Exploring Selection Mechanisms for an Agent-Based Distributed Evolutionary Algorithm

J. L. J. Laredo
Department of Architecture and Computer Technology
University of Granada. ETSIT.
juanlu@geneura.ugr.es

E. A. Eiben
Department of Computer Science
Vrije Universiteit Amsterdam
gusz@cs.vu.nl

M. Schoenauer
Projet TAO, INRIA Futurs
LRI, Univ. Paris-Sud
marc@lri.fr

P. A. Castillo
Department of Architecture and Computer Technology
University of Granada. ETSIT
pedro@atc.ugr.es

A. M. Mora
Department of Architecture and Computer Technology
University of Granada. ETSIT
amorag@geneura.ugr.es

J. J. Merelo
Department of Architecture and Computer Technology
University of Granada. ETSIT
jmerelo@geneura.ugr.es

ABSTRACT

In this paper we propose an agent-based model of evolutionary algorithms (EAs) which extends seamlessly from concurrent single-host to distributed multi-host installations. Since the model is based on locally executable selection, we focus on the comparison of two selection mechanisms which accomplish with such a restriction: the classical tournament method and a new one called autonomous selection. Using the latter method the population size changes during runtime, hence it is not only interesting as a new selection mechanism, but also from the perspective of scalable networks.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search, Heuristic Methods*

; C.2.4 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS—*Distributed Systems*

General Terms

Algorithms

Keywords

self-adaptation, genetic algorithms, multi-agent systems, parallelization, performance analysis

1. INTRODUCTION

Distributed applications are emerging with renewed interest since the Peer-to-Peer(P2P) paradigm [16] appeared on the area of heterogeneous networks. In order to deal with development of applications in that environment, questions such as *scalability* (since P2P systems can become large-scale networks) or *robustness* (given that they have dynamics topologies with continuous variants either on the connected nodes or the network structure) become of the maximum interest and have to be addressed [19].

This paper outlines general aspects for the development of distributed Evolutionary Algorithms (EAs) specially concerning P2P networks and focuses on the analysis of two selection mechanisms for a proposed new model.

EAs are population-based stochastic search methods with an inherent parallelism [7]. Their possible parallelism has been widely studied (see e.g. [4] for a survey) but mainly under two approaches: master-slave and islands. In the master-slave mode the algorithm runs on the master and the individuals are sent for evaluation to the slaves, in an approach usually called *farming*. Using the island model several EAs (islands) are employed processing their own population, and exchanging the best individuals between islands with a certain rate [3]. Both cases present major adoption problems in heterogeneous fully decentralized networks such as P2P networks. On the one hand, master-slave features do not match with large-scale system robustness (master represents a single point of failure) and scalability (since it depends on evaluation function cost, and has a bottleneck in the efficiency of the master performing the evolutionary operations). On the other hand, P2P systems do not provide the knowledge of the global environment that the island model needs in order to set parameters such as the number of islands, the population size per island and the migration rate.

Nevertheless, there is a third, finer grained approach, termed Fully Distributed Model, in which processors host single individuals that evolve on their own. Operations that require more than a single individual (e.g., selection and crossover) take place among a defined set of neighbors (between individuals on different nodes or available locally to a node). This model is able to adapt to heterogeneous networks since some P2P overlay networks [5] provide a dynamic neighborhood whose size grows logarithmically with respect to the total size of the system in a small-world fashion. Following

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

a gossip style, these small-world networks spread information in an epidemic manner through the whole network (as can be seen in [10, 9]), what means that the risk of having obsolete individuals across the network is minimized as a consequence of the probabilistic global "infection" that the nodes undergo.

It is obviously not straightforward to outline a method that takes advantage of those P2P properties, obtaining at the same time high performance and good scalability. That is why we propose a model where each individual in an evolutionary computation population schedules its own actions. This model is a step towards a "Fully Distributed Model" for designing EAs in heterogeneous networks.

A first analysis on the model scalability can be seen in [12] while within this paper we compare two different selection mechanisms for suitability within this framework: firstly, a classical tournament method and secondly the autonomous selection. Autonomous selection (described in detail in section 3.2) lets each individual control its own reproductive and life status based on an estimation of some population fitness statistic. This status determines whether the individual will die or survive, and if it survives, whether it is fertile to produce offspring. It is essential here that death and birth are not synchronized. An agent can die without producing offspring, and newly created children can be added to the population without removing old ones. As a consequence, this mechanism leads to a runtime population size adjustment. Such feature can offer benefits when running on a P2P system since available resources change dynamically.

Since the model is based on locally executable selection, the main objective of this paper is to explore two selection mechanisms which tackle such a restriction. To this end, we perform a study which compares the tournament-based and the autonomous agents with a Simple Genetic Algorithm (SGA), in a Symmetric Multiprocessing (SMP) node scenario.

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 overall model operation and focuses on the implementation of the two selection mechanisms (sections 3.1 and 3.2). Section 4 explains the experimental setup and section 5 deal with the obtained results. Finally, section 6 reaches some conclusions and presents some future works.

2. RELATED WORK

Due to the diversity of fields that this study involves, it is convenient to revise them in order to set the scope of the work.

Concerning development of P2P distributed computing systems, there are some frameworks such as:

- DREAM [1], which focuses in distributed processing of EAs and uses the P2P network DRM.
- G2DGA [2], equivalent to the previous. It centers on distributed genetic algorithms processing by the use of the network G2P2P.
- JADE (Java Agent Development Framework, available from `http://jade.cselt.it/`), a P2P system which includes agents as software components.

The mentioned DRM is an implementation of the newscast protocol [10]. This protocol has served as a guide for the proposed communication mechanism within this model. Newscast is an epidemic approach where every node shares local information with its neighbourhood by selecting a node from it with uniform probability each certain time (refresh rate). Our communication model is inspired by such a protocol. However, our model considers a dynamic refresh rate which depends on the QoS parameters: latency and bandwidth.

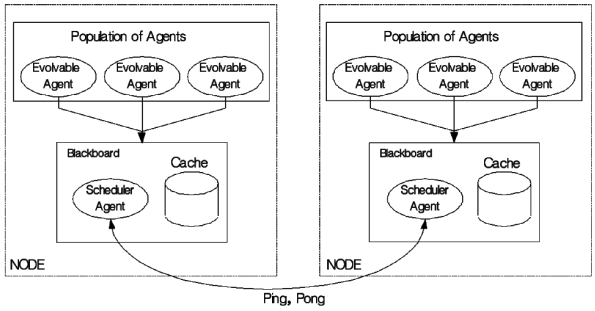

Figure 1: Overall architecture of the model

There are also several approaches to evolutionary computation using multi-agent systems: Vacher et al. present in [17], for instance, have presented a multiagent approach to solve multiobjective problems; recently, Lee [14] has also presented a multi-agent synchronous evolutionary system, which uses the JADE middleware mentioned above.

There are some works regarding optimization of parallel evolutionary algorithms; Viveros and Barán [18] propose the combination of parallel evolutionary algorithms with local optimization functions which depends on processor capacities in heterogeneous computational systems. The authors have related works on this field [8], shows that the number of parallel executions must be equivalent to the number of available processors in order to equilibrate computational effort and algorithmic results [6] report the benefits of considering population size adjustment on runtime. Finally some of the authors of this paper present in [11], another agent-based model where the load of every evolutionary computation experiment is self-adaptive depending on the architecture where it is executed, yielding more efficient results than the classical sequential approach.

In this paper we explore two selection mechanisms for our distributed evolutionary algorithm proposal which intends to be a step toward future models allowing real self-adaptation in dynamic environments such as P2P systems.

3. OVERALL MODEL DESCRIPTION

The overall architecture of our Evolvable Agent Model is depicted on figure 1. It consists of a group of Evolvable Agents (each one running on its own thread) whose main design objective is to carry out the principal steps of evolutionary computation: selection and variation (crossover and mutation). Obviously, the key element here is the locally executable selection. Crossover and mutation never involve many individuals, but selection in EAs usually requires a comparison among all individuals in the population. Consider, for example, roulette wheel or rank-based selection.

The agents know the environmental status by means of

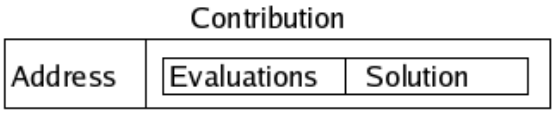

Figure 2: Format of a cache entry. It provides the following information about a foreign node: Address, number of evaluations performed and one individual of its population termed solution

a blackboard mechanism [15]. The blackboard allows the interchange of information between agents (Agent-Agent) or with cache (Agent-Cache).

Within this work we will focus on the Agent-to-Agent communication in a single node scenario as we intend to prove that, even in a single node, this novel evolutionary scheme performs as well or better than traditional, generational, synchronous evolutionary algorithms. Agent-to-Cache and the underlying gossip communication mechanism has been evaluated in [12]. The functional analysis of this system will be presented below.

The blackboard implements a Scheduler Agent that allows information spread among nodes in a gossip style. The messages used among nodes are called *contributions* and their structure matches with a cache entry (figure 2). Thus, instead of the classical view of a population of solutions managed by an oracle, this model proposes a population of (more or less) autonomous agents, each one representing a solution.

Algorithm 1 Scheduler Agent

$\Delta T \Leftarrow$ 1 sec.
loop
  sleep ΔT
  Node $\Leftarrow$ Random selected node
  Sol $\Leftarrow$ Selected Solution in P_{agents}
  Contribution $\Leftarrow Num_{Evaluations}$,Sol
  Ping (Node,Contribution)
end loop

Algorithm 2 Ping Handler

Require: Node, Contribution
  Cache(Node)$\Leftarrow$ Contribution
  Pong(Node,OK)

Algorithms 1,2 and 3 show the pseudo-code of the main tasks in the communication process which build the overlay network. Each blackboard maintains a cache with a maximum of one entry per node in the network. Each entry follows the contribution format (Figure 2). The cache indexes the entries with the Address field. Therefore, the newest contributions replaces the oldest ones. This process leads the removal of obsolete individuals and allows a global evolution in a decentralized environment. The scheduling mechanism is carried out by each node as explained next:

- **Algorithm 1** Each ΔT time, a node (the current node) selects another node with uniform probability to establish communication. Current node sends an application level Ping message to the selected node with information about a random solution in the population of agents (P_{agents}) in a contribution format (Figure 2).
- **Algorithm 2** The selected node stores that solution in its cache and sends back an acknowledge message (Pong).
- **Algorithm 3** At the arrival of the Pong, the current node updates its refresh rate (ΔT) with the time spent in the operation.

Algorithm 3 Pong Handler

Require: Node
  $\Delta T \Leftarrow$ Time used answering the Ping

Next we present the two possible methods of an Evolvable Agent under study. They differ in the selection mechanism. In both cases, selection is locally executable, in the sense that it does not require direct comparisons with all individuals in the population.

The model using the Basic Evolvable Agent is close to standard evolutionary algorithms. Local selection is performed by tournament selection, cf. Section 3.1. The simplicity of this scheme makes it appropriate as the base line for comparison. The model using the Experimental Evolvable Agent (Section 3.2) breaks with the traditional EA models more radically. Local selection is performed by autonomous selection and population updates are made asynchronous by allowing the addition/removal of a new/old individual without keeping the population size constant. This causes an extra challenge to EA designers, because in principle the population could grow beyond feasible range or shrink to complete extinction. This necessitates further research in order to define mechanisms (other than trial-error tuning of parameters) to maintain the population size between desirable ranges and according with available resources. Therefore, we give it the status of experimental agent at this stage.

3.1 Basic Evolvable Agent with Tournament Selection

Algorithm 4 Evolvable Agent with Tournament Selection

$S_t \Leftarrow$ Initialize Agent
Register Agent on the blackboard
loop
  Sols $\Leftarrow$ Selection(k, Blackboard)
  $S_{t+1} \Leftarrow$ Recombine(Sols,P_c)
  $S_{t+1} \Leftarrow$ Mutate(S_{t+1}, P_m)
  $S_{t+1} \Leftarrow$ Evaluate(S_{t+1})
  if S_{t+1} better than Blackboard.BestSol **then**
    Blackboard.BestSol $\Leftarrow S_{t+1}$
  end if
  if S_{t+1} better than S_t **then**
    $S_t \Leftarrow S_{t+1}$
  end if
end loop

Algorithm 4 shows the pseudo-code of an Evolvable Agent which uses Tournament Selection. The agent owns a solution (S_t) which it tries to evolve. The selection mechanism works as follows: Each agent selects k (k = tournament

size) solutions among other agents' current solutions and solutions stored in cache (which are migrants from network nodes) with uniform probability by means of the blackboard. The two best solutions in k are stored in "Sols" ready to be recombined by a crossover operator. The crossover returns a single solution S_{t+1} that is mutated and evaluated. If the newly generated solution S_{t+1} is better than the old one S_t, it becomes the current solution. Finally, Blackboard maintains global elitism by storing the best solution found so far in Blackboard.BestSol.

3.2 Experimental Evolvable Agent with Autonomous Selection

Algorithm 5 shows the pseudo-code of an Evolvable Agent which uses Autonomous Selection. This kind of agent differs from the previous one (described in Algorithm 4) in the ability of autonomously determining its own survival (survive(S_t)) and child production (fertile(S_t)). The individual survival and fertility are determined by heuristic decisions (explained below) about whether it is "good enough" to remain in the population (survival) and to reproduce (fertility). If the agent survives, it will remain in the population of agents otherwise it disappears. Furthermore, a surviving agent can be fertile in order to generate a child that becomes a new evolvable agent in the population (new Evolvable Agent (S_{t+1})). A child does not replace either of its parents, thus the population size can change.

$$sig_{m,s}(\Delta f(x)) = \frac{1}{e^{-m(\Delta f(x)-s)}} \quad (1)$$

Survival and fertility probabilities (respectively P_s,P_f) depend on the sigmoid function (1). This sigmoid yields large probabilities when $\Delta f > 0$ and small probabilities when $\Delta f < 0$ as illustrated in Figure 3. Parameter m (multiplier) corrects the sharp shape of the sigmoid and s (shift) shifts the sigmoid left or right. This way, P_s and P_f differ on the parameters m and s. Survive function uses m_s (multiplier survive function), s_s(shift survive function) and Fertile function m_f (multiplier fertile function), s_f(shift fertile function).

Since each agent takes these decisions locally it is necessary to supply it with some global status information. It is provided by the fitness deviation $\Delta f(x) = f(x) - \overline{f}$ in which $f(x)$ is the fitness of the current solution x (e.g. $f(S_t)$) and $\overline{f}$ is the mean fitness of the global population (it could be other class of global estimation).

Algorithm 5 Evolvable Agent with Autonomous Selection

```
S_t ⇐ Initialize Agent
Register Agent on the blackboard
while isAlive do
  if not survive(S_t) then
    isAlive ⇐ false
  else if fertile(S_t) then
    S_aux ⇐ Random Solution
    S_{t+1} ⇐ Recombine(S_aux,S_t,P_c)
    S_{t+1} ⇐ Mutate(S_{t+1}, P_m)
    S_{t+1} ⇐ Evaluate(S_{t+1})
    if S_{t+1} better than Blackboard.BestSol then
      Blackboard.BestSol ⇐ S_{t+1}
    end if
    new Evolvable Agent (S_{t+1})
  end if
end while
```

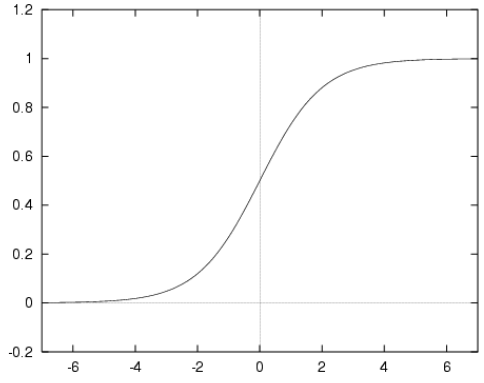

Figure 3: Sigmoid curve shape

4. EXPERIMENTAL SETUP

We have carried out an empirical investigation over the Evolvable Agent Model by comparing between the model variants using Basic Evolvable Agents and Experimental Evolvable Agents in a SMP node scenario. Here we also include the Simple Genetic Algorithm (SGA) as a baseline result.

As a test problem we have chosen the Travelling Salesman Problem (TSP) [13]. The TSP is a classical combinatorial optimization problem widely used to test evolutionary algorithms [20]. In this problem there is a set of $N = 1, \ldots, n$ cities which have to be visited once in such a manner that the path forms a graph cycle that minimizes the travelled distance. We have selected three symmetrical instances with different complexities: bier127, d198 and lin318, extracted

Test-bed	
Node Processor	AMD Athlon(tm) X2 4200+
Operating System	Linux 2.6.17-1.2142_FC4smp
Java version	J2RE (build 1.5.0_06-b05)
Operators and Rates	**All EAs variants**
Crossover	Order crossover(OX)
Mutation	2-Opt mutation
Termination condition	Max. number of eval.
Probability of crossover	0.7
Probability of mutation	0.1
Evolutionary Algorithm	**SGA**
Population size	32
Tournament size	2
Evolutionary Algorithm	**Basic**
Population size	32
Tournament size	7
Evolutionary Algorithm	**Experimental**
Initial population size	32
Maximum population size	500
m_s	1000
s_s	0.5
m_f	60
s_f	0.95

Table 1: SMP Node Scenario: Test-bed and Operator rates. All EA variants share the main evolutionary operators and rates. SGA stands for Simple Genetic Algorithm, Evolvable Agent is labelled *Basic and Evolvable Agent with Autonomous Selection* Experimental.

from TSPLIB[1]. It is important to note that our research objective is not to establish a TSP-solver outperforming existing ones. Rather, the TSP is just used as a test problem to perform an empirical study among variants of our algorithm.

This case study is intended to provide experimental data on the solution quality (accuracy) and algorithm speed obtained by our two Evolvable Agent variants. The more classical variant uses tournament selection and the experimental one uses autonomous selection and decentralized population updates. Table 1 shows the general settings for this case study. As a test-bed, we have considered a single node with a Dual-Core processor. The test-bed has been deliberately chosen since Evolvable Agent thread based implementation, as explained in section 3, takes advantage of the application level parallelism in shared memory computers (dealing with a self-adaptive response to the heterogeneity found in P2P nodes). These tests focus on an efficiency and accuracy analysis, rather than on scalability. Solution quality is compared using the Mean Best Fitness (MBF), while speed is again measured by the average time spent when computing a fixed number of 100000 evaluations. Both measures are calculated over 30 independents runs.

The case of tuning the Autonomous Selection Mechanism requires special attention. Since population size changes during a run, we set the initial size to 32 and the maximum number of individuals to an upper bound of 500. Based on experience we gained with preliminary experimentation

[1]http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ Accessed on January 2007

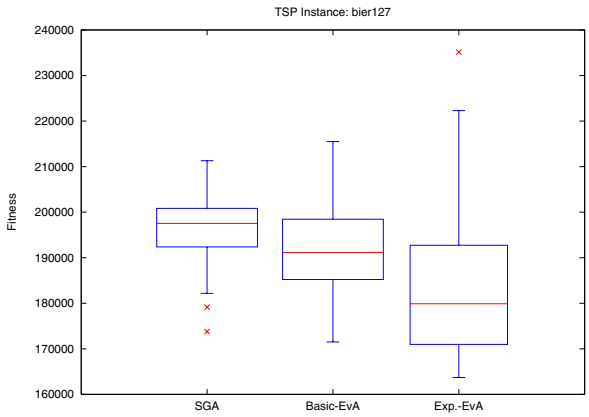

Figure 4: Box-plot representation of the best fitness obtained for the bier127 instance. SGA stands for Simple Genetic Algorithm, Basic-EvA and Exp.-EvA for Basic and Experimental Evolvable Agent

we scaled the TSP tour lengths so that we got Δf values between -1 and 1 and the multipliers and shifts have been tuned to $m_s = 1000$ $s_s = 0.5$ $m_f = 60$ $s_f = 0.95$.

5. EXPERIMENTAL RESULTS

5.1 Algorithmic comparison

The results of comparing tournament and autonomous selection as the locally executable selection mechanisms for the Evolvable Agent model are depicted in figures 4, 5 and 6. To provide a generic benchmark, we also show the outcomes obtained with a simple Genetic Algorithm (SGA).

We have carried out a t-Student analysis on results for each problem instance (bier127, d198 and lin318) coupling the three algorithms under study. They are all different with a significance of 99% except in d198 instance (figure 5) for the Simple Genetic Algorithm and the Autonomous Selection which is different with a 95% significance.

From the bier127 instance (figure 4) with the slightest workload to the lin318 (figure 6) with the heaviest one, we can observe how the Evolvable Agent with tournament selection gradually outperforms the rest. In terms of measures of central location i.e. the mean, the Evolvable Agent with autonomous selection would outperform the SGA, but the spread in the distribution shape makes difficult a comparison. Such a shape could be an effect of the on-the-fly population size adjustment which introduces an extra non-deterministic effect on results. We focus on that mechanism in Section 5.3.

5.2 Scalability in the SMP node

Table 2 shows the time that the different algorithms spent in processing a fixed computational effort of 100000 evaluations when solving the three problem instances. The Evolvable Agent model using tournament selection turns out to be the fastest in all instances (speed-up close to a factor 2 regarding the SGA) while the Evolvable Agent using autonomous selection is slower or faster than SGA depending on the instance.

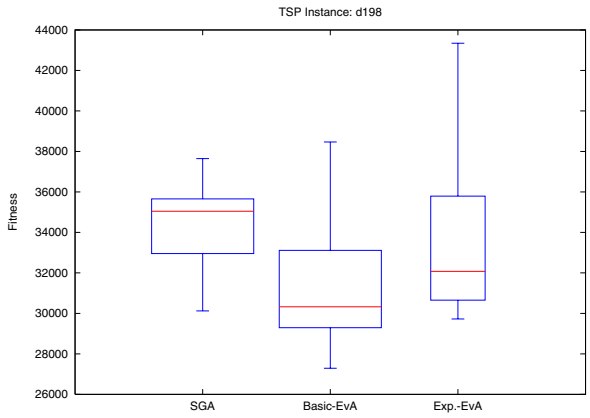

Figure 5: Box-plot representation of the best fitness obtained for the d198 instance. SGA stands for Simple Genetic Algorithm, Basic-EvA and Exp.-EvA for Basic and Experimental Evolvable Agent

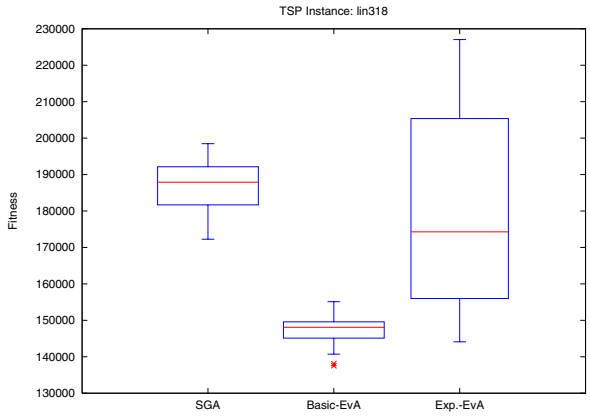

Figure 6: Box-plot representation of the best fitness obtained for the lin318 instance. SGA stands for Simple Genetic Algorithm, Basic-EvA and Exp.-EvA for Basic and Experimental Evolvable Agent

TSPInstance	EA	Time(seconds)$\pm\sigma$
bier127	SGA	15.6 $\pm$ 0.3
	Basic-EvA	**9.75 $\pm$ 0.344**
	Exp.-EvA	23.2 $\pm$ 1.93
d198	SGA	37.7 $\pm$ 1.5
	Basic-EvA	**19 $\pm$ 0.88**
	Exp.-EvA	40.7 $\pm$ 3.98
lin318	SGA	98.3 $\pm$ 4.7
	Basic-EvA	**47.64 $\pm$ 2.6**
	Exp.-EvA	79.9 $\pm$ 5.56

Table 2: Time comparison of SGA, Basic and Experimental Evolvable Agents when solving 3 instances of the TSP. Effort is fixed to 100000 evaluations.

These results show how the thread-based implementation of the Evolvable Agent Model takes implicit advantage of the SMP computers regarding sequential implementation of the canonical form of a SGA. SGA is a single-threaded process which is able to execute in a single processor. The number of threads in the Evolvable Agent Model is determined by the population size, such a kind of parallelism occurs at application level since the operating system balances the threads among the available processors. Unfortunately the experiments conducted on a SMP Dual-Core processor are just a proof of concept instead of a real scale-up analysis. Future works will have to address this question.

5.3 The Experimental Evolvable Agent: population size and fitness curve

The Evolvable Agent using autonomous selection present an extra non-deterministic effect on results which we try to analyze within this section. This effect is caused by the on-the-fly population adjustment that takes place. We consider a run of the d198 problem instance depicted in figures 7 and 8. Figure 7 illustrates how the fitness of the population evolves over the time by plotting the average and best fitness. For the same execution, figure 8 depicts the population sizes, either the total size and the number of fertile individuals.

There are two key observations to be made.

- First, within the range of [0,20000) evaluations, figure 8 depicts how the global population and fertile individuals shrink and grow while figure 7 shows that the best and average fitness remain close. Therefore, Δf values for the sigmoid function must be around 0 within this fragment of execution which allows that the best individuals become fertile and the worst are erased maintaining the selection pressure balanced.

- Second, from 20000 evaluations ahead, the population size explode to the artificial bound of 500 while fertile individuals decrease with significance. The selection pressure increases since just few best individuals are fertile which affects to diversity.

This study reveals that further research is needed in order to control the selection pressure when using autonomous selection as selection mechanism for the Evolvable Agents. Nevertheless, the comparison carried out in Section 5.1 shows how the autonomous selection, even at this early stage, can obtain better results than SGA and better than Evolvable Agents with tournament selection in some of the instances.

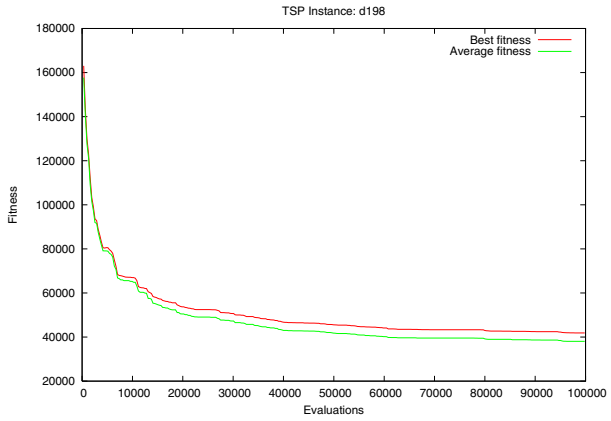

Figure 7: Experimental Evolvable Agent. Best/average fitness curves for the d198 instance

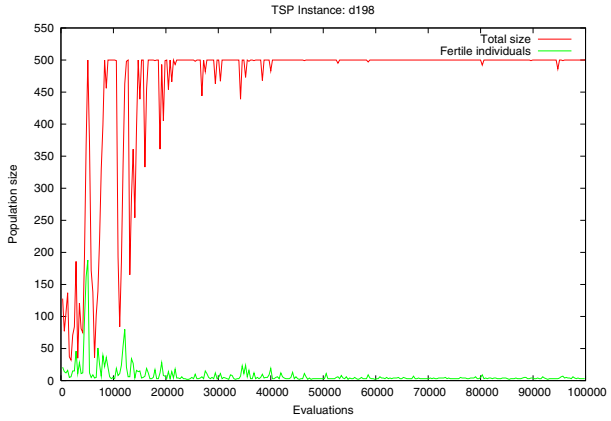

Figure 8: Experimental Evolvable Agent. Population size and fertile individuals for the d198 instance

6. CONCLUSIONS AND FUTURE WORKS

In this paper we present an Agent-based approach towards a fully distributed EA model. The model is designed to deal with heterogeneous networks features and specially P2P networks. The evolution process consists in maintaining a population of agents that evolve single solutions. Each agent can access other agents' current solution in operations that needs more than one individual (e.g. selection) by means of the blackboard mechanism described in section 3.

From the proposed experiments we conclude that:

- The structural changes that the model proposes on the EA make it faster in obtaining a better fitness than the canonical SGA for the same number of evaluations, running in a single node.
- One major aspect, so far outlined in the paper, is that P2P networks are composed by a wide range of heterogeneous nodes. As can be gleaned from the case study in a Dual-Core processor, Evolvable Agents take implicit advantage of the application level parallelism in shared memory computers (dealing with a self-adaptive response to the heterogeneity found in P2P nodes).
- Concerning the two Evolvable Agent methods under study, we can conclude that Tournament is more efficient than Autonomous Selection at least with the current implementations (since a profiling study of this last could reveal several ways of optimization). However, the algorithmic results are dependent on the problem instance without a clear winner.
- This approach is a proof of concept towards a distributed EA model where scalability and quality results were possible both together.

Finally, beyond these preliminary tests, we consider that population size adjustment on runtime should be considered as a natural mechanism for reaching adaptation to dynamic P2P networks in which available resources are changing through the time. Therefore our future research will focus on the improvement of Evolvable Agents with Autonomous Selection. Hence, we would have an adequate adjustment of the population size if we could control the selection pressure by self-adapting the m_s, f_s, m_f and s_f parameters. Such a dynamic population will allow taking advantage of the P2P available resources by the use of correct load balancing policies.

Future works will have to consider the experimentation in large-scale networks where further conclusions can be reached respecting scalability limitations, adaptation to heterogeneity and algorithmic effects of having a real network with high latency links. Within this line we plan to implement the model into a P2P framework such as DREAM [1] which shares its main design objectives.

Acknowledgements

This work has been partially supported under the Project NadeWeb (TIC2003-09481-C04).

7. REFERENCES

[1] M. Arenas, P. Collet, A. Eiben, M. Jelasity, J. Merelo, B. Paechter, M. Preuss, and M. Schoenauer. A framework for distributed evolutionary algorithms. In J. M. Guervós, P. Adamidis, H. Beyer, J. F.-V. nas, and H. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002. Proceedings*, number 2439 in Lecture Notes in Computer Science, LNCS, pages 665–675. Springer-Verlag, 2002.

[2] J. Berntsson. G2DGA: an adaptive framework for internet-based distributed genetic algorithms. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 346–349, 2005.

[3] E. Cantú-Paz. Topologies, migration rates, and multi-population parallel genetic algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 91–98, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[4] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[5] D. Doval and D. O'Mahony. Overlay networks: A scalable alternative for P2P. *IEEE Internet Computing*, 7(4):79–82, July/August 2003.

[6] A. Eiben, E. Marchiori, and V. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 41–50, Birmingham, UK, September 2004. Springer-Verlag.

[7] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[8] I. Hidalgo and F. Fernández. Balancing the computation effort in genetic algorithms. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1645–1652. IEEE Press, 2005.

[9] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

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

[11] J.L.J.Laredo, P. Castillo, A. Mora, and J. Merelo. Estudio preliminar sobre autoadaptación en agentes evolutivos sobre arquitecturas heterogéneas. In *XVII Jornadas de Paralelismo – XVII JP*, pages 389–394, September 2006.

[12] J. Laredo, E. Eiben, M. Schoenauer, P. Castillo, A. Mora, F. Fernández, and J. Merelo. Self-adaptive gossip policies for distributed population-based algorithms, 2007. http://arxiv.org/abs/cs/0703117.

[13] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys. *The travelling salesman problem: A guided tour of combinatorial optimization.* New York: Wiley and Sons, 1985.

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

[15] H. P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, 1986. Cited By (since 1996): 101.

[16] R. Steinmetz and K. Wehrle. What is this peer-to-peer about? In R. Steinmetz and K. Wehrle, editors, *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, pages 9–16. Springer, 2005.

[17] J.-P. Vacher, A. Cardon, F. Lesage, and T. Galinho. Genetic Algorithms in a Multi-Agent System. In *Proceedings IEEE International Joint Symposia on Intelligence and Systems*, pages 17–26, Rockville, MD, USA, 1998.

[18] E. Viveros and B. Barán. Algoritmos genéticos asíncronos combinados para una red heterogénea de computadoras. In *Conferencia Internacional de Ciencia y Tecnología para el Desarrollo*, 1997. http://www.cnc.una.py/cms/invest/download.php?id=46205,66,1.

[19] S. Voulgaris, M. Jelasity, and M. van Steen. A robust and scalable peer-to-peer gossiping protocol. In Moro, Sartori, and Singh, editors, *Agents and Peer-to-Peer Computing*, volume 2872 of *Lecture Notes in Computer Science (LNCS)*, pages 47–58. Springer Berlin / Heidelberg, 2004.

[20] L. Wang, A. Maciejewski, H. Siegel, and V. Roychowdhury. A comparative study of five parallel genetic algorithms using the traveling salesman problem. In *IPPS: 11th International Parallel Processing Symposium*, pages 345–349. IEEE Computer Society Press, 1998.