Managing Combinatorial Optimization Problems by means of Evolutionary Computation and Multi-Agent System

Mauricio Paletta
Centro Inv. en Inf. y Tec. de la Comp. (CITEC)
Universidad Nacional de Guayana (UNEG)
Ciudad Guayana, Venezuela
Email: mpaletta@uneg.edu.ve

Pilar Herrero
Facultad de Informática
Universidad Politécnica de Madrid (UPM)
Madrid, Spain
Email: pherrero@fi.upm.es

***Abstract*—Having the necessity of solving a combinatorial optimization problem is very common, especially in those cases where there is many parameters that should be properly configured to solve a specific problem. Several searching techniques could be used to deal with such situations. Unfortunately, depending on the complexity of the problem, some of these techniques require high computational capabilities to be applied, primarily in those cases in which measuring the quality of a potential solution is very demanding in these capabilities. However, advances in Distributed Artificial Intelligence (DAI), Multi-Agent Systems (MAS) to be more specific, could help users to deal with this matter. This paper presents an inter-agent MAS protocol for evolving an Evolutionary/Genetic Program, aiming to reduce the computational requirements, and allowing a response within a reasonable period of time. This paper also presents a JADE-based implementation and evaluation, focusing on the interaction among agents in the MAS, and emphasizing on the optimization obtained by means of the evolutionary algorithm/technique.**

I. Introduction

EVOLUTIONARY/GENETIC Programs (EPs) [6], [10] are powerful searching techniques used to solve Combinatorial Optimization Problems (COPs) in many disciplines. Its objective is to achieve the convergence to a good solution (the optimal or close to it) to those problems. Depending of the complexity of the problem, EPs could require high computational capabilities such as CPU time, memory, etc. This problem increases considerably if the calculation of the fitness function also requires high computational capabilities for each of the potential solutions. The combination of EP and MAS technologies [8] is an alternative to deal with this situation, aiming to distribute the computational recourses in many sources**.**

To cover the necessity previously mentioned, this paper focuses on defining a multi-agent architecture, establishing an interaction protocol for obtaining a suitable solution, based on EP specifications for a specific COP. The paper also presents details of the implementation for this proposal and some results obtained from experiments done on a particular COP.

The rest of the paper is organized as follows: section 2 presents some work related to the scope of this paper; section 3 describes the proposed inter-agents protocol; section 4 shows details of the implementation and evaluation of this proposal. Finally, section 5 reaches some conclusions and future work.

II. Related Work

Some examples of making evolutionary algorithms in distributed environments can be consulted on [1], [14], [18]. In [1] authors describe the DREAM (Distributed Resource Evolutionary Algorithm Machine) framework for the automatic distribution of evolutionary algorithms. This research is focused on an evolution algorithm approach more than the MAS inter-agent communication aspects.

In the same order of ideas, authors in [14] present an approach to an addressing automatic test generating system in distributed computing context by the integration of genetic algorithm and MAS. Authors do not use a communication protocol for taking charge of scheduling the genetic operations. Instead, they use a control agent.

Finally, in [18] authors use a MAS guided by a multi-objective genetic algorithm to find a balance point in the respect of a solution of the Pareto front. Interaction between agents in this research is not properly done with a communication protocol; instead authors use functions of communication.

DREAM [1] and G2DGA [3] are examples of frameworks concerning development of Peer-to-Peer distributed computing systems. Using JADE [2], [4] as a middleware to propose a multi-agent synchronous evolutionary system can be reviewed on [7], [13], [14].

However, as we know, no work has focused on the main purpose of this proposal, which has to do with the design of an inter-agent communication protocol for cooperation aiming to implement an EP. The newscast protocol [11],

which is a lazy fully distributed information propagation protocol, is an example of a work with the same goal but different scope.

III. Overall Inter-Agent Protocol

The proposed inter-agents communication protocol occurs in the general MAS-based architecture depicted in Fig. 1. It basically consists of two different types of agents:

1) EPEnvAgent: It informs the rest of the agent about the EP parameters and problem specifications. It also takes control of the population growth. On the other hand, it allows the evolution among multiple system nodes.

2) EPAgent: It represents a potential solution to the problem as well as carry out the main steps of evolutionary computation: selection and variation (crossover and mutation) - to generate descendants (new agents and therefore new potential solutions).

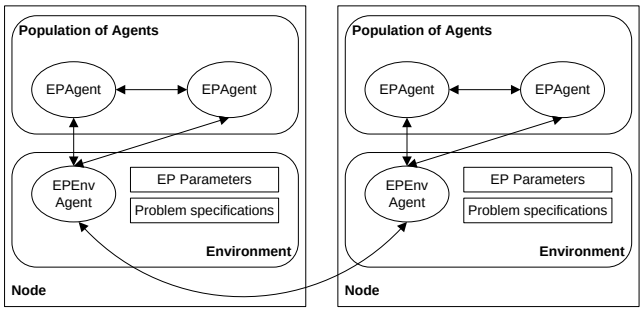

Fig. 1 General architecture

Other elements involved in the architecture are the EP parameters (population size, probability of applying genetic operators, among others), and the problem specifications which is necessary to define a specific COP.

Each computational node may have multiple EPAgent (a population) but only one EPEnvAgent is needed. The number of EPAgent in the system (population size) may vary, as the EP evolves. An EPAgent can be created as well as eliminated from the system. The creation of new EPAgent is the result of applying genetic operators, while the disposal is due to self-destruction or suicide. Some parameters are used to control these functions.

As it is happening with the agent containers in JADE, there are two execution modes for the EPEnvAgent: 1) main, and 2) auxiliary. There is only one main EPEnvAgent in the entire system and corresponds to the first instantiated agent. For each node in which the main EPEnvAgent is missing, an auxiliary EPEnvAgent is necessary. Therefore, it is possible to distribute the EP execution between multiple computational nodes.

On the other hand, and following the FIPA ACL specifications [9], agents communicate with each other according to the messages shown in Fig. 2. Messages in this inter-agent communication protocol are the following:

- INFORM, from EPAgent to EPEnvAgent. For EPEnvAgent to keep some statistics of the evolutionary process and keep track of the best solution it has at any given time, any EPAgent uses this message to report or inform its measure of quality (fitness), as well as the structure that represents the solution to the COP (chromosome).
- PROPAGATE, from EPEnvAgent to EPEnvAgent. When an EPEnvAgent realizes that has obtained a better individual (solution) in the population of EPAgents, it uses this message to transmit /propagate this information to the rest of EPEnvAgent. Therefore all nodes in the system keep updated.
- REQUEST, from EPAgent to EPEnvAgent. In this proposal, EP parameters may change dynamically. An EPAgent should, from time to time, requests by using this message the EPEnvAgent to send back the parameters. Therefore, new possible values can be known.
- INFORM, from EPEnvAgent to EPAgent. In response to the previous message, the EPAgent is receiving the EP parameters from the EPEnvAgent. It is important to mention that these parameters can change the way in which EPAgents may vary to produce descendants (new EPAgents) or self-destruction. Therefore, the population growth can be controlled.
- INFORM_REF, from EPEnvAgent to EPEnvAgent. By using this message, the main EPEnvAgent informs all the auxiliary EPEnvAgent changes in the EP parameters.
- PROPOSE, from EPAgent to EPEnvAgent. In this proposal, unlike other similar proposals that cover the same scope of this paper, each EPAgent is responsible for searching (selecting) a suitable partner to cross (in other proposals, this is done by using a control agent [14]). Through this message, the EPAgent makes a request to find a suitable partner by sending its fitness and Id as parameters of the message. These parameters are needed when some other EPAgent accepts the request and responds appropriately to the sender. This is practically the selection mechanism used in this proposal, which is necessary for all EP.
- PROPOSE, from EPEnvAgent to EPAgent. By this message, the EPEnvAgent replicates the proposal sent by an EPAgent for the rest of EPAgents. They become aware of it, so that, they can respond directly to EPAgent who made the original request (because agent Id is one of the message parameters).
- ACCEPT_PROPOSAL, from EPAgent to EPAgent. An EPAgent uses this message to respond positively to a request from another EPAgent for crossing with it. This happens only if it decides to accept the proposal (based on the fitness received from the sender agent and by using some probabilities). The EPAgent sends its chromosome so that the sender can use it for applying the corresponding genetic operation.
- REQUEST_WHENEVER, from EPAgent to EPEnvAgent. EPAgents are responsible for the selection mechanism and for applying genetic operators to generate new EPAgents. These new agents must be created by the EPEnvAgent. Through this message, the EPEnvAgent knows

that a new EPAgent must be created with the chromosome given as a parameter of the message.

- CONFIRM, from EPEnvAgent to EPAgent. EPEnvAgent has the control to start and stop the evolution. When the process has to be finished, the EPEnvAgent uses this message to give to all EPAgents the order to self-destruct.
- CONFIRM, from EPAgent to EPEnvAgent. Once an EP-Agent is self-destructed, either because it received an order from the EPEnvAgent or for effect of this evolutionary algorithm (by using some parameters and based on the current fitness), its uses this message to confirm his suicide.
- CONFIRM, from EPEnvAgent to EPEnvAgent. If the receiver is the main EPEnvAgent, then some auxiliary EPEnvAgent has decided to stop its participation in the evolution process. On the other hand, if the receiver is an auxiliary EPEnvAgent, then the entire evolution process must be stopped.

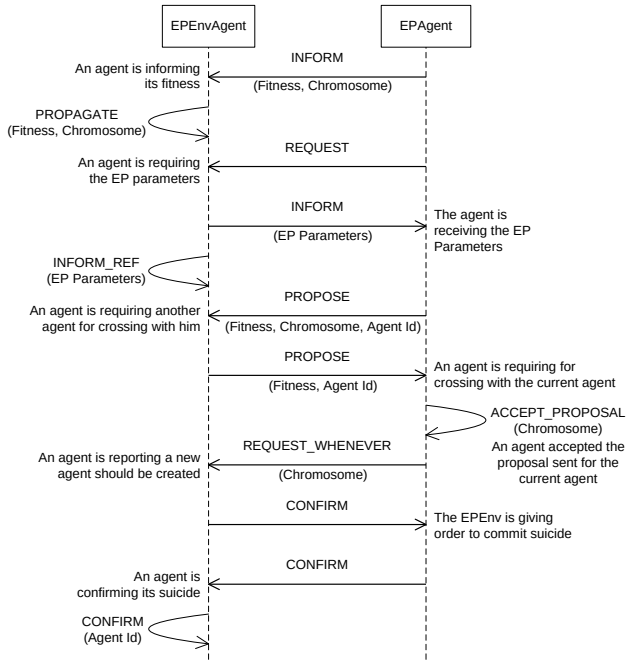

Fig. 2 Inter-agents communication protocol

A method for estimating the total number of messages *TM* requires to complete an evolution, which has duration of *T* time units, is shown below. It has the following parameters:

- p_1: the probability that an agent commits self-suicide.
- p_2: the probability that an agent accepts to cross with another agent.
- p_3: the probability that an agent will produce at least one descendant.
- t_1: time interval in which parameters are sought.
- *NA*: the total number of agents.
- Na_i: the total number of agents for node *i*.
- *n*: the total number of nodes.
- Msg_i: the total number of messages of type *i*, where *i* is: 1) INFORM(EPAgent→EPEnvAgent), 2) PROPAGATE, 3) REQUEST, 4) INFORM(EPEnvAgent→EPAgent), 5) INFORM_REF, 6) PROPOSE(EPAgent→EPEnvAgent), 7) PROPOSE(EPEnvAgent→EPAgent), 8) ACCEPT_PROPOSAL, 9) REQUEST_WHENEVER, 10) CONFIRM(EPEnvAgent→EPAgent), 11) CONFIRM(EPAgent→EPEnvAgent), and 12) CONFIRM(EPEnvAgent→EPEnvAgent).

$$
\begin{aligned}
&TM = \sum_{i=1}^{12} Msg_i; \quad NA = \sum_{i=1}^{n} Na_i \\
&Msg_1 = Msg_7 = p_1 * p_2 * NA; \quad Msg_2 = n * Msg_1; \\
&Msg_3 = Msg_4 = n * T / t_1; \quad Msg_5 = n * Msg_3; \\
&Msg_6 = Msg_7 = p_3 * NA; \quad Msg_8 = Msg_9 = p_2 * Msg_6; \\
&Msg_{10} = Msg_{11} = p_1 * NA; \quad Msg_{12} = n * Msg_{10}
\end{aligned}
\tag{1}
$$

Next section provides details of the implementation of the proposal shown above as well as some results from experiments conducted with it.

IV. Implementation and Evaluation

A. Implementation

In order to implement the inter-agents communication protocol proposed in this paper as well as the MAS-based evolutionary algorithm, a JADE-based framework called EP-MAS.Lib [16] was developed. EP-MAS.Lib is a second-layer framework to define and resolve any particular COP by using the combination of EP and MAS technologies, being JADE the first-layer framework. As one of the differences from previous works in this same context, EP-MAS.Lib allows the definition of any problem through the incorporation of operations for the calculation of fitness and genetic operation of crossing and mutation. In this matter, Fig. 3 shows the EP-MAS.Lib class diagram with the new classes defined and the relationship of these classes with some of the JADE classes. Classes defined on the EP-MAS.Lib framework are the following:

- EPEnv: It represents the environment in the MAS; has the JADE containers; contains the EPEnvAgent, EP parameters and problem specifications.
- EPParam: It has the set of parameters to control the EP.
- EPEnvAgent: It is used to define the EPEnvAgent explained in Section 3.
- EPEnvAgentBehaviour: It is the JADE behaviour model associated to the EPEnvAgent.
- EPAgent: It is used to define the EPAgent explained in Section 3.
- EPAgentBehaviour: It is the JADE behaviour model associated to the EPAgent.
- EPProblem: It is an abstraction of the problem specifications (see below for details).
- FitnessExp: It is an abstraction of the fitness expression to calculate the solution quality represented by the EPAgents.

- CrossoverOper: It is an abstraction of the crossover genetic operation to generate one or two new chromosomes from two given chromosomes.
- MutationOper: It is an abstraction of the mutation genetic operation to generate a new chromosome from a given chromosome.

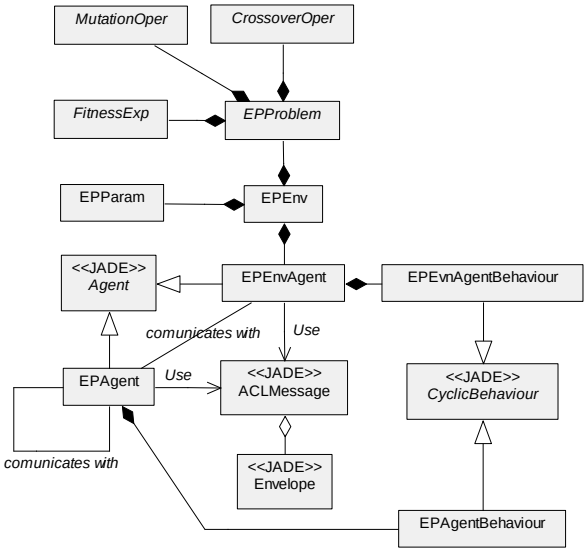

Fig. 3 EP-MAS.Lib class diagram

To define a specific COP to be solved with EP-MAS.Lib, it is necessary to do the following steps (an example of the code appears in the appendix at the end of this paper):

1) To define the chromosome as a *n*-vector of real values, in which *n* is the chromosome size.

2) To define a new specialized class from the EPProblem class; the virtual methods "getRandomChromosome" and "IsChromosomeValid" have to be defined.

3) To Implement the FitnessExp interface and defining the CalculateFitness method.

4) To implement, for each genetic operation required, the CrossoverOp and/or the MutationOp interfaces depending on what kind of operation is being done. For each CrossoverOp implementation, it is necessary to define the MakeCrossing method. For each MutatioOp implementation, it is necessary to define the MakeMutation method.

5) In the new class constructor, an instance of the FitnessExp implementation must be assigned to the problem with the method "setFitnessExp".

6) In the new class constructor, an instance for each CrossoverOp and MutationOp implementations has to be added to the problem with the methods "addCrossoverOp" and "addMutationOp".

Algorithm 1 and Algorithm 2 below show the pseudo-code related with the communication process used in this MAS-based proposed, and conducted between the EPEnvAgent and EPAgent agents. Algorithm 1 corresponds to EPEnvAgent and Algorithm 2 corresponds to EPEnv respectively.

Algorithm 1 *EPEnvAgent*

```
Loop
  msg ← ReceiveMessage()
  if (¬msg) then
    Wait()
  if (msg = INFORM) then
    if (msg.Fitness > current fitness) then
      UpgradeBestSolution(msg.Fitness, msg.Chromosome)
      ∀x, x is EPEnvAgent
        SendMessage(x, PROPAGATE(msg.params))
  if (msg = INFORM_REF) then
    UpgradeParameters(msg.EP parameters)
  if (msg = PROPOSE) then
    ∀x, x is EPAgent
      SendMessage(x, PROPOSE(msg.params))
  if (msg = REQUEST) then
    if (status = stopped) then
      ReplyMessage(CONFIRM)
    else
      ReplyMessage(INFORM(EP parameters))
  if (msg = REQUEST_WHENEVER) then
    CreateEPAgent(msg.Chromosome)
    PopulationSize ← PopulationSize + 1
  if (msg = CONFIRM) then
    if (¬msg.Id) then
      PopulationSize ← PopulationSize – 1
    else
      ∀x, x is EPAgent
        SendMessage(x, CONFIRM)
  if (msg = PROPAGATE) then
    UpgradeBestSolution(msg.Fitness, msg.Chromosome)
end loop
```

Algorithm 2 *EPAgent*

```
loop
  msg ← ReceiveMessage()
  ea ← the EPEnvAgent
  if (¬msg) then
    if (has to be crossed) then
      SendMessage(ea, PROPOSE(current fitness,
        current chromosome, Id))
    if (has to be mutated) then
      Chrom ← ApplyMutation()
      SendMessage(ea, REQUEST_WHENEVER(Chrom))
    Wait()
  if (msg = INFORM) then
    UpgradeParameters(msg.EP parameters)
    if (has to be self-destructed) then
      Delete()
      SendMessage(ea, CONFIRM)
  if (msg = PROPOSE) then
    if (accepted the proposal) then
      SendMessage(msg.Id, ACCEPT_PROPOSAL(current
        fitness, current chromosome))
  if (msg = CONFIRM) then
    Delete()
    SendMessage(ea, CONFIRM)
  if (msg = ACCEPT_PROPOSAL) then
    C[] ← ApplyCrossover(current chromosome, msg.Chr.)
    ∀c ∈ C[], c is a Chromosome
    SendMessage(ea, REQUEST_WHENEVER(c))
end loop
```

B. Evaluation

For evaluating the proposal presented in this paper we have implemented a solution for two different problems,

varying the computational capabilities demanded to calculate the fitness of an individual or potential solution for the problem.

The first issue to be considered is related to the Traveling Salesman Problem (TSP) which is a well-known NP-hard COP [12], [19]. In this problem, there are *C* cities and the distance from city *i* to city *j* is d_{ij}. A tour is a path that starts from a city; visits each city exactly once, and goes back to the starting city. The goal is to find a tour with the minimum possible length (inverse in this case because in an EP the goal is to maximize the measure of quality). The fitness (quality) of a valid solution (chromosome → EPAgent) to this problem consists in calculating the length of a path. In this case, computational capabilities demanded are very low. The experimentation was conducted using the data relative to the 51-cities Christofides and Eilon [5].

The second problem has to do with finding the proper setting of parameters required to configure an Artificial Neural Network (ANN) for a particular investigation that is currently being developed, and whose first results can be consulted in [15]. Fitness in this problem consists in training an ANN and obtains the corresponding total average error aiming to reduce it. The ANN is configured with the parameters indicated in the particular individual. Unlike the previous case, the calculation of this fitness is very demanding on computational capabilities.

Both problems were implemented using a conventional simple genetic program, as well as using the distributed model presented in this paper, aiming to compare the efficiency of each case to deal with the same problem. The experimentation for the simple program was conducted in a PC with the following hardware platform: Intel T2600 (2.16 GHz) with 2 GB RAM. The experimentation for the distributed solution was conducted using three different nodes with the following hardware platform: 1) Intel T2600 (2.16 GHz) with 2 GB RAM; 2) Pentium 4 (3 GHz) with 1 GB RAM; 3) Pentium 4 (3.2 GHz) with 1 GB RAM.

Based on the results obtained we observed:

1) A solution for TSP problem was obtained more efficiently using the simple genetic program than the MAS-based distributed proposal. The difference is about 10 to 1 in execution time.

2) Related to the ANN configuration, the simple genetic program needed more than two days for obtaining a satisfactory solution. However, by using the MAS-based distributed proposal (with the 3 nodes previously mentioned), a solution was obtained in no more than 6 hours.

Fig. 4 shows a screenshot of the test application in which the main EPEnvAgent is created (node-1) as well as the initial population of EPAgents associated with this node. Left hand side has the EP parameters. If a parameter has to be changed at runtime, simply change the value and press the associated button. Right hand side has some current statistics of the evolution (major and minor fitness, best chromosome, among others). For the other nodes, the test application window is similar to the one shown in Fig. 4, except that parameters cannot be changed.

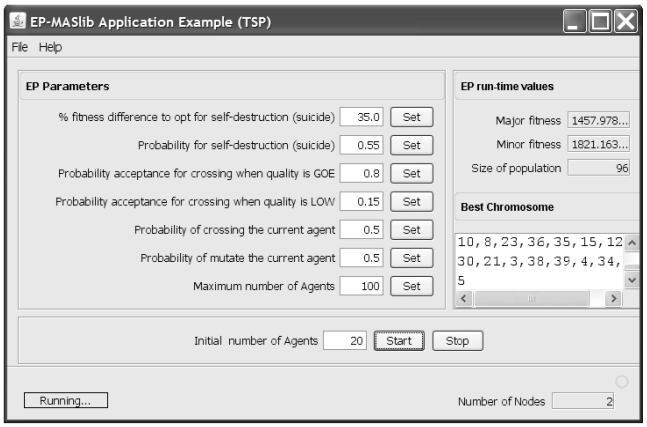

Fig. 4 Screenshot of the main Test Application window for TSP problem

On the other hand, Fig. 5 shows a screenshot of the JADE Sniffer agent window in which it's possible to see the interaction between the agents and how the protocol is working. The results show that the proposed protocol allows distributed computational capabilities among more than one node in such a way to expedite the process of convergence of the searching algorithm. It is important to mention that the effectiveness and efficiency of the evolutionary process (find the optimal solution in shortest time) can be improved by changing the EP parameters as well as evolutionary genetic operators. This is fully configurable within this proposal.

Due to the fact that JADE is costly, it is important to consider the maximum number of agents to be created in any node. According to experiments made, a number between 20 and 100 is acceptable. Of course, a higher number of individuals (agents) in the evolution can converge faster to a solution, but more computational resources are required. Since each node has different computational capacities, this parameter is one that can be changed dynamically during the evolution process. Moreover, it is possible to assign a different value for each node based on the capacities of this.

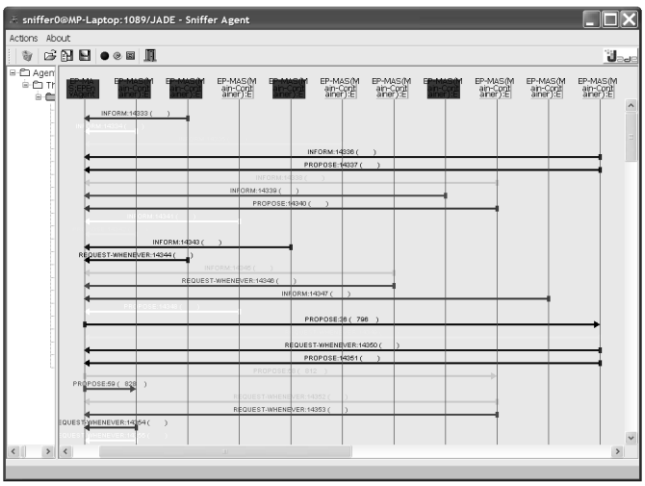

Fig. 5 Screenshot of the JADE sniffer

Aiming to compare this approach with another similar work, DREAM was used to implement the same experiments. Both proposals are capable of performing the EP in distributed environments and reach an acceptable solution to the COP. However, for environments where the aspects related to communication between nodes is not a problem (eg a Local Area Network), the proposal presented in this paper, which is primarily based on a communication protocol between agents, tends to converge faster on those COPs with greater complexity. For Wide Area Network environments, this proposal may not be the most convenient.

V. Conclusions and Future Work

In this paper we present a communication protocol between agents to be used in a multi-agent system scenario aiming to resolve a specific combinatorial optimization problem. Our proposal was implemented by using JADE and as we know, this proposal differs from other similar works in the following aspects:

- It focuses on the communication process between the agents in the systems and therefore to the cooperation needed for solving the evolutionary algorithm, instead of the properly used elements of the evolutionary algorithm (selection mechanism and genetic operations).
- The evolutionary algorithm (selection and variation) is not controlled by a central entity, instead it's controlled by all individuals (agents) of the population actively involved in this process.
- The needed information (EP parameters and problem specifications) is not located in a central repository, but it is replicated for all who need it.
- The definition of a particular COP by using the procedural shown in Section 4.1 is possible.

The obtained result point out that agent in a MAS-based environment can interact with each other to solve any COP by using the communication protocol proposed. Although using a simple genetic program is more efficient than using the distributed solution proposal for the majority COPs, problems, with complex fitness calculations that require high demanding computational capabilities are more efficiently solved with the MAS-based distributed proposal than simple genetic programs.

On the other hand, another important difference to consider between simple genetic program and the distributed proposal is that the last one can dynamically change its efficiency by adding new nodes to the CDE.

We are working on extending this proposal to be used in collaborative grid environments as well as the possibility of further reducing the flow of messages and data that is required to avoid possible bottleneck. References [15], [17] are some previous work related to the relationship between CDE and grid computing where this proposal can be extended.

Appendix

```
public class COP_Name extends EPProblem {
  // Chromosome size
  //
  public final short ChrSize = 51;

  // Data related to the problem…

  // Fitness calculation
  //
  private class theFitness implements
    EPProblem.FitnessExp {
    public double CalculateFitness(
      double []crom) {

      // Code to calculate fitness from crom

      return Fitness;
    }
  }

  // Genetic operations
  //
  private class CrossoverOp1 implements
    EPProblem.CrossoverOp {
      public Object []MakeCrossing(double []crom1,
        double []crom2) {

        // Code to create two descendants D1 & D2

        return new Object[] { D1, D2 };
    }
  }

  private class MutationOp1 implements
    EPProblem.MutationOp {
      public double []MakeMutation(double []crom) {
        // Code to create one descendant D1
        return D1;
      }
  }

  // New genetic operations…

  // Get a valid randomly chromosome
  //
  public double[] getRandomChromosome() {

    // Code to create a chromosome C

    return C;
  }

  // Class constructor
  //
  public COP_Name() {
    super();
    setChromosomeSize(ChrSize);
    setFitnessExp(new theFitness());
    addCrossoverOp(0.95, new CrossoverOp1());
    addMutationOp(0.65, new MutationOp1());
    invFitness = true;
  }
}
```

References

[1] M. I. Arenas, P. Collet, A. E. Eiben, M. Jelasity, J. J. Merelo, B. Paechter, M. Preuss, and Marc Schoenauer, "A Framework for Distributed Evolutionary Algorithms", *in Proc. 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*. LNCS 2439, pp. 665-675, 2002.

[2] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE – A FIPA-compliant agent framework", *Telecom Italia internal technical*

report. *In Proceedings International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAM'99)*, pp. 97-108, 1999.

[3] J. Berntsson, "G2DGA: an adaptive framework for internet-based distributed genetic algorithms", *in Proc. of the 2005 workshops on Genetic and Evolutionary Computation (GECCO)*, pp. 346-349, 2005.

[4] K. Chmiel, M. Gawinecki, P. Kaczmarek, M. Szymczak, and M. Paprzycki, "Testing the Efficiency of JADE Agent Platform", *in Proc. 3rd International Symposium on Parallel and Distributed Computing (ISPDC), IEEE Computer Society Press*, 13(2) pp. 49-57, 2005.

[5] N. Christofides, and S. Eilon, "Algorithms for Large-Scale Travelling Salesman Problems", *Operations Research Quarterly*, 23(4), pp. 511-518.

[6] A. E. Eiben, and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Verlag. ISBN: 3-540-40184-9.

[7] A. E. Eiben, M. Schoenauer, J. L. Jiménez, P. A. Castillo, A. M. Mora, and J. J. Merelo, "Exploring Selection Mechanisms for an Agent-Based Distributed Evolutionary Algorithm", *in Proceedings Genetic and Evolutionary Computation Conference (GECCO)*. ACM 978-1-59593-698-1/07/0007, pp. 2801-2808, 2007.

[8] J. Ferber. Les systems multi-agents, *Vers une intelligence collective*. Ed. InterEditions, pp. 1-66, 1995.

[9] Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification, SC00061, Geneva, Switzerland (2002), http://www.fipa.org/specs/ fipa00061/ index.html.

[10] L. C. Jain, V. Palade, and D. Srinivasan (Eds.), *Advances in Evolutionary Computing for System Design*. *Studies in Computational Intelligence*, vol. 66. Springer Verlag. ISBN: 978-3-540-72376-9, 2007.

[11] M. Jelasity, and M. van Steen, "Large-scale newscast computing on the Internet", *Technical Report IR-503*, Vrije Universiteit Amsterdam, Department of Computer Science, October, 2002.

[12] E. L. Lawler, J. K. Lenstra, A. H. Rinnooy, and D. B. Shmoys (Eds.), *The Travelling Salesman Problem: A guided tour of combinatorial optimization*. New York: Wiley and Sons. ISBN: 978-0471904137, 1985.

[13] W. Lee, "Parallelizing evolutionary computation: A mobile agent-based approach", *Expert Systems with Applications*, 32(2), pp. 318-328, 2007.

[14] A. Meng, L. Ye, D. Roy, and P. Padilla, "Genetic algorithm based multi-agent system applied to test generation", *Computers & Education 49*, pp. 1205-1223, 2007.

[15] M. Paletta, and P. Herrero, "Learning Cooperation in Collaborative Grid Environments to Improve Cover Load Balancing Delivery", *in Proc. IEEE/WIC/ACM Joint Conferences on Web Intelligence and Intelligent Agent Technology*. IEEE Computer Society E3496, ISBN: 978-0-7695-3496-1, pages 399-402, Sydney, Australia, December 2008.

[16] M. Paletta, and P. Herrero, "EP-MAS.Lib: A MAS-Based Evolutionary Program Approach", *in Proc. 4th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2009)*, E. Corchado et al. (Eds.), LNAI 5572, Springer-Verlag, pp. 9-17, 2008.

[17] M. Paletta, and P. Herrero, "Foreseeing Cooperation Behaviors in Collaborative Grid Environments", *in Proc. 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'09)*, Vol. 50/2009, Springer-Verlag, pp. 120-129, 2009.

[18] J. P. Vacher, T. Galinho, F. Lesage, and A. Cardon, "Genetic Algorithms in a Multi-Agent system", *in Proc. IEEE International Joint Symposia on Intelligent and Systems*, pp. 17-26, 1998.

[19] L. Wang, A. A. Maciejewski, H. J. Siegel, and V. P. Roychowdhury, "A comparative study of five parallel genetic algorithms using the travelling salesman problem", *in Proc. 11th International Parallel Processing Symposium (IPPS)*. IEEE Computer Society Press, pp. 345-349, 1997.