A Genetic Approach for Long Term Virtual Organization Distribution

Víctor Sánchez-Anguix
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia,
Camí de Vera s/n
Valencia, 46022, Spain
sanguix@dsic.upv.es

Soledad Valero
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia,
Camí de Vera s/n
Valencia, 46022, Spain
svalero@dsic.upv.es

Ana García-Fornes
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia,
Camí de Vera s/n
Valencia, 46022, Spain
agarcia@dsic.upv.es

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

An agent-based Virtual Organization is a complex entity where dynamic collections of agents agree to share resources in order to accomplish a global goal or offer a complex service. An important problem for the performance of the Virtual Organization is the distribution of the agents across the computational resources. The final distribution should provide a good load balancing for the organization. In this article, a genetic algorithm is applied to calculate a proper distribution across hosts in an agent-based Virtual Organization. Additionally, an abstract multi-agent system architecture which provides infrastructure for Virtual Organization distribution is introduced. The developed genetic solution employs an elitist crossover operator where one of the children inherits the most promising genetic material from the parents with higher probability. In order to validate the genetic proposal, the designed genetic algorithm has been successfully compared to several heuristics in different scenarios.

Keywords: Virtual organizations; genetic algorithms; multi-agent systems.

1. Introduction

A Virtual Organization (VO) is a complex entity where dynamic collections of individuals and institutions agree to share resources (software services, computational resources, etc.).[1,2,3,4] Some works have already stated that Multi-agent systems (MAS) and agent organizations are one of the possible technologies for the implementation of VOs.[1,2,4] MAS are distributed systems where *software agents* are executed. These agents are defined as social, reactive and proactive according to its user's goals.[5]

Computational performance is one of the key factors in the VO success.[3] For instance, a poor optimized VO may offer services with response times that are unacceptable for the agents/users that request them. Since service consumers are likely to change service providers in a competitive environment, offering high quality services is crucial.[6,7,8,9] A proper distribution of agents across the shared resources may pursue such goal since avoiding host overload may reduce service time response. Ideally, all of the resources should be employed with similar usage rates and none of them should be overloaded. The problem of distributing workload across shared distributed computational resources is known in the literature as *load balancing*.[10,11,12] In this article, we are interested in the problem of load balancing in long term VOs. Long term/Permanent VOs usually offer complex and stable services that persist over time (*e.g.*, travel services, e-commerce services, etc.).[13,14] Although this type of VO may need to adapt itself to accommodate new agents/resources, the initial agent distribution may remain as a basis for a long time.

In MAS, it is natural for agents to have different levels of confidence or untrust in their partners. Some of the partners may have not interacted enough to be considered as "trusted", or they may have experienced some problems that have hindered their relationship (*e.g.*, payment difficulties).[15,16,17,18] As a result of these experiences, some resource owners may be reluctant to share computational resources with certain partners. Additionally, open MAS make also reasonable to assume that different agents may have different software requirements (operating system, software libraries, etc.).[19] Consequently, these requirements are to be taken into account when distributing agents across shared resources.

Thus, the problem of load balancing in Virtual Organization consists of assigning resources (hosts) to agents. Due to the fact that the system may become large-scaled, the problem may become a large combinatorial problem where the load balancing must be optimized. Classic approaches, like A and A* algorithms,[20,21,22] are not computationally efficient enough to deal with such problems. However, Genetic algorithms (GAs) are algorithms based on the process of evolution in nature.[23,24,25,26] They have proven to be especially interesting in solving optimization problems in large search spaces, and in situations where very little information about the domain is available.[27,28,29,30] GAs are based in a population of solutions which iteratively converges towards high quality solutions. Several operations (crossover, mutation and selection) are performed over the population in order to produce the next generation (iteration) of solutions.

In this article, a solution for agent distribution across hosts in an agent-based VO is proposed. This approach is based on a Genetic Algorithm (GA) that is meant to be applied just after the VO formation phase in order to provide a basis distribution for a long term VO. The proposed GA takes into account trust issues, software requirements of the agents, and load balancing. Our goal is to give a proper distribution of agents in a long term VO according to the load balance and trust issues. More specifically, host overload is avoided while providing agents with the software requirements needed by agents (*e.g.*, libraries). Two different aspects are optimized with our proposed solution: load balance and untrust level in the VO. This proposal is based on our previous work.[31] The main difference between this present work and our previous work is our proposed abstract MAS architecture, the methodology employed for the experiments and parameter tuning, and new experiments which show the performance of our proposed architecture in different scenarios (*e.g.*, fitness evolution through time, different importance for load and untrust levels, etc.).

The remainder of this work is organized as follows. First, in Sec. 2, we describe some related work in the area of grid load balancing and multi-agent management. In Sec. 3, a formal definition of the problem is given. After that, an abstract architecture which supports VO load balancing is described in Sec. 4. The design of the GA is thoroughly described in Sec. 5. In Sec. 6, experiments that were carried out to validate the proposed GA are detailed. It includes how a good set of parameters were found, a comparison with different heuristics which solve the same problem, and an experiment where different importance is given to the untrust level and the load level. Finally, some lines of future work and conclusions are pointed out.

2. Related Work

To the best of our knowledge, the problem of VO distribution in Multi-agent Systems has not been addressed before. However, there are three related areas: Multi-agent systems management, agent-based virtual organization formation, and grid computing load balancing by means of genetic algorithms.

Multi-agent system management consists of the visualization and control of the physical and virtual resources located in a multi-agent system. [32] Several MAS platforms include management tools that allow agents to be distributed across platforms hosts. [19,32,33,34,35] Sanchez-Anguix, *et al.*,[32] propose a multi-agent management architecture based on the Magentix platform.[36] It provides a graphical tool which allows several management operations such as MAS visualization, agent creation, agent removal, and MAS' history visualization. However, the type of management is not automatic since it is carried out by a human operator. On top of that, trust issues are very restrictive since remote operations require SSH username and password. JADE and JADEX provide graphical tools which allow management actions to be carried out by a human operator.[34,35] Nevertheless, its main aim is to provide developer tools instead of management tools, since remote actions can be performed without any kind of security. Aglets offers mobility services to agents, but as far as we know, trust and load balancing are not taken into account.[33] In Giampapa, *et al.*, a graphical interface is presented which helps human administrators to launch agents based on their software requirements and the requirements offered by the different platform hosts.[19] In that sense, the idea is similar to the way we propose to handle agent requirements. However, these operations are launched by a human operator and they do not seek to automatically balance workload on hosts.

One of the most important issues regarding VO's is how they are formed. Partners should be selected according to several criteria such as the services they offer, service quality, which computational resources they provide, trust, and so forth. Norman, *et al.*, propose methods for the rapid formation of agent-based VO according to the required service requested by an agent.[4] The solution is based on a constraint satisfaction problem solver which is used by a service provider agent in order to determine whether it can provide the full service, its current VO can provide the service, or it needs to form a VO to provide the service. Hoogendoorn, *et al.*, propose a negotiation framework for VO formation.[37] Customer agents request for a set of tasks that need to be completed in a specific time interval and specific order. The system gathers bids from service provider agents regarding their preferences to perform the requested tasks and allocates them according to this information. Our proposed approach comes naturally after the process of VO formation. However, our approach focuses on VOs where computing resources may be pooled for different agents to be used, which is not considered in the previous works.

Grid load balancing is a close research area to our proposed work.[38,39,40] However, there is a main difference between both which resides in the nature of both types of systems. In grid systems, load balancing is usually achieved by distributing tasks among the different hosts which are part of the grid system. Tasks may be *one-shot* processes, whereas agents perform several tasks during their lifetime in the system, which may last until the end of the system's execution (*e.g.*, a sell service in a *e*-commerce application). Thus, in grid systems, time response for tasks is usually the object of optimization. Given agents' characteristics, it may be more reasonable to provide balanced resource utilization in a MAS setting. Therefore, system optimization actions such as load balancing have slightly different goals in grid and MAS systems. Regarding grid load balancing works, Di Martino studied the use of a GA whose aim is task distribution in a Grid environment. [38] In this work, it is assumed that different tasks have different constraints which affect where they can be executed. The GA takes these constraints into account and looks for a correct task allocation. However, trust issues are not taken into account by the GA. Later, Cao, *et al.*, used a GA to schedule different tasks in a local grid environment.[39] A local grid is a cluster of workstations. Therefore, the GA is only applied at a local level. This work also ignores trust and software requirements. Finally, Mello, *et al.*, designed a GA to distribute tasks in a grid environment.[40] Each application is divided into different tasks that are to be distributed between the machines of a neighborhood. Its main goal is to make a good initial distribution of the tasks throughout the neighborhood. The GA takes into account different performance measures such as CPU, memory usage, hard disk access and so forth. Nevertheless, it assumes that machines offer the same software requirements and trust issues are not studied. The present work is different from related grid approaches in the sense that it applies the novel idea of resource distribution in agent-based VO's. Additionally, it takes into account software requirements of the agents and trust issues.

3. Problem Definition

An agent-based VO is composed of a set of agents and hosts that belong to different entities. The problem of distribution in a VO consists of assigning agents to proper hosts for execution. However, agents cannot be executed in every host. Their software requirements, or more specifically software libraries, must be provided by the host in which they are to be executed. Furthermore, agents must be distributed in such a way that the hosts are not overloaded. Ideally, the load should be equal in all of the hosts that are part of the VO. Moreover, since agents represent different entities and hosts may be owned by different entities, there may be certain untrustness to share computational resources with other partners. Ideally, agents should be distributed in a way whereby they minimize the untrust generated.

A Virtual Organization can be seen as a tuple $VO = (H, A, R, \Gamma, \Upsilon, \delta, \mu)$ where: H is the set of hosts resources that are shared; A is the set of agents that are part of the organization; R is the set of software requirements that the agents need in order to be executed; $\Gamma : H \times A \to 0 \vee 1$ is a function that indicates if an agent is being executed in a node host; $\Upsilon : H \times A \to x, x \in \Re \quad \wedge \quad x \in [0, 1]$ is a function that measures the level of untrust regarding sharing the host resource with a specific agent; $\delta_h : H \to \Re$ represents the load that the node host is capable of handling; $\delta_a : A \to \Re$ represents the average load that the agent generates; $\mu_h : H \to R' \ \wedge R' \subseteq R$ is the set of software requirements that the node host offers and $\mu_a : A \to R' \ \wedge R' \subseteq R$ is the set of software requirements that an agent needs in order to be executed. In this work, the load generated by an agent concerns the use of the CPU (*e.g.*, MFLOPS, CPU usage). This information can be provided by the designer of the agent.

3.1. *Constraints*

When distributing agents across host resources, it is important to satisfy two different types of constraints. The first type of constraint is related to the workload level of a host. The maximum load provided by a host should not be surpassed by the total load generated by the agents executed in that host. The formal definition of this constraint can be found in Eq. (1).

$$\delta_h(h) \geq \sum_{a \in A} \Gamma(h, a) \ \times \ \delta_a(a) \ . \tag{1}$$

The second type of constraint relates to the software requirements offered by a host and the requirements needed by the agents executed in that host. One agent can be executed in a specific host if its software requirements are provided by this host. The formalization of this constraint can be found in Eq. (2).

$$\forall a, h \quad \Gamma(h,a) = 1 \rightarrow \mu_a(a) \subseteq \mu_h(h) \; . \tag{2}$$

3.2. *Optimality measures*

In this case, there are two different measures that may describe the quality of a solution. First, it is appropriate that the usage rates of the hosts that are part of the system are balanced. Second, it is also important that the untrust generated in the VO is minimized. This is accomplished by distributing agents in hosts whose owners are more willing to share resources with them.

The global untrust level in a VO can be defined as a social welfare function that must be minimized in order to provide distributions where entities share resources with partners they trust. It can be formally defined as stated in Eq. (3). It is the average of the untrust generated by the agents in their corresponding host.

$$\Theta = \frac{\sum_{a\in A}\sum_{h\in H}\Gamma(h,a)\times\Upsilon(h,a)}{|A|} \; . \tag{3}$$

The other optimality measure relates to the load balance. The load x_h in a machine h can be defined as the load generated by the agents divided by the load supported by the host h.

$$x_h = \frac{\sum_{a\in A}\Gamma(h,a)\times\delta_a(a)}{\delta_h(h)} \; . \tag{4}$$

The average load of the system $\bar{x}$ is defined as the average load of the hosts that compose the system. We also define d as the standard deviation for this distribution.

$$\bar{x} = \frac{\sum_{h\in H} x_h}{|H|} \; . \tag{5}$$

$$d = \sqrt{\frac{\sum_{h\in H}(\bar{x}-x_h)^2}{|H|}} \; . \tag{6}$$

Finally, the global load balance is defined as $\beta = 1 - \frac{d}{\bar{x}}$. The best load balance is obtained when $\frac{d}{\bar{x}} = 0$ and consequently $\beta = 1$. The expression $\frac{d}{\bar{x}}$ is 0 when $d = 0$, therefore all of the hosts have the same level of load.

4. Abstract Multi-Agent System Architecture for Virtual Organization Load Balancing

Virtual Organization load balancing requires infrastructures which provide support for different tasks such as agent migration, agent organization management, resource monitoring, and so forth. In this section we provide an abstract architecture which is needed for agent-based virtual organization load balancing. The designed abstract architecture is aimed to work on top of MAS platforms for open environments.

These platforms are designed to support and enable the deployment of VOs in open MAS, where resources (hosts, agents, agent organizations, etc.) are added and deleted during runtime.[41] More specifically, their main goal is providing support for complex agent societies where agents may be grouped in agent-based VOs which cooperate with others by means of negotiation/argumentation, are governed by norms, and provide complex services.

A MAS platform may be composed by different distributed hosts executing a platform kernel. Next, we define which services would be needed in a MAS platform in order to provide proper VO load balancing:

- **Agent Management Service (AMS):** This service registers which agents are being executed in the MAS, their identity (owner), their load profile (average load in MFLOPS), and their address (host where it will execute). Whenever a new agent enters the system, it is registered in the AMS along its identity, and its address. Optionally, it may include its load profile.
- **Load Statistics Service (LSS):** It monitors the available computing capability (MFLOPS). Additionally, it registers the load produced by each agent (MFLOPS) that is being executed in the machine. This information may be requested by any system service.
- **Organizational Support:** It allows agents to form and manage their agent organizations. It is divided into two different services.
  - *Organization Management System (OMS):* This service manages VOs, takes control of the underlying VO structure, manages the roles played by the agents and the norms that govern the VO.
  - *Service Facilitator (SF):* It allows agents/VOs to register and search services that may be invoked by other agents/VOs.
- **Migration Support Service (MSS):** This service transfers agents from one host to another in a secure way.

Once the abstract architecture has been described, it is necessary to determine how these services will interact with each other in order to provide the load balancing service. Whenever a VO is to be formed, the proposed abstract VO architecture should behave as follows:

(1) The participant agents and hosts (those that have agreed to share resources for the VO) request the OMS to register a new VO, including its role structure and norms.
(2) The OMS service asks for a participation confirmation for every resource involved in the VO.
(3) Once a confirmation has been received from every resource, the VO is registered. Then, it is necessary to calculate a proper agent distribution. The OMS opens a *Borda Voting* process where participant hosts privately rank which agents they prefer to execute based on their experiences.[42,43]

(4) It also requests the AMS and LSS information regarding the load capabilities and needs of participant hosts and agents.
(5) When this information has been provided, the OMS runs a local service which executes the GA proposed in this article. The response of this service is a proper distribution of the agents across the hosts.
(6) Finally, the OMS requests involved MSS to transfer agents where it is required.

This abstract architecture is currently being developed extending the THOMAS framework,[2,44] which can run on top of JADE and Magentix agent platforms.[34,36] THOMAS provides support for virtual organizations in open multi-agent systems, managing agent registration (AMS), managing agent organizations (OMS), and publishing agent services (SF).

Once the abstract architecture has been described, the genetic algorithm executed by the local service in the OMS is thoroughly described.

5. Genetic Algorithm Design

A Genetic Algorithm is proposed to solve the addressed problem. In this section, an explanation of its design is given. More specifically, some of the aspects explained are: chromosome and phenotype representation, initial population generation, fitness function and genetic operators (crossover, mutation and selection). The general schema of the proposed genetic algorithm can be found in Alg. 2. It should be pointed out that the design of this GA has been adopted to tackle the problem of the initial distribution of the agents. For the dynamics adaptation of the initial distribution, other types of designs or techniques would be more adequate.

5.1. *Chromosome and phenotype representation*

On the one hand, agents can be identified as integers that range from 1 to $|A|$, where $|A|$ is the total number of agents to be distributed. On the other hand, hosts are identified with integers that range from 1 to $|H|$, where $|H|$ is the total number of machines available. Each individual of the population is represented as an integer vector. Indexes of the vector represent the agent, whereas the content of a specific index position represents the host where the agent is assigned to be executed. A chromosome example of the proposed representation can be found in Fig. 1.

5.2. *Initial population generation*

Several initial populations of 512 individuals are generated in a random way. The most diverse population (higher population variance) is selected as the starting point for the GA. Furthermore, the solutions generated were assured to satisfy the constraints described in the previous section.

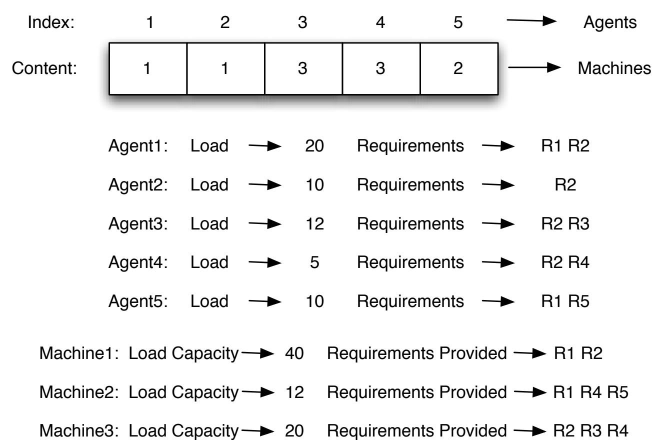

Index: 1 2 3 4 5 → Agents
Content: 1 1 3 3 2 → Machines
Agent1: Load → 20 Requirements → R1 R2
Agent2: Load → 10 Requirements → R2
Agent3: Load → 12 Requirements → R2 R3
Agent4: Load → 5 Requirements → R2 R4
Agent5: Load → 10 Requirements → R1 R5
Machine1: Load Capacity → 40 Requirements Provided → R1 R2
Machine2: Load Capacity → 12 Requirements Provided → R1 R4 R5
Machine3: Load Capacity → 20 Requirements Provided → R2 R3 R4

Fig. 1. This figure shows an example of agent distribution across the hosts of the VO. There are five different agents and three different hosts where agents can be assigned. The content of each chromosome position indicates a host where the agent, represented as a chromosome index, is to be executed. It must be highlighted that each host provides the software requirements needed by the agents that are planned to be executed in that specific host. Additionally, the load capacity of a specific host is not surpassed by the load generated by the agents that are to be executed there.

5.3. *Fitness function*

The designed GA's goal is to reduce the untrust level and obtain the best possible load balancing. The best load balancing results are obtained when $\beta = 1$. This is equivalent to minimizing the $\frac{d}{\bar{x}}$ term. The employed fitness function takes into account untrust and $\frac{d}{\bar{x}}$, trying to minimize the value of Eq. (7).

$$f = w_{load} \times \frac{d}{\bar{x}} + w_{untrust} \times \Theta \; . \tag{7}$$

5.4. *Crossover*

The developed crossover operator follows an *elitist crossover strategy*. The approximation is based on k-point crossover with two parents and two children.[26,45] A number of k+1 segments are determined by the k points selected in the chromosome. In classic k-point crossover, one child inherits the segment of one of his parents randomly, whereas the other child inherits the other parent's segment. However, the developed strategy assigns one of the children as the preferred one. The segments are evaluated by a subfitness function and the most promising segment is inherited by the preferred child with a probability of p_{good}. The other child inherits the less promising segment

One of the keys is the selection of the subfitness function. It is not necessarily related to the global fitness function, although it should measure how promising the segment is for the final genetic material. Equations 8, 9 and 10 describe the selected subfitness function.

$$\Theta_v = \frac{\sum_{l=i}^{j} \Upsilon(v_l, a_l)}{|v|} \ . \tag{8}$$

$$x_v = \frac{\sum_{l=i}^{j} \frac{\delta_a(a_l)}{\delta_h(v_l)}}{|v|} \ . \tag{9}$$

$$f_v = w_{load} \times x_v + w_{untrust} \times \Theta_v \ . \tag{10}$$

where v denotes a segment which goes from position i to position j, Θ_v relates to the average untrust of the segment, and x_v is the average load of the segment. Both values are combined linearly in the segment subfitness function f_v. It must be remarked that parents and children are incorporated into the candidate solution pool for the next generation. A general schema for the crossover operator can be found in Algorithm 1.

One of the benefits of this crossover operator is that its parameters allow us to be more or less elitist according to the problem at hand (governed by p_{good}). Additionally, the k parameter also allows us to produce as many segments as needed depending on the type of problem which is faced.

5.5. *Mutation*

The mutation is performed by randomly changing the assignation of an agent to a host.[26,46] The operation is governed by two different parameters: p_{mut} and $average_{mut}$. The first one relates to the probability of an individual being mutated, producing a new child. The $average_{mut}$ parameter is the average number of phenotypes to be mutated from an individual selected for mutation. The number of phenotypes mutated in an individual is lower or equal to $average_{mut}$ and is selected randomly. The parent and the mutated child are added to the candidate solution pool for the next generation.

5.6. *Selection*

The selection operator is applied twice in our genetic approach. Firstly, it is employed in order to determine which individuals the crossover operation is applied on. Secondly, it is used in order to select which individuals are part of the next generation after the crossover and mutation phase. The selection method chosen is the ranking selection,[26,47,48] which assigns an individual selection probability that is equal to:

$$p_i = \frac{1}{N} \times \left(min + \frac{(max - min) \times (i-1)}{N-1} \right) \ . \tag{11}$$

where N is the population size, i is the position of the individual in the ordered population and max and min are two parameters that determine how probable it is for the best and the worst individual to be selected. More specifically, the

Algorithm 1 Algorithm for the proposed elitist crossover operator.

```
$p_1$ : The first parent     $p_2$ : The second parent
$c_1$ : The most preferred child     $c_2$: The second child
$k$ :  Number of crossover points
$p_{good}$ : Probability of inheriting the most promising segment by $c_1$

/*Calculate k+1 segments for both parents*/
$S$ =Partition($p_1$,$p_2$,$k$)
$c_1 = \emptyset$
$c_2 = \emptyset$

For each segment limits $s$ in $S$
    $f_1$=SubFitness($p_1(s)$)
    $f_2$=SubFitness($p_2(s)$)
    If $f_1 \leq f_2$
        If Random(0,1) $\leq \frac{p_{good}}{100}$
            $c_1$ =Append($c_1$,$p_1(s)$)
            $c_2$ =Append($c_2$,$p_2(s)$)
        Else
            $c_1$ =Append($c_1$,$p_2(s)$)
            $c_2$ =Append($c_2$,$p_1(s)$)
        End
    Else
        If Random(0,1) $\leq \frac{p_{good}}{100}$
            $c_1$ =Append($c_1$,$p_2(s)$)
            $c_2$ =Append($c_2$,$p_1(s)$)
        Else
            $c_1$ =Append($c_1$,$p_1(s)$)
            $c_2$ =Append($c_2$,$p_2(s)$)
        End
    End
End

Return $c_1, c_2$
```

probability that the best individual is selected is $p_N = \frac{max}{N}$ and the probability that the worst individual is selected is $p_1 = \frac{min}{N}$. The number of individuals to be selected for crossover operation is governed by the parameter p_{select}, whereas the number of individuals to be selected as the next generation is equal to the maximum population $|P_{max}|$.

5.7. *Stop criteria*

The proposed genetic algorithm continues its iterative process until one of the following two criteria has been fulfilled: (i) the best fitness has not improved in 10 generations; (ii) a total computation time of t_n seconds has been exceeded since the beginning of the GA.

Algorithm 2 General schema of the proposed genetic algorithm.

```
$P$ : Population of individuals      $P_{cr}$ : Population selected for crossover
$P_{new}$ : New individuals          $n$ : Generations since last fitness improvement
$t$ : Current execution time         $bi$ : Best individual

Initialize $P$
$n = 0$
$bi =$BestIndividual($P$)

Do
    /*Crossover Phase*/
    Sort($P$)
    $P_{cr} =$RankingSelection($P$,$|P| \times \frac{p_{select}}{100}$)
    Shuffle($P_{cr}$)
    For $i = 1$ until $i = |P_{cr}| - 1$
        $s =$Crossover($P_{cr}(i)$,$P_{cr}(i+1)$)
        $P_{new} = P_{new} + s$
        $i = i + 1$
    End

    /*Mutation Phase*/
    For $i = 1$ until $i = |P|$
        If Random(0,1) $\leq \frac{p_{mut}}{100}$
            $s =$Mutation($P(i)$)
            $P_{new} = P_{new} + s$
            $i = i + 1$
        End
    End

    /*Update population*/
    $P = P + P_{new}$
    $P =$RankingSelection($P$,$|P_{max}|$)

    /*Update stop criteria variables*/
    If Fitness($bi$) $\leq$ Fitness(BestIndividual($P$))
        $n = n + 1$
    Else
        $bi =$BestIndividual($P$)
        $n = 0$
    End
While $n \leq 10 \wedge t < t_n$

Return $bi$
```

6. Experiments and Results

In this section, the design of the experiments and their results are described. It must be highlighted that the experiments were designed to be carried out in a simulation environment where different problem instances can be generated. These instances are fed directly to the GA. Firstly, experiments that aim to find a good set of values for the different parameters of the GA are described and analyzed (p_{mut}, $average_{mut}$, p_{select}, p_{good}, k, and $|P_{max}|$). Secondly, some experiments where the designed approach is compared to other methods that solve the same problem are described and analyzed. In these two experiments, the same importance was given to the untrust and load goals ($w_{load} = 0.5$,$w_{untrust} = 0.5$). Lastly, we test our proposed method in scenarios where it may be more important to optimizer one of the goals over the other (different values for w_{load} and $w_{untrust}$).

6.1. *Tuning parameters*

The first task consists of tuning the parameters of our proposed GA in order to find a good set of values to be used during the simulations. It must be highlighted that for these first experiments, the same weight was given to the load balance and the untrust level ($w_{load} = w_{untrust} = 0.5$). The methodology followed to compare several parameter configurations are confidence intervals[a]. When confidence intervals overlap for several parameter configurations, we can consider that the results obtained for those parameter configurations are statistically equivalent. In that case, we may employ $\frac{fitness}{time}$ [b] to determine which parameter configuration should be used. When confidence intervals overlap, a higher $\frac{fitness}{time}$ value implies that the parameter configuration is able to obtain a statistically equivalent fitness in less time. Thus, that configuration is preferred from a computational perspective. If two confidences intervals do not overlap, we select the parameter configuration which is better for the fitness.

6.1.1. *Mutation parameters*

The first set of experiments was related to parameters concerning mutation. It was necessary to set the probability of an individual being mutated p_{mut} and the average number of attributes to be mutated $average_{mut}$. Values tested for p_{mut} were 2, 5, 8 and 10% whereas values tested for $average_{mut}$ were 3, 5 and 8%. The rest of parameters were set to: $p_{good} = 50\%$, $p_{select} = 50\%$, $k = 499$, $|P_{max}| = 4096$, $t_n = 300$ seconds. The problem used as test during the parameter adjustment phase was a problem instance of 500 agents that need to be assigned to 70 different hosts. Each experiment was repeated 10 times and information about the fitness of the best individual was gathered.

[a]Confidence intervals are calculated for a confidence level of $95\%(\alpha = 0.05)$.
[b]Since we are minimizing the fitness, we employ $\frac{1-fitness}{time}$.

The results show the confidence intervals for solution fitness in Table 1. The different parameter configurations were compared according to their confidence intervals in order to assess if there were statistical differences between them. Additionally, Table 2 shows Fitness/Time results in order to select the best configuration in case of no statistical differences.

Table 1. Tests on mutation parameters: Results Fitness*100 confidence intervals (95%).

		$average_{mut}$		
		3%	5%	8%
p_{mut}	2%	[21.74-26.71]	[21.28-27.77]	[20.22-29.19]
	5%	[22.34-27.93]	[22.59-26.34]	[22.73-27.62]
	8%	[22.07-27.00]	[22.35-27.80]	[21.31-28.16]
	10%	[21.71-28.88]	[22.08-26.08]	[22.07-26.47]

Table 2. Tests on mutation parameters: Fitness*100/Time(s).

		$average_{mut}$		
		3%	5%	8%
p_{mut}	2%	1.02	1.07	1.01
	5%	1.03	1.00	1.04
	8%	0.95	0.98	1.04
	10%	0.97	0.94	0.94

It can be observed that all of the possible configurations perform in a similar way both in final fitness and also in Fitness/Time. The selected configuration was $p_{mut} = 10\%$ and $average_{mut} = 5\%$ despite the fact that it had slightly worst Fitness/Time results. Nevertheless it was the one that showed less variability and therefore it can be considered as more stable.

6.1.2. *Crossover parameters*

The parameters that affect the crossover operator are: the probability to be affected by the crossover operator (p_{select}); the number of crossover points (k); and the probability of inheritance of the best segment by the preferred child (p_{good}). Tests for finding a good set of values for p_{select} and k were performed first. Tested values were 40, 60, 80 and 100% for p_{select}; and 1, 9, 99, 199, 299, 399, and 499 for the parameter k. After those parameters were established, different values for the p_{good} parameter were checked. The previous good values for p_{select}, k, and mutation parameters were maintained. The different checked configurations for p_{good} were 50, 60, 70, 80, 90 and 100%. Each experimentation was repeated 10 times and $t_n = 300$ seconds. Information about the fitness of the best individual was gathered. The fitness confidence intervals for the different configurations can be found in Table 3. Furthermore, Fitness/Time results for the same experiment can be found in Table 4.

Table 3. Tests on crossover parameters: Fitness*100 confidence intervals (95%).

		k crossover points						
		1	9	99	199	299	399	499
p_{select}	40%	[66.53-68.22]	[45.06-51.81]	[43.95-47.32]	[34.70-43.81]	[25.76-42.17]	[19.09-30.26]	[12.98-24.03]
	60%	[66.67-67.94]	[44.55-51.78]	[45.25-47.52]	[37.20-40.17]	[28.13-33.02]	[20.55-24.72]	[12.27-19.40]
	80%	[65.75-68.66]	[45.38-49.47]	[43.61-48.04]	[37.30-38.95]	[28.16-32.53]	[19.59-25.72]	[12.87-17.22]
	100%	[66.27-68.14]	[44.49-51.82]	[43.50-48.79]	[36.69-39.22]	[28.68-32.49]	[10.33-40.84]	[13.80-18.09]

Table 4. Tests on crossover parameters: Fitness/Time(s).

		k crossover points
		499
p_{select}	60%	0.82
	80%	0.60
	100%	0.50

The experiments showed that the best results were obtained for k=499[c], the length of the chromosome. Nevertheless the solution for k=499 and $p_{select} = 40\%$ was discarded since it showed much more variability than the others. Thus our decision was made between p_{select}=60, 80 and 100%. When analyzing Fitness/Time results for these configurations, Table 4, the configuration k=499, $p_{select} = 60$ was the one selected since it provided better Fitness/Time values.

The second experimentation regarding crossover parameters had the goal of adjusting the p_{good} parameter. In this case k was set to 500 and p_{select} was set to 60%. The Mutation parameters are the same ones that were used before, thus $p_{mut} = 10\%$ and $average_{mut} = 5\%$. The results showing fitness confidence intervals can be observed in Table 5.

In this case it was clear that the best values for the p_{good} parameter were found in 80%. This is statistically different to 50, 60, 90 and 100%. Although 70 and 80% are not statistically different since there is certain interval overlapping, the value of $p_{good} = 80\%$ was preferred since it showed less variability. Moreover, it is interesting to remark that extremely elitist crossover ($p_{good} = 100\%$) and classic crossover ($p_{good} = 50\%$) produced worse results than slightly elitist strategies ranging from 60% to 90%. It was interesting to observe how the introduction of domain specific information in the crossover operation by means of subfitness functions can improve the results obtained by classic crossover operation.

[c]This indicates that the k parameter should be adjusted to the size of the chromosome minus 1.

Table 5. Tests on crossover parameters: Fitness*100 confidence intervals (95%).

		Fitness*100
p_{good}	50%	[27.41-38.38]
	60%	[19.50-31.89]
	70%	[15.21-24.02]
	80%	[13.94-17.33]
	90%	[17.92-21.79]
	100%	[29.80-33.89]

6.1.3. *Population control parameters*

The studied parameter that affects the population control is the maximum population ($|P_{max}|$). Values tested for $|P_{max}|$ were 1024, 2048, 4096, and 8192. The same methodology and good values of the parameters found in previous experimentations were maintained. All of the configurations performed similarly, thus additional time measures were taken into account. The confidence intervals for this experiment can be seen in Table 6. Moreover, the Fitness/Function values can be observed in Table 6.

Table 6. Population parameters: Fitness*100 confidence intervals (95%) and Fitness/Time(s).

		Fitness*100	Fitness/Time(s)
$\|P_{max}\|$	1024	[14.03-19.19]	1.26
	2048	[13.14-19.61]	0.81
	4096	[10.13-21.34]	0.39
	8192	[10.86-19.23]	0.15

All of the configurations performed similarly when comparing the Fitness obtained. If we take into account the Fitness/Time values it is highlighted smaller population sizes such as 1024 and 2048 obtain faster results. Since in our next experiments we want to tackle larger problems, we select 2048 as population size despite the fact that it is slower than 1024. In larger problems instances, a population of 1024 may be too small.

6.2. *Proposal evaluation*

Once a good parameterization for the GA was found, it was necessary to test the designed GA in several problem instances. Moreover, the implemented solution was also compared to several baselines. More specifically, it was compared to a grasp implementation, a GA that uses classic multicrossover[d] (k=4 and k=$\frac{|V|}{2}-1$, $p_{good} = 50\%$), and other GA that employs uniform crossover (k=$|V|-1$, $p_{good} = 50\%$). The

[d] $|V|$ is the size of the chromosome

rest of parameters of the cited GA's are common, they only differ in the parameters concerning the crossover operator. As for the goals' weights, it was set to $w_{load} = w_{untrust} = 0.5$. Following, we describe the other methods in more detail. After that, we describe the experiments carried out in order to compare all of the methods.

6.2.1. *Grasp method*

A grasp Algorithm is a metaheuristic method which basically consists of two different phases: construction phase and local search phase.[49,50] In the construction phase, a random solution for the problem is constructed. Each assignment is made randomly, providing that it does not violate the software requirement constraints. After all of the assignments have been done, the solution is tested in order to check if there are overloaded hosts. In that case, the solution is discarded and a new one is generated. If the solution satisfies all of the constraints then the local search phase starts.

The local search aims to look for better solutions in the neighborhood of the obtained solution. The fitness function used in the genetic algorithm is also employed in the grasp algorithm to check solution quality. From all of the possible assignments for an agent, those that do not violate constraints in the current global assignment, the one that improves the fitness function the most is selected. In this case each assignment is revisited in order until all agent assignments have been analyzed. The solution obtained has better or equal quality to the one obtained randomly. If the solution is better than the best solution obtained until now, then the best solution is updated with the recently produced solution. The grasp algorithm goes back to the construction phase and continues until a stop criterion is met. In this work it was decided that the grasp algorithm should stop after t_n seconds.

6.2.2. *Multicrossover genetic algorithms*

A multicrossover GA divides the parents in a number of crossover points which is greater than one.[51] Therefore, the parents have at least three segments. The segments that will be inherited by the children are usually randomly determined. It is easy to obtain a multicrossover GA with our proposed GA if $k \geq 2$ and $p_{good} = 50\%$. The rest of parameters of this GA were set to $p_{mut} = 10\%$, $average_{mut} = 5\%$, $p_{select} = 60\%$, $|P_{max}| = 2048$. Two different multicrossover GAs were used: the first one using $k = 4$, and the second one using $k = \frac{|V|}{2} - 1$.

6.2.3. *Uniform crossover*

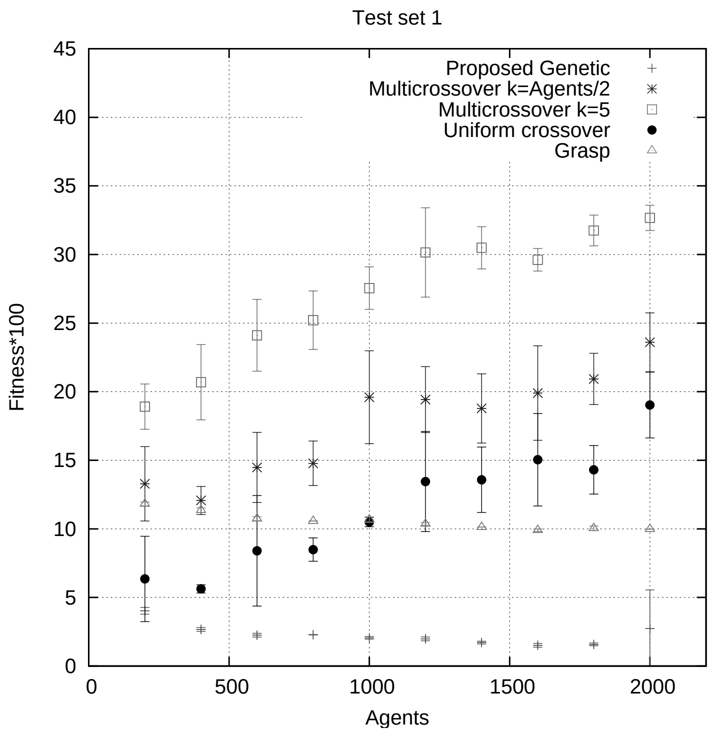
A uniform crossover GA divides the parents in as many segments as the size of the array.[52] Each child randomly inherits one of the parent's segments, whereas the other parent's segment goes to the other child. A uniform crossover GA can be obtained using our proposed GA if $k = |V| - 1$ and $p_{good} = 50\%$. The rest of parameters of this GA were set to $p_{mut} = 10\%$, $average_{mut} = 5\%$, $p_{select} = 60\%$, $|P_{max}| = 2048$.

6.2.4. *Comparison experiments*

In order to compare the different methods, two different sets of experiments were designed. In the first set of experiments, several instances of the problem were generated with different numbers of hosts and agents. The first test set was created with the number of agents ranging from 200 to 2000 and the number of hosts five times lower than the number of agents. Therefore, the problems increased the array size although they had similar characteristics. In the second set of experiments, the number of hosts remained static at 30 and the number of agents ranged from 200 to 600. Since the number of hosts was static and the number of agents gradually increased, the quality of the solution and the number of possible solutions were expected to decrease. The goal of the second set of experiments was to check that the designed GA outperformed the other methods in more complex problems. The proposed GA was configured with the best parameterization found in the previous experiments. Each experiment was repeated 10 times and the result was the fitness of the best individual (for all the methods). The maximum execution time for all of the methods was set to $t_n = \frac{|V| \times 48}{200}$ seconds, where $|V|$ is the size of the chromosome.

Figure 2 shows the results achieved in the two sets of experiments. The upper graphic shows the results for the first set, whereas the lower graphic shows the result for the second set. More specifically, the average fitness reached in every tested method and its associated confidence interval (95%) are shown in both graphics. In the first experiment, it can be appreciated that the proposed GA outperforms the other metaheuristics for every problem instance. The grasp implementation is the closest method to our proposed approach. However, our method still gets statistically better results than grasp's results. It can also be observed that the proposed GA and the grasp method are the only methods whose qualities remain unaffected as the size of the problem increases, which is also significantly important since it allows working with bigger problem instances. Additionally, it can also be appreciated that the uniform crossover GA gets worse results than our proposed approach. This fact suggests that the elitist crossover strategy we introduced helps to achieve higher quality solutions.

In the second experiment, the number of hosts remained static while the number of agents increased. Therefore, it is more difficult to properly distribute the agents among the hosts, and the quality of the solutions is expected to get worse. In fact, that is the behavior which can be observed in the right graphic. Every method reduces its performance as the number of agents increases. However, despite the fact that the problem difficulty increased, the proposed method was able to achieve statistically better results than the other methods. This result implies that the designed GA is less prone to suffer from quality reduction due to higher complexity problem instances.



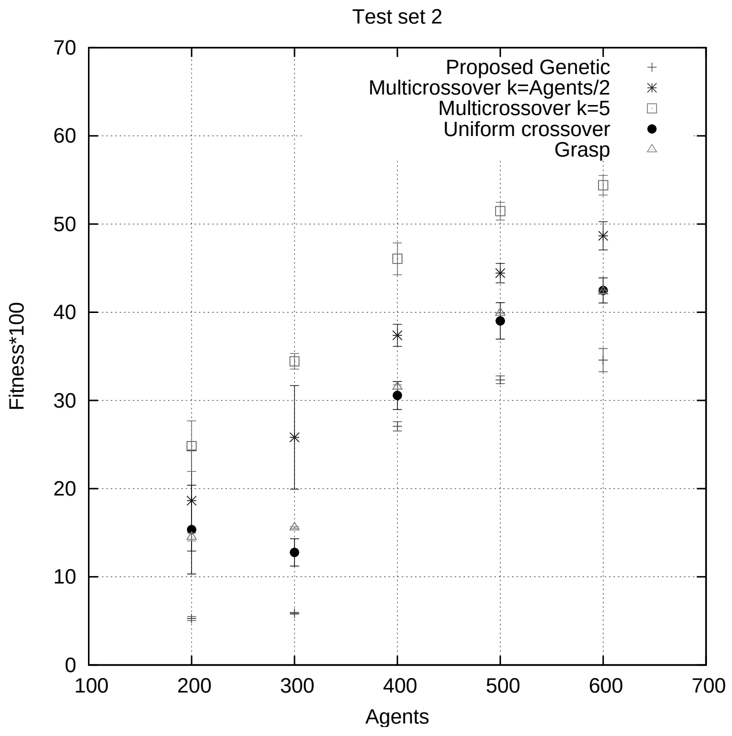


Fig. 2. The upper figure shows the results for the first set of experiments, whereas the lower figure shows the results for the second set of experiments

Additionally, we decided to analyze the fitness evolution of the proposed GA, the grasp method, and the uniform crossover method. The main goal was to determine the quality of the best solution found by each method as time increases. The number of agents was set to 300, whereas the number of hosts was set to 30. The results for this experiment can be observed in Fig. 3. The graphic implies that the grasp method is capable of finding an acceptable solution in a very few seconds. However, the quality of the best solution remains almost unchanged during the rest of the process. This phenomenon may be caused by the fact that grasp methods may get stuck in local optimal solutions. Despite the fact that the proposed GA starts with lower quality solutions, it gradually converges each iteration towards highly fit solutions. In fact, the proposed GA is capable of finding equivalent solutions in approximately 18 seconds, which is a very small time considering that the VO is aimed to be long term. From that point, as time increases, the proposed GA is capable of outperforming the grasp method. As for uniform GA, it can be appreciated that it has a slower convergence than our proposed method. It takes approximately 40 seconds to outperform grasp solutions, which is almost the double than our proposed GA took.

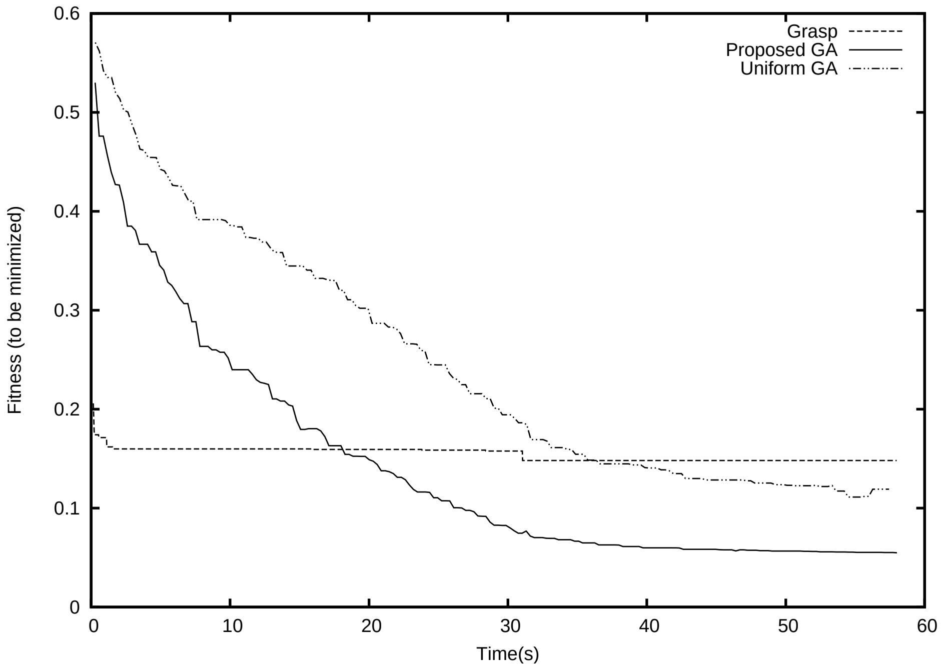


Fig. 3. Fitness evolution through time

For long term VOs, it seems more adequate to employ our proposed method since it is able to get better basis distributions in a relatively small time. Moreover, the proposed method has proved to be more robust in large problem instances and high complexity problem instances. Nevertheless, one issue that has not been covered by our proposed approach is VO adaptation. In open MAS, it is possible that VOs need to adapt themselves in order to increase their benefits. This may result in new agents/hosts that need to be accommodated in the current system. From the results observed in this section, it seems like a good approximation to calculate a good basis distribution after the VO formation phase and then modify this distribution as few as possible to accommodate new resources by means of a fast heuristic (*e.g.*, grasp). Since this may gradually lower the quality of the distribution, it may be wise to re-run the proposed GA after the quality has surpassed a certain threshold.

6.3. *Testing different importance levels for untrust and load*

The goal of this experiment is to assess that the proposed GA is capable of working in different scenarios where different weights are given to the untrust and load levels. For that purpose, we selected a problem instance where 100 agents have to be distributed across 10 hosts. This problem instance was used changing the importance of the different goals: ($w_{load} = 1, w_{untrust} = 0$), ($w_{load} = 0.7, w_{untrust} = 0.3$), ($w_{load} = 0.5, w_{untrust} = 0.5$), ($w_{load} = 0.3, w_{untrust} = 0.7$), and ($w_{load} = 0, w_{untrust} = 1$). The goal of this experiment is to observe how the proposed GA is capable of finding different quality solutions for the same problem conditions depending on the most preferred goal (untrust or load). Each parameter configuration was executed 10 times and parameters of the GA were set to the good values found in the previous experiments. The results of this experiment can be found in Table 7.

Table 7. Mean load level×100 and mean untrust level×100 in the proposed scenarios

		Load Level×100	Untrust Level×100
Weights Importance	$w_{load} = 1$ $w_{untrust} = 0$	74.38	45.09
	$w_{load} = 0.7$ $w_{untrust} = 0.3$	78.27	43.02
	$w_{load} = 0.5$ $w_{untrust} = 0.5$	79.32	40.98
	$w_{load} = 0.3$ $w_{untrust} = 0.7$	81.76	39.53
	$w_{load} = 0$ $w_{untrust} = 1$	89.62	37.95

It can be observed how as the weight given to the load level increases the load level tends to be optimized, whereas the untrust level tends to be ignored. Analogously, as the weight given to the untrust level increases the untrust level tends to be optimized and the load level is gradually ignored. It must be taken into account that for a goal to be *ignored* does not mean that the quality of that goal degrades. It simply means that its value does not affect the quality of the solution whether the goal's value increases or not. In any case, the proposed GA was able to find solutions with different levels of untrust and load depending on the importance of the two different goals.

7. Conclusions and Future Work

In this work, an elitist Genetic Algorithm for the distribution of agents across the shared hosts in a long term agent-based Virtual Organization has been presented. The goal is to provide a proper distribution of agents so that poor performance due to computational problems is avoided. The main differences between the proposed method and other approaches, which mainly come from grid computing, are that the proposed method takes into account agent heterogeneity and partners' trust in each other. The problem is solved by means of a genetic mediated solution which is performed on top of an abstract multi-agent systems platform. The designed abstract architecture is currently being developed using the THOMAS framework,[2,44] which runs on top of JADE and Magentix agent platform.[34,36] THOMAS provides support for virtual organizations in open multi-agent systems. More specifically, it provides services for managing agent registration, managing agent organizations, and publishing agent services.

The proposed genetic algorithm employs mutation operator, ranking selection, and a special crossover operation which we have named as elitist crossover. When this operator is performed, one of the children inherits the most promising segments from the parents with higher probability. A subfitness function which employs domain specific information is used in order to assess the quality of the segments. Experimental results have shown that better results are achieved when our proposed GA is configured to work as uniform crossover using a highly elitist crossover operation.

Moreover, some experiments have been designed and carried out in order to compare the proposed approach with several well-known heuristics. More specifically, it has been compared with a grasp implementation, uniform crossover and several multipoint crossover genetic algorithms. Results imply that the proposed genetic algorithm is able to get statistically better results than the other methods. Moreover, it is robust to larger problems and more difficult problem instances.

Despite the fact that its goal is to provide solutions for long term VOs, the proposed genetic algorithm has been able to outperform faster heuristics (*e.g.*, grasp) in relatively small time.

One of the issues that has not been thoroughly covered in this article is how to adapt the VO when new resources/partners dynamically appear. On the one hand, it seems reasonable to employ the designed genetic algorithm for a basis distribution for a long term VO. On the other hand, other heuristics are able to get less fit solutions very quickly (*e.g.*, grasp). Therefore, it may be adequate to employ faster heuristics to adapt the current distribution to new changes. Once the distribution quality has decreased to a certain threshold, it may be advisable to recalculate a new solution by means of the proposed genetic algorithm. Nevertheless, this issue needs further research and it is appointed as future work.

Acknowledgments

This work is supported by TIN2008-04446, TIN2009-13839-C03-01, CSD2007-00022 and FPU grant AP2008-00600 of the Spanish government, and PROMETEO 2008/051 of the Generalitat Valenciana.

References

1. E. Argente, V. Julián and V. Botti, Multi-Agent System Development Based on Organizations, in *Electronic Notes in Theoretical Computer Science*. 150, (2006) 55–71
2. E. Del Val, N. Criado, M. Rebollo, E. Argente and V. Julián, Service-Oriented Framework for Virtual Organizations, in *Proc. 2009 International Conference on Artificial Intelligence*, eds. H. R. Arabnia, D. de la Fuente and J.A. Olivas (CSREA Press, Las Vegas, 2009), pp. 108–114.
3. I. Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, in *Int. J. High Perform. Comput. Appl.*. 15, (2001) 200–222
4. T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray and N. J. Fiddian, Agent-Based Formation of Virtual Organisations, *Knowledge Based Systems*, 17, (2004) 103–111.
5. M. J. Wooldridge, *Multi-agent Systems: An Introduction*, 2nd edn. (Wiley & Sons, 2001).
6. P. Dawkins and F. Reichheld, Customer Retention as a Competitive Weapon, *Directors and Boards*, 14 (1990) 42–47
7. V. A. Zeithaml, L. L. Berry and A. Parasuraman, The Behavioral Consequences of Service Quality, *Journal of Marketing* , 60(2) (1996) 31–46
8. G. Lee and H. Lin, Costumer perceptions of e-service quality in online shopping, *International Journal of Retail & Distribution Management*, **33(2)** (2005) 161–176
9. Z. Yang and M. Jun, Consumer perception of e-service quality: from Internet purchaser and non-purchaser perspectives, *Journal of Business*, 19(1) (2008) 19–41
10. G. Cybenko, Dynamic Load Balancing for Distributed Memory Multiprocessors, *Journal of Parallel and Distributed Computing*, 7 (1989) 279–301
11. R. Schoonderwoerd, O. E. Holland, J. L. Bruten, and L. J. M. Rothkrantz, Ant-Based Load Balancing in Telecommunications Networks, *Adaptive Behavior*, 5(2) (1997) 169–207
12. Y. Chow and W. H. Kohler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, *IEEE Transactions on Computers* , 28(5) (2006) 354–361
13. J. W. Palmer and C. Speier, A typology of virtual organizations: an empirical study, in *Proc. of the Association for Information Systems*, ed. J. Gupta (America's Conference, Indianapolis, 1997)
14. W. Jansen, W. Steenbakkers and H. Jagers, Electronic Commerce and Virtual Organisations, *Special Issue of eJov*, 1(1) (1999) 54–68
15. S. D. Ramchurn, T. D. Huynh and N. R. Jennings, Trust in multi-agent systems, *The Knowledge Engineering Review*, 19 (2004) 1–25
16. J. Sabater and C. Sierra, Review on Computational Trust and Reputation Models, *Artificial Intelligence Review*, 24(1) (2005) 33–60
17. T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, An integrated trust and reputation model for open multi-agent systems, *Autonomous Agents and Multi-Agent Systems*, 13(2) (2006) 119–154
18. J. M. Such, A. Espinosa, V. Botti and A. García-Fornes, Trust and Reputation Through Partial Identities, in *Proc. 1st International Workshop on Infrastructures*

and Tools for Multiagent Systems at AAMAS'10 , eds. V. Botti, A. García Fornes, J. F. Hubner, A. Omicini and J.A. Rodriguez-Aguilar (IFAAMAS ,Toronto, 2010), pp. 18–25

19. J. A. Giampapa, O. H. Juarez-Espinosa and K. P. Sycara, Configuration management for multi-agent systems, in *Proc. 5th Internation Conference on Autonomous Agents*, (ACM, 2001), pp. 230–231.
20. P. E. Hart, N. J. Nilsson and B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics*, 4 (2) (1968) 100–107
21. R. Dechter and J. Pearl, Generalized best-first search strategies and the optimality of A*, *Journal of ACM*, 32(3) (1985) 505–536
22. S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, (Prentice Hall, Upper Saddle River, New Jersey, 2003), pp. 97–104
23. J. Holland, *Adaptation in Natural and Artificial Systems*, (MIT Press, Cambridge, 1992)
24. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley, Boston, 1989).
25. D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, (Addison-Wesley, Reading, 2002)
26. C. R. Reeves, Genetic Algorithms, in *Handbook of Metaheuristics*, (Springer, 2010), pp. 109–139
27. C. Chang, Y. Kuo and W. Tai, Genetic-based approach for synthesizing texture, *International Journal on Artificial Intelligence Tools*, 17(4) (2008) 731–743
28. S. Valero, E. Argente, V. Botti, J. M. Serra, P. Serna, M. Moliner and A. Corma, DoE Framework for Catalyst Development based on Soft Computing, *Computers & Chemical Engineering.* 33 (2009) 225–238.
29. L. Trujillo, P. Legrand, G. Olague and C. Perez, Optimization of the hölder image descriptor using a genetic algorithm, in *Proc. of the 12th Anual Conference on Genetic and Evolutionary Computation*, eds. M. Pelikan and J. Branke (ACM, New York, 2010), pp. 1147–1154
30. V. Sánchez-Anguix, S. Valero, V. Julián, V. Botti and A. García-Fornes, Evolutionary-aided negotiation model for bilateral bargaining in Ambient Intelligence domains with complex utility functions, *Information Sciences*, In Press (2010), doi:10.1016/j.ins.2010.11.018
31. V. Sánchez-Anguix, S. Valero and A. García-Fornes, Tackling Trust Issues in Virtual Organization Load Balancing, in *Proc. of the 23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, eds. N. García-Pedrajas, F. Herrera, C. Fyfe, J. M. Benitez and M. Ali (Springer,2010), pp. 586–595
32. V. Sánchez-Anguix, A. Espinosa, L. Hernández and A. García-Fornes, MAMSY: A Management Tool for Multi-Agent Systems, in *Proc. 7th International Conference on Practical Applications of Agents and Multi-Agent Systems*, eds. Y. Demazeau, J. Pavón, J.M. Corchado and J. Bajo (Springer Berlin, Heidelberg, 2009), pp. 130–139.
33. G. Karjoth, D. B. Lange and M. Oshima, A Security Model for Aglets, *IEEE Internet Computing*, 1(4) (1997) 68–77
34. F. Bellifemine, A. Poggi and G. Rimassa, JADE - A FIPA compliant agent framework, in: *Proc. Practical Applications of Intelligent Agents*, (1999), pp. 97–108
35. A. Pokahr, L. Braubach and W. Lamersdorf, Jadex: Implementing a BDI-Infrastructure for JADE Agents, *EXP - In Search of Innovation (Special Issue on JADE)*, 3(3) (2003) 76–85

36. J. M. Alberola, J. M. Such, A. Espinosa, V. Botti and A. García-Fornes, Magentix: a Multiagent Platform Integrated in Linux, in *Proc. 6th European Workshop on Multi-Agent Systems*, (Bath, 2008), pp. 1–10
37. M. Hoogendoorn and C. M. Jonker, Formation of Virtual Organizations through Negotiation, in *Proc. 4th German Conference on Multiagent Technologies*, (Springer,2006), pp. 135–146.
38. V. Di Martino and M. Mililotti, Sub Optimal Scheduling in a Grid Using Genetic Algorithms,*Parallel Computing*. 30, (2004) 553–565
39. J. Cao, D. P. Spooner, S. A. Jarvis and G. R. Nudd, Grid Load Balancing Using Intelligent Agents, *Future Gener. Comput. Syst.*. 21, (2005) 135–149
40. R. F. de Mello, J. A. A. Filho, L. J. Senger and L. T. Yang, RouteGA: A Grid Load Balancing Algorithm with Genetic Support, in *Proc. 21st International Conference on Advanced Networking and Applications*, (IEEE Computer Society, Washington, 2007), pp. 885–892.
41. C. Hewitt, Open Information Systems Semantics for Distributed Artificial Intelligence, in *Artificial Intelligence*, 47(1-3) (1991) 79–106.
42. D. G. Saari, Mathematical Structure of Voting Paradoxes: II. Positional Voting, *Economic Theory*, 15(1), (2000) 55–102
43. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*, (Cambridge University Press, Cambridge, 2009)
44. E. Del Val, N. Criado, C. Carrascosa, V. Julián, M. Rebollo, E. Argente and V. Botti, THOMAS: A Service-Oriented Framework for Virtual Organizations, in *Proc. 9th International Conference on Autonomous Agents and Multiagent Systems*, (Toronto, 2010), pp. 1631–1632
45. L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, Biases in the Crossover Landscape, in *Proc. 3rd International Conference on Genetic Algorithms*, ed. J. D. Schaffer (Morgan Kaufmann, San Francisco, 1989), pp. 10–19.
46. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, (Springer-Verlag, New York, 1994)
47. J. E. Baker, Adaptive Selection Methods for Genetic Algorithms, in *Proc. of the 1st International Conference on Genetic Algorithms*, ed. J. J. Grefenstette (L. Erlbaum Associates, Hillsdale, 1985), pp. 101–111.
48. K. F. Man, K. S. Tang and S. Kwong, Genetic algorithms: concepts and applications, in *IEEE Transactions on Industrial Electronics*, 43(5) (2002) 519–534
49. T. A. Feo and M. G. C. Resende, A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem, *Operations Research Letters*. 8, (1989) 67–71
50. M. G. C. Resende and C. C. Ribeiro, Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications, in *Handbook of Metaheuristics*, (Springer, 2010), pp. 283–319.
51. W. M. Spears and K. A. DeJong, An analysis of multipoint crossover, *Foundations of Genetic Algorithms*, 2, (1991), 301–315
52. G. Syswerda, Uniform crossover in genetic algorithms, in *Proc. 3rd International Conference on Genetic Algorithms*, ed. J. D. Schaffer (Morgan Kaufmann, San Francisco, 1989), pp. 2–9