the multi-agent system that minimize communication among different clusters. Originally, a modified fuzzy clustering algorithm was used for this purpose. In this paper, the problem was transformed to multi-objective optimization. A multi-objective iterative optimization algorithm called "Multi-objective Prototype Optimization with Evolved iMprovement Steps" (mPOEMS) was used as the clustering algorithm. Results achieved with the proposed mPOEMS algorithm are significantly better than those achieved by the modified fuzzy clustering algorithm. Another advantage of mPOEMS is that it provides a user with multiple optimal solutions obtained in just one run from which the best one can be selected by providing some additional information.

1 Introduction

There is a need for visualization tools for viewing changing coupling relationships and clustering behaviors in multi-agent systems. Information about this coupling or agent clustering can be used by a system developer or integrator in various ways. For example, the structure of agent clusters can be mapped to agent execution hardware. Placing agents that belong to one cluster into one computation unit can improve system performance if cost of communication within a computation unit is lower than communication outside of it. The opposite question that can be answered is how to place agents into a given number of computation units such that communication among these units is minimized.

To enable visualization and debugging of communication in a multi-agent system, a tool called the Java Sniffer [Tichý *et al.*, 2006] has been created and enhanced to support detection and visualization of agent clustering [Staron *et al.*, 2007]. There are two methods for analysis. The first one deals with a dynamic model that considers a collection of agents as a system of masses exerting attractive, repulsive and frictional forces on one another, observing the changes in each agent's position and velocity. Messages sent between agents represent attractive force and thus agents form clusters to reflect their relation. Further topological analysis via Gustafson-Kessel algorithm [Höppner and Runkler, 1999] is used to identify clusters.

In this paper we focus on a static analysis that can be obtained by considering all messages sent among agents simultaneously and directly forming agent clusters using a predefined criterion without directly computing their positions [Staron *et al.*, 2007]. The criteria are to minimize communication among agent clusters and maximize communication within each agent cluster. Objective function-based algorithms work with pattern positions while our task is given by the matrix of messages. The number of messages sent between two agents is indirectly proportional to an agent distance but the transformation into pattern positions is not straightforward. Hence the objective function based methods can not be used for the static analysis. But there is a variety of methods that can be utilized for this purpose, e.g., heuristic clustering algorithms and genetic algorithms.

Originally, a modified fuzzy clustering algorithm from [Höppner and Runkler, 1999] was proposed in [Staron *et al.*, 2007]. In this paper we implement an evolutionary-based algorithm called *Multi-objective Prototype Optimization with Evolved iMprovement Steps* (mPOEMS) for this purpose. mPOEMS uses multiple objectives as quality measures of the sought optimal clustering. The concept of multiple optimization objectives is well suited for the clustering problem at hand. Besides the minimization of the inter-cluster communication, we can define other aspects that disqualify false optimal solutions that would be considered good otherwise, discussed in section 2.1.

The paper is organized as follows. Section 2 defines the problem of static analysis of agent clustering and involved optimization criteria. Section 2 describes modified fuzzy clustering algorithm used for the static analysis in [Staron *et al.*, 2007]. Section 4 and 5 introduces multi-objective optimization and describes the proposed approach, respectively. Experimental results on a real MAS application are presented in Section 6, and Section 7 concludes the paper.

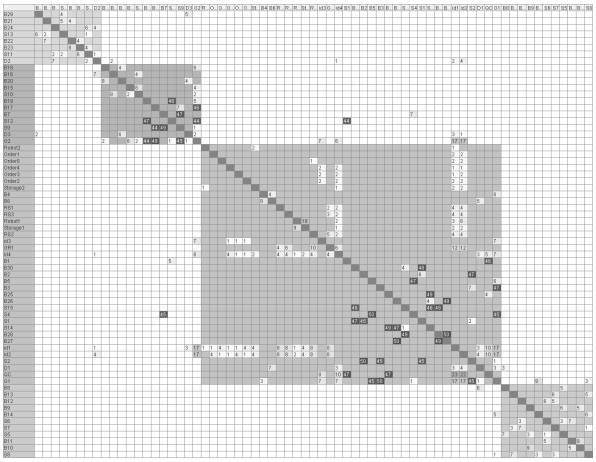

Figure 1: Visualization of static agent clustering in Java Sniffer

2 Static Analysis of Agent Clustering

Static clustering computed from messages is visualized in the Java Sniffer as a table having the agent names listed in both column and row headers. The number of messages sent and received can be visualized either separately or combined. If they are combined the table becomes symmetric along its main diagonal. If they are treated separately, each row contains the number of messages that the agent in the given row sent to agents identified by the columns. The color saturation of the background under each number in the table is proportional to its value for better visibility of higher numbers. As a result of the static agent clustering analysis, agents are reordered in such a way that agents belonging to the first cluster are placed at the beginning, agents from the second cluster follow, and so on. Clusters are depicted as squares along the diagonal of the table. Since the criterion for static agent clustering is to maximize communication within clusters, the sum of the numbers in the corresponding squares is maximized and sum of the numbers outside of the squares is minimized.

However, this criterion alone would not guarantee generating truly optimal solutions. One can imagine that taking into account only this objective might result in finding possibly very unbalanced clusterings that would be constituted of $k-1$ clusters of size 1 and one cluster of size $n-k+1$, where n is the number of agents and k is the specified number of clusters. Solutions like this would be optimal with respect to the maximal internal communication if the cluster sizes were not important. In our case, the goal is to find such a clustering that would simultaneously (i) maximize the communication within clusters (and minimize inter-cluster communication) and (ii) will disqualify those unbalanced solutions. So the task can be considered a multi-objective optimization problem.

2.1 Objectives

In this work we consider the following objectives:

- **Total internal communication**. First, internal communication is calculated as the total communication among agents that belong to the same cluster (computation unit) summed up over all clusters. Then, the objective is calculated as the ratio of the total internal communication to the overall communication observed among all agents. This objective is to be maximized.
- **Clusters sizing**. This objective relates to the need for solutions that consist of reasonably sized clusters. In fact, we do not know in advance what sizes are the best for particular data. So, the objective is defined such that the solutions of equally sized clusters are preferred. It calculates a maximal difference of a size of the biggest and smallest cluster in the solution that is to be minimized.

3 Modified fuzzy clustering algorithm

The data set is normalized such that all variables are contained in the unit interval: $x_{jk} \in [0,1]$. The clusters are represented by Gaussian membership functions of the following form:

$$\mu_{ik} = \exp\left(-\sum_{j=1}^{\dim}\left(\frac{m_{ji}-x_{jk}}{2\sigma_{ji}}\right)^2\right) \qquad (1)$$

where μ_{ik} is the membership degree of data point $\mathbf{x}_k$ in cluster i, $\mathbf{m}_i$ is the ith cluster center and σ_{ji} is the cluster width along the jth feature. It is assumed that the data samples are vectors in a metric space. If this is not the case, a suitable dissimilarity measure can be defined instead of the distance.

The algorithm processes the input patterns one by one. Initially, the first input sample defines the first cluster center and the initial width of this cluster is set to some default value (a user-defined parameter). For each pattern, the algorithm then determines whether it belongs to a sufficient degree to some existing cluster. If this is the case, the pattern is added to that

---

Fast clustering algorithm

Set the parameters $\mu_{\min}$, α_c, α_e, $\sigma_{\min}$, $\sigma_{\max}$ and create the first cluster: $\mathbf{m}_1 := \mathbf{x}_1$, $\boldsymbol{\sigma}_1 := \boldsymbol{\sigma}_{\text{init}}$, $N_1 := 1$, $c := 1$.

Repeat for $k = 2, \ldots, N$

Compute μ_{ik}, $\forall i$ by using (1) and find the closest cluster: $I = \arg\max_i \mu_{ik}$.

if $\mu_{Ik} \geq \mu_{\min}$, add the input pattern to cluster I and modify the cluster's parameters:

$\mathbf{m}_I := \frac{(\mathbf{m}_I N_I)+\mathbf{x}_k}{N_I+1}$, $N_I := N_I + 1$,

$$\sigma_{jI} := \begin{cases} \min(\alpha_e \sigma_{jI}, \sigma_{\max}) \\ \quad \text{if } \exp[-(m_{jk}-x_{jk})^2/{\sigma_{jI}}^2] > 0.5 \\ \max(\alpha_c \sigma_{jI}, \sigma_{\min}) \quad \text{otherwise} \end{cases}$$

for $j = 1, \ldots, \dim$

else create a new cluster:

$c := c+1$, $m_c := x_k$, $\sigma_c := \sigma_{\text{init}}$, $N_c = 1$

Figure 2: Modified fuzzy clustering algorithm

cluster and the cluster center and volume are changed to account for this new input pattern. Otherwise, a new cluster is created.

The threshold $\mu_{\min} \in (0, 1)$ is used to find out whether a given input pattern belongs to an existing cluster. Along with the maximal width $\sigma_{\max}$ and the contraction/expansion parameter α, it indirectly determines the number of clusters. The larger $\mu_{\min}$ and smaller $\sigma_{\max}$ are chosen, the more clusters will be created. The default setting for σ_{init} is $0.75 \cdot \sigma_{\max}$.

4 Multiobjective Optimizations

In many real-world optimization problems a solution is sought that is optimal with respect to multiple optimization criteria (often conflicting with each other). Having multiple objectives specifying quality measures of solutions result in no single optimal solution. Instead there is a set of alternative solutions that are optimal in a sense that (*i*) none of them is superior to the others and (*ii*) there is no other solution in the whole search space that is superior to these optimal solutions when all objectives are considered. Thus, a good multi-objective optimization technique must be able to search for a set of optimal solutions concurrently in a single run.

For this purpose, evolutionary algorithms seem to be well suited because they evolve a population of diverse solutions in parallel. Many evolutionary-based approaches for solving multiobjective optimization problems have been proposed since the mid-1980's. The most distinguishing features are (*i*) the fitness assignment strategy for evaluating the potential solutions, (*ii*) evolutionary model with a specific selection and replacement strategy, and (*iii*) the way the diversity of the evolved population is preserved.

Perhaps the most widespread and successful are multiobjective evolutionary algorithms that use a concept of *dominance* for ranking of solutions. By definition [Deb, 2002], a solution x dominates the other solution y, if the solution x is no worse than y in all objectives and the solution x is strictly better than y in at least one objective. Naturally, if solution x dominates solution y then x is considered better than y in the context of multiobjective optimization. However, many times there are two different solutions such that neither of them can be said to be better than the other with respect to all objectives. Such solutions are called *non-dominated* solutions.

The concept of dominance can be used to divide a finite set S of solutions chosen from the search space into two non-overlapping sets, the *non-dominated set* S_1 and the *dominated set* S_2. The set S_1 contains all solutions that do not dominate each other. The set S_2, which is a complement of S_1, contains solutions that are dominated by at least one solution of S_1. If the set S is the whole feasible search space then the set S_1 is a set of optimal solutions called *Pareto-optimal* solutions and the curve formed by joining these solutions is called a Pareto-optimal front. Note that in the absence of any higher-level information, all Pareto-optimal solutions are equally important [Deb, 2002]. That is why the goal in a multiobjective optimization is to find a set of solutions that is (*i*) as close as possible to the Pareto-optimal front and (*ii*) as diverse as possible so that the solutions are uniformly distributed along the whole Pareto-optimal front.

Of the Pareto-based approaches, perhaps the most well-known are Non-dominated Sorting GA II [Deb *et al.*, 2002b] and Strength Pareto Evolutionary Algorithm 2 [Zitzler *et al.*, 2002].

```
1  generate(Prototype)
2  repeat
3    BestSequence ← run_EA(Prototype)
4    Candidate ← apply(BestSequence,Prototype)
5    if(Candidate is_better_than Prototype)
6        Prototype ← Candidate
7  until(POEMS termination condition)
8  return Prototype
```

Figure 3: An outline of the single-objective POEMS algorithm

```
1  generate(SolutionBase)
2  repeat
3    Prototype ← choose_prototype(SolutionBase)
4    ActionSeqs ← MOEA(Prototype, SolutionBase)
5    NewSols ← apply_to(ActionSeqs, Prototype)
6    SolutionBase ← merge(NewSols,SolutionBase)
7  until(termination condition is fulfilled)
8  return SolutionBase
```

Figure 4: An outline of the mPOEMS algorithm

5 Proposed Approach - mPOEMS

Standard evolutionary algorithms (EAs) typically evolve a population of candidate solutions to a given problem. Each of the candidate solutions encodes a complete solution i.e. a complete set of the problem parameters in parameter optimizations, a complete schedule in the case of scheduling problems, a complete tour for the traveling salesman problem, etc. This implies, especially for large instances of the solved problem, that the EA operates with very big and complex structures.

5.1 Single Objective POEMS

In POEMS [Kubalik and Faigl, 2006], an evolutionary algorithm does not handle the solved problem as a whole. Instead, one candidate solution called the *prototype* is generated at the beginning and then it is further improved within an iterative process, where the best performing modification of the current prototype is sought using the EA in each iteration, see Figure 3.

The modifications are represented as a sequence of primitive actions/operations, defined specifically for the problem at hand. The evaluation of action sequences is based on how well/badly they modify the current prototype, which is passed as an input parameter to the EA. Moreover, sequences that do not

```
input: Prototype, SolutionBase
output: Population of evolved action sequences

1  generate(OldPop)
2  evaluate(OldPop)
3  repeat
4      NewPop ← evolutionary_cycle(OldPop)
5      evaluate(NewPop)
6      OldPop ← merge(OldPop, NewPop)
7  until(EA termination condition is fulfilled)
8  return OldPop
```

Figure 5: An outline of the multiobjective evolutionary algorithm used in mPOEMS

change the prototype at all are penalized in order to avoid generating useless trivial solutions. After the EA finishes, it is checked whether the best evolved sequence improves the current prototype or not. If an improvement is achieved, then the sequence is applied to the current prototype and the resulting solution becomes a new prototype. Otherwise the current prototype remains unchanged for the next iteration. The process of iterative prototype improvement stops when the termination condition is fulfilled.

5.2 Multiobjective POEMS

The multiobjective "Prototype Optimization with Evolved iMprovement Steps" (mPOEMS) belongs to a class of multiobjective optimization algorithms that use the concept of dominance. In this section we describe the way the set of non-dominated solutions progressing towards the Pareto-optimal set is evolved in mPOEMS. Note that in multiobjective optimization the goal is to find a set of optimal solutions (as close as possible to the Pareto-optimal set) that are as diverse in both the variable space and the objective space as possible. Thus, the main differences between mPOEMS and POEMS are that

- mPOEMS maintains a set of best solutions found so far, called a *solution base*, not just one prototype solution that is maintained in POEMS. In each iteration of mPOEMS one solution from the set of non-dominated solutions in the solution base is chosen as the prototype for which the action sequences will be evolved by the EA.
- mPOEMS uses a kind of a multiobjective EA (MOEA) based on the dominance concept, not just a simple EA. The output of the MOEA is a set of action sequences (not just one action sequence) generating new solutions that are merged with the current solution base resulting in a new version of the solution base.

Figure 4 shows the main steps of the mPOEMS algorithm. It starts with generating the initial solutions of the solution base. The size of the solution base is denoted as $SbSize$ and stays constant through the whole mPOEMS run.

The first step of the main body of the iterative process is the selection of the prototype for the current iteration. In this work, the prototype is chosen randomly from the non-dominated set of the solution base. The prototype is passed as an input parameter to the multiobjective EA, where the action sequences possibly altering the prototype towards the Pareto-optimal set are evolved. The other input parameter of MOEA is the current solution base that is used for evaluation purposes, see below. MOEA returns the final population of action sequences, which are then applied to the current prototype resulting in a set of new solutions.

The outline of the multiobjective EA used in mPOEMS is shown in Figure 5. First, it generates a starting population of action sequences of size $PopSize$. The action sequences are evaluated based on the quality of the solution that is produced by applying the given action sequences to the prototype. Then, the population of action sequences is evolved within a loop until some stopping condition is fulfilled. In the first step of the loop, a new population of action sequences is generated using standard operations of selection, crossover and mutation. The action sequences are evaluated and assigned fitness values. Finally, the new population and the old one are merged and $PopSize$ solutions of the best non-dominated fronts of that joint population constitute the resulting population.

Since we are dealing with multiobjective optimization problems, each solution is assigned multiple objective values. The evaluation procedure uses a concept of dominance between solutions in order to find a single fitness value specifying the solution quality in terms of its non-domination level. In order to have more levels of non-domination that better distinguishes solutions the evaluated solutions are merged with solutions from the solution base resulting in a temporary set of solutions S (the prototype solution is included in the set S as well). The process of calculating the level of non-dominance starts with finding the non-dominated solutions among the whole set S. These solutions belong to the first level of non-domination front and are assigned a non-domination level $ND_{level} = 1$. Then they are temporarily disregarded from the set S and the set of non-dominated solutions is sought among the remaining solutions. These are the solutions of the second level of non-domination and are assigned a non-domination level $ND_{level} = 2$. The process goes on until there is no solution left in S, i.e. every solution has assigned its ND_{level} value. In the second phase of the evaluation procedure, the evaluated solutions are assigned their fitness value. Solutions that belong to a better than or the same level of non-domination as the prototype solution are assigned a fitness value equal to their ND_{level} value. Solutions with the ND_{level} higher than the prototype solution are assigned a fitness value equal to $ND_{level} + 0.5 * P_D$, where P_D is 1 iff the given solution is dominated by the prototype, and 0 otherwise. Note that the smaller fitness the better solution. So, the selection pressure is towards the solutions that

1. belong to a better non-domination front than the prototype, if possible, and
2. are not dominated by the prototype solution.

New solutions produced by action sequences evolved by the MOEA are merged with the current solution base resulting in the new solution base. This is done in two steps. First, the new solutions as well as the solutions of the solution base are classified into different non-domination fronts and ranked according to their level of non-domination. Then, the new solutions are taken one by one, starting from the best non-domination front to the worst one, and for each of them the solution base is traversed in an opposite order (starting from the worst non-domination front) looking for a solution that is dominated by the given new solution. If such solution exists in the solution base, it is replaced by the given new solution iff the solution base does not contain a copy of that individual yet[1]. In this way, the worst solutions of the solution base are replaced by the best new solutions while the size of the solution base is kept constant.

6 Experiments

We now present experimental results on one real MAS. Agents in this MAS are used for control of the transportation of products or discrete materials on the factory's shop floor using a network of conveyor belts [Fletcher *et al.*, 2003].

The MAST environment simulates all the components of the holonic packing cell of the Center for Distributed Automation and Control (CDAC) at the University of Cambridge, U.K. This laboratory provides a physical testbed for experiments with the Radio Frequency Identification (RFID) technology and the agile and intelligent agent-based manufacturing control, for details we refer the interested reader to [Vrba and Marik, 2006].

The analyzed MAS involves 67 agents. A matrix that combines sent and received messages is considered. The number of sought clusters is set to $k = 4$.

6.1 mPOEMS Implementation

Solution base initialization. The implementation presented in this paper used a greedy heuristic method for generating solutions of the initial solution base. This heuristic aims at maximizing the internal communication within clusters. First, k agents representing seeds of clusters are found so that the mutual communication among those seeds is minimal. Then, remaining agents are picked up one by one on a random basis and for each agent a total communication to all clusters (represented by already assigned agents) is calculated. Finally, the agent is assigned a cluster with the maximal communication.

Representation. The MOEA evolves linear chromosomes of length *MaxGenes*, where each gene represents an instance of certain action chosen from a set of elementary actions defined for the given problem. Each action has its name *action_type* and a list of parameters. Besides so called *active actions* that truly modify the prototype there is also a special type of action called *nop* (no operation). Any action with *action_type* = *nop* is interpreted as a void action with no effect on the prototype, regardless of the values of its parameters. A chromosome can contain one or more instances of the *nop* operation. This way a variable effective length of chromosomes is implemented. In this work, action sequences evolved by MOEA are composed of *nop* and just one type of active action – $change\_cluster(agentId, newCluster)$. This action changes current cluster assigned to agent $agentId$ to $newCluster$. At the beginning of each MOEA run, the action sequences are generated by random.

[1]This condition ensures that the solution base consists of unique solutions only

Table 1: Optimal values of *internal communication* (O_2) for a range of *clusters sizing* (O_1) values found by a single run of mPOEMS.

O_1	O_2	O_1	O_2	O_1	O_2	O_1	O_2
1	0.828	11	0.881	21	0.926	31	0.952
2	0.837	12	0.878	22	0.929	32	0.953
3	0.843	13	0.890	23	0.933	33	0.957
4	0.840	14	0.881	24	0.935	34	0.961
5	0.851	15	0.895	25	0.939	35	0.962
6	0.856	16	0.898	26	0.942	36	0.965
7	0.862	17	0.905	27	0.942	37	0.966
8	0.863	18	0.912	**28**	**0.944**	38	0.969
9	0.871	19	0.917	29	0.948	39	0.971
10	0.877	20	0.919	30	0.951	40	0.971

6.2 mPOEMS Setup

The following setup was used in our simulations:

- Action sequence length: 20,
- Population size: 50,
- Solution base size: 150,
- Number of generations: 15,
- Crossover rate: 0.75,
- Mutation rate: 0.25,
- Tournament selection: 3,
- Number of iterations: 150.

6.3 Results

Since EAs are stochastic algorithms, their performance must be assessed statistically. We carried out 10 independent runs of the algorithm and observed the average value of *internal communication* objective achieved for different values of *clusters sizing* objective over the 10 runs. Results are presented in Table 1.

As the reference result we use the result achieved by the modified fuzzy clustering algorithm presented in [Staron *et al.*, 2007] - it gives *internal communication* of 0.911 for *clusters sizing* = 28.

We see that mPOEMS gives significantly better results in terms of *internal communication* for many

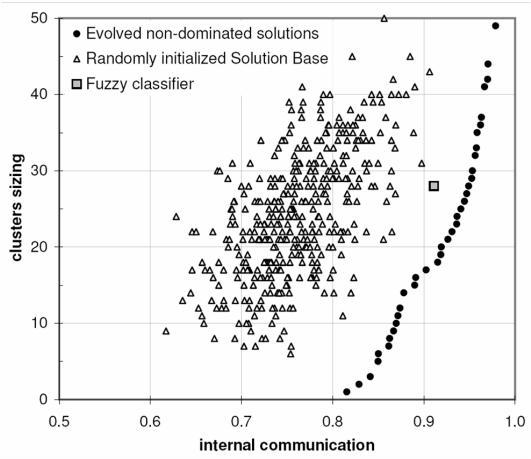

Figure 6: Illustration of one run of mPOEMS.

various *clusters sizing* values. This is due to the fact that the mPOEMS algorithm returns a set of non-dominated solutions that are spread over a range of both *clusters sizing* and *internal communication*. A general trend is the less *clusters sizing* the worse *clusters sizing*. However, the most interesting are the results achieved for *clusters sizing* ≤ 28. For *clusters sizing* $= 28$, the mPOEMS gives on average *internal communication* $= 0.949$, the best value observed within the 10 runs was *internal communication* $= 0.953$ and the worst observed value was *internal communication* $= 0.931$.

A typical mPOEMS run is illustrated in Figure 6. It shows an initial solution base of solutions created by the greedy heuristic method. Since *clusters sizing* and *internal communication* are to be minimized and maximized, respectively, the search progresses towards the lower right region of the search space. Here, solutions of the final non-dominated set sample *clusters sizing* objective within a range from 1 to 49.

7 Conclusions

This paper proposes a multiobjective iterative optimization algorithm for static agent clustering analysis within multi-agent systems. Our algorithm clearly illustrates advantages of multiobjective optimization algorithms that (i) the objectives are optimized simultaneously, and (ii) in contrast to traditional approaches, which return just one solution per run, mPOEMS algorithm returns a set of optimal solution with respect to both objectives from which one can choose the best one according to some further knowledge.

An important issue in all applications of evolutionary-based algorithms is how to incorporate problem specific knowledge into the algorithm so that the blindness of its search is maximally reduced. In this work we used just a heuristic sampling of the initial set of solutions. Further research might be focused on a proper design of the elementary actions and a proper composition of action sequences. For example, actions may be generated such that they prefer agents from bigger clusters and change their association to some smaller cluster and vice versa.

Acknowledgement

Research described in the paper has been supported by the research program No. MSM 6840770038 "Decision Making and Control for Manufacturing III" of the CTU in Prague.

References

[Deb, 2002] Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Ltd., New York, 2002

[Deb *et al.*, 2002b] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002

[Fletcher *et al.*, 2003] Fletcher, M., McFarlane, D., Lucas, A., Brusey, J., Jarvis, J.: The Cambridge Packing Cell - A Holonic Enterprise Demonstrator. Proc. of 3rd Int. / Central and Eastern European conference on Multi-Agent Systems, pp. 533-543, Czech Republic, 2003.

[Höppner and Runkler, 1999] Höppner, F. and Runkler, T.: Fuzzy cluster analysis, John Wiley , New York (1999)

[Kubalik and Faigl, 2006] Kubalik J. and Faigl J.: Iterative Prototype Optimisation with Evolved Improvement Steps. In: P. Collet, M. Tomassini, M. Ebner, A. Ekart and S. Gustafson (Eds.): Genetic Programming. Proceedings of the 9th European Conference, EuroGP 2006. Heidelberg: Springer, 2006, pp. 154–165

[Staron *et al.*, 2007] Staron J.R., Tichý P., Šindelář R., and Maturana F.: Methods to Observe the Clustering of Agents Within a Multi-Agent System. In (Mařík V. et al., eds.) Proc. of the 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2007, LNAI 4659, pp.127-136, Springer-Verlag, Heidelberg, 2007.

[Tichý *et al.*, 2006] Tichý, P., Šlechta, P., Staron, R.J., Maturana, F.P., Hall, K.H.: Multiagent Technology for Fault Tolerance and Flexible Control. IEEE Transactions on System, Man, and Cybernetics-Part C: Applications and Reviews, vol. 36, no. 5, pp.700-705, 2006.

[Vrba and Marik, 2006] Vrba, P., Mařík, V.: Simulation in agent-based control systems: MAST case study. International Journal of Manufacturing Technology and Management, vol. 8, no. 1/2/3, pp. 175-187, 2006.

[Zitzler *et al.*, 2002] Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm For Multiobjective Optimization, In: Evolutionary Methods for Design, Optimisation, and Control, Barcelona, Spain, pp. 19-26, 2002.