ADVANTAGES OF MULTIAGENT SYSTEMS IN OPTIMIZATION:
CASE OF THE DISTRIBUTED GENETIC ALGORITHM

Sadok Bouamama*, khaled Ghédira **

SOIE (ex Uriasis)
SOIE/ISG/Université Tunis
B. 204, Département d'Informatique
41, rue de la liberté, 2000 cité Bouchoucha. Tunisie
* sadok.bouamama@laposte.net, ** khaled.ghedira@isg.rnu.tn

Abstract

This paper aims to determine the advantages of the Multi-agent use in an optimization process. For this it will focus on the case of genetic algorithms which are known to be “expensive” in time. Especially it presents and studies a distributed Guided Genetic Algorithm (DGGA) dealing with Maximal Constraint Satisfaction Problems. This algorithm consists of dynamically created agents cooperating in order to satisfy the maximal number of constraints. Each agent performs its own GA, guided by both the template concept and the Min-conflict-heuristic, on a sub-population composed of chromosomes violating the same number of constraints. The objective here is to determine the effect of each one of the improvement factors (guidance by both min-conflict heuristic and the template concept and the same fitness chromosomes species) if or not they are combined with multi-agent approach. So here we will proceed by experimental comparison of a family of genetic algorithms having included all possible combination of factors described above. The result shows clearly that, thanks to agents interactions, the use of multi-agent systems intensifies the betterment given by the other improvement factors.

Key Words
Max_CSPs, multi-agent systems, genetic algorithms, Min-conflict-heuristic, template concept.

1. Introduction

CSP formalism consists of variables associated with domains and constraints involving subsets of these variables. A CSP solution is an instantiation of all variables with values from their respective domains. The instantiation must satisfy all constraints. In addition to their simple and generic formalization, CSPs are omni-present in many real-life problems ranging from industrial applications, such as scheduling and planning to school examples, such as n-queens and graph coloring problems. A CSP solution, as defined above, is costly to get and does not necessarily exist in every problem. In such cases, one had better search an instantiation of all variables that satisfies the maximal number of constraints. Such problems, called Maximal CSPs and referred to as Max-CSPs, make up the framework of this paper.

Max-CSPs have been dealt with by complete or incomplete methods. The first ones, such as extended forward checking algorithm [17, 7], and branch and bound algorithms [17, 26] are able to provide an optimal solution. Unfortunately, this advantage is thwarted by the combinatorial explosion. The second ones, such as Simulated Annealing, Genetic Algorithms [13] and Tabu Search have the same property to avoid the trap of local optima. They also sacrifice completeness for efficiency. There is another incomplete but distributed method known as Distributed Simulated Annealing (DSA). It has been successfully applied to several fields: static Max-CSP [10], dynamic Max-CSP [8], Resource allocation problem [Ghédira 95] and graph partitioning problem [4]. As DSA outperforms the centralized Simulated Annealing in terms of both optimality and quality, the same idea is adopted for Centralized Genetic Algorithms (CGAs) which are especially known to be expensive. The result was the distributed guided genetic algorithm for Max_CSPs [11]. Our interest in GAs is also motivated by their proven usefulness in hard optimization problems [24], solving multiprocessor scheduling problems [Michael & Al 99, 25], optimal image enhancement [20], etc.

This paper is organized as follows: The next subsection recalls presents the Distributed Guided Genetic Algorithm: the basic concepts, the agent structure and finally the global dynamic. The following one details both experimental design and results used to study the effect of multiagent systems use on GAs. Finally, concluding remarks and possible extensions to this work are proposed.

2. Distributed Guided Genetic Algorithm

2.1. Basic principles

The relationship between both genetic and CSP formalisms is outlined in figure 1. Each chromosome (respectively gene) is equivalent to CSP potential solution (respectively variable). Moreover, each allele corresponds to a value. On the other hand, each chromosome is attached to a template [24] that is made up of weights referred to as templatei,j. Each one of them corresponds to geneij where i refers to the chromosome and j to the position. δij represents the number of violated constraints by geneij. These weights are updated through the penalty operator.
The DGGA approach draws basically on the concept of both species and ecological niches. The species consists of several organisms having common characteristics whereas the ecological niche represents the task performed by a given species. Goldberg sets that the sexual differentiation based on specialization via both the building of species and the exploitation of ecological niches provides good results [13]. A certain number of methods have been settled in order to favor the building of ecological niches [DEJONG 89, 12] in GAs.

| Variables: | $V_1, V_2, V_3, V_4$ |
| Di: Domain of $V_i$ | $D_i = \{1, 2, 3\}$ for all $i$ in $\{1, 2, 3, 4\}$ |
| Constraints: | $C_1 : V_1 + V_3 > 4$, $C_2 : V_2 - V_4 \geq 2$, $C_3 : V_3 - V_2 = 2$ |

**Figure 1. Relationship between Genetic and CSP formalisms**

So, the idea here is to partition the initial population into sub-populations and to assign each one of them to an agent called Specie agent. A given sub-population consists of chromosomes violating the same number of constraints. This number, say $n$, is called the specificity of the Specie agent Specie$_n$. Thus, DGGA involves at most $n_c$ (total number of constraints) Specie agents in interaction, in order to reach a total satisfaction namely the maximal number of satisfied constraints. For this reason, each Specie agent performs its own GA which has been enriched by the Min-conflict-heuristic and the template concept. An intermediary agent is necessary between the society of Specie agents and the user, essentially to detect the best partial solution reached during the dialogue between the Species. This agent, called Interface, may also possibly create new Specie agents.

### 2.2. Agent Structure

Each agent has a simple structure: its acquaintances (the agents it knows and with which it can communicate), a local knowledge composed of its static and dynamic knowledge, and a mailbox where it stores the received messages to be later processed one by one.

#### 2.2.1. Specie agent

A Specie agent has got as acquaintances the other Specie agents and the Interface agent. Its static knowledge consists of the CSP data (i.e. the variables, their domains of values and the constraints), the specificity (i.e. the number of violated constraints) and its local GA parameters (mutation probability, cross-over probability, number of generations, etc.). Its dynamic knowledge takes components as the population pool which varies from one generation to another (chromosomes, population size).

#### 2.2.2. Interface agent

An Interface agent has as acquaintances all the Specie agents. Its static knowledge consists of the CSP data. Its dynamic knowledge includes the best chromosome (i.e. best partial solution).

### 2.3. Global agents dynamic

The Interface agent randomly generates the initial population and then partitions it into sub-populations accordingly to their specificities. After that the former creates Specie agents to which it assigns the corresponding sub-populations. Then the Interface agent asks these Specie agents to perform their optimization processes (figure 2 line 3). So, before starting its optimization process, i.e. its behavior (figure 3), each Specie agent, Specie$_n$, initializes all templates corresponding to its chromosomes (figure 3 line 3). After that it carries out its genetic process on its initial sub-population, i.e. the sub-population that the Interface agent has associated to it at the beginning. This process, which will be detailed in the following subsection, returns a sub-population “pop” (figure 3 line 4) that has been submitted to the crossing and mutating steps only once, i.e. corresponding to one generation. For each chromosome of pop, Specie$_n$ computes the number of violated constraints “nvc” (figure 3 line 6). Consequently, two cases may occur. The first one corresponds to a chromosome violating the same number of constraints of its parents. In this case, the chromosome replaces one of the latter randomly chosen (figure 3 line 8). In the second case is that this number (nvc) is different from (n), i.e. the specificity of the corresponding Specie$_n$. Then the chromosome is sent to another Specie$_{nvc}$ (figure 3 line 10) if such an agent already exists, otherwise it is sent to the Interface agent (figure 3 line 11). The latter creates a new agent having nvc
as specificity and transmits the quoted chromosome to it. Whenever a new Specie agent is created, the Interface agent informs all the other agents about this creation (figure 2 line 7) and then asks the new Specie to perform its optimization process (figure 2 line 3). Note that message processing is given a priority. So, whenever an agent receives a message, it stops its behavior, saves the context, updates its local knowledge, and restores the context before resuming its behavior.

1. \( m \leftarrow \text{getMsg (mailBox)} \)
2. case \( m \) of
3.   optimization-process (sub-population) :
4.      apply-behavior (sub-population)
5.   take-into-account (chromosome) :
6.      population-pool \( \leftarrow \) population-pool \( \cup \) \{chromosome\}
7.   inform-new-agent (Specie\(_{nvc}\)) :
8.      list-acquaintances \( \leftarrow \) list-acquaintances \( \cup \) \{Specie\(_{nvc}\}\}
9.   stop-process : stop-behavior

Figure 2. Message processing relative to Specie\(_{nvc}\)

If all the Specie agents did not meet a chromosome violating zero constraints at the end of their behavior, they successively transmit one of their randomly chosen chromosomes, linked to its specificity to the Interface agent. The latter determines and displays the best chromosome namely the one which violates the minimal number of constraints.

2.3.1. The detailed genetic process

This process differs from the canonical GA described [12] of the use of both templates and min-conflict-heuristic [9, TSANG & AL 99]. It starts with determining the mating-pool which consists of pairs of chromosomes, randomly selected by means of the matching procedure (figure 4 line 1). Out of each pair of chromosomes, the cross-over operator produces a new child as described in figure 6. The child inherits the best genes, i.e. the “lighter” ones, from its parents. The probability, for a parent chromosome, \( (i = i_1 \) or \( i_2) \), where \( \text{sum} = \text{template}_{i_1,j} + \text{template}_{i_2,j} \) to propagate its gene\(_{i,j}\) to its child chromosome is equal to \( 1 - \text{template}_{i,j}/\text{sum} \). This confirms the fact that the “lighter” genes, i.e. having the smallest number of violated constraints, are more likely than the other to be passed to the child. For each one of its chromosomes selected according to the mutation probability \( P_m \) (figure 8 line 2 ), Specie\(_n\) uses the min-conflict-heuristic (figure 8 line 3) first to determine the gene (variable) involved in the maximal number of violated constraints (see figure 7 line 1), secondly to select from this gene domain the value that violates the minimal number of constraints (figure 7 lines 2-7) and finally to instantiate this gene with this value (figure 7 line 8).

It is tempting to note that the change of both gene (variable) and allele (value) is dynamic. Indeed, it depends on other variable values relative to the same chromosome. So different variables (see figure 7 line 1) as well as different values may be chosen (figure 7 lines 2-7). Thanks to these changes, new values may appear and consequently new chromosomes (that were not in the initial population) may occur. Thereby, the population will be more diversified thus enhancing the search and giving it more chance to converge.

If the obtained chromosome does not violate any constraint (figure 8 line 4), Specie\(_n\) asks the Interface agent to stop the whole process (figure 8 line 5). Thus, the latter first requests all the Specie agents to stop their behaviors (figure 2 line 9) and then displays the quoted chromosome. Otherwise, this chromosome is added to the offspring pool and Specie\(_n\) carries on with its behavior. Note that the stopping message also occurs in the crossing step (figure 5 line 8).

Here we describe the syntax used in the figures:
- \( \text{sendMsg (sender, receiver,'message') } \): ‘message’ is sent by “sender” to “receiver”.
- \( \text{getMsg (mailBox)} \): retrieves the first message in mailBox.

Apply-behavior (initial-population)
1. init-local-knowledge
2. for \( i := 1 \) to number-of-generations do
3. template-updating (initial-population)
4. pop \( \leftarrow \) genetic-process (initial-population)
5. for each chromosome in pop do
6. nvc \( \leftarrow \) compute-violated-constraints (chromosome)
7. if (nvc = n) then replace-by (chromosome)
8. else if exist-ag (Specie\(_{nvc}\)) then sendMsg (Species\(_n\), Species\(_{nvc}\), 'take-into-account (chromosome)' )
9. else sendMsg (Species\(_n\), Interface, 'create-agent (chromosome)' )
10. sendMsg (Species\(_n\), Interface, 'result (one-chromosome, specificity)' )

Figure 3. Behavior relative to Species\(_n\)
3. Experimentation

3.1 Introduction

The goal of our experimentation is to compare a family of genetic algorithms using or not multiagent systems. The implementation has been done with ACTALK [BRIOT 89], a concurrent object language implemented above the Object Oriented language SMALLTALK-80.
3.2 Experimental design

Our experiments are performed on binary CSP-samples randomly generated. The generation is guided by classical CSP parameters: number of variables (n), domain size (d), constraint density p (a number between 0 and 100% indicating the ratio between the number of the problem effective constraints to the number of all possible constraints, i.e. a complete constraint graph) and constraint tightness q (a number between 0 and 100% indicating the ratio between the number of forbidden pairs of values (not allowed) by the constraint to the size of the domain cross product). As numerical values, we use n = 20, d = 20. Having chosen the following values 0.1, 0.3, 0.5, 0.7, 0.9 for the parameters p and q, we obtain 25 density-tightness combinations. For each combination, we randomly generate 30 examples to obtain 750 examples. Moreover and considering the random aspect of genetic algorithms, we have performed 10 experimentations per example and then we take the average without considering outliers. For each combination density-tightness, we also take the average of the 30 generated examples. Regarding GA parameters, all implementations use a number of generations (NG) equal to 10, an initial-population size equal to 1000, a cross-over probability equal to 0.5, a mutation probability equal to 0.2 and a random replacement. The performance is assessed by the two following measures:

- Run time: the CPU time requested for solving a problem instance,
- Satisfaction: the number of satisfied constraints.

The first one shows the complexity whereas the second tells about the quality. In order to have a quick and clear comparison of the relative performance of two approaches A1 and A2, we compute ratios of A1 and A2 performance using the Run time and the satisfaction, as follows:

\[
\text{CPU-ratio} = \frac{\text{A1-Run-time}}{\text{A2-Run-time}} \\
\text{Satisfaction-ratio} = \frac{\text{A2-Satisfaction}}{\text{A1-Satisfaction}}.
\]

Thus, A1 performance is the numerator when measuring the CPU time ratios, and the denominator when measuring satisfaction ratio. Then, any number greater than 1 indicates superior performance by A2. Consequently the advantage of specialization based distribution. Note that this kind of comparison has already been proven to be efficient [11].

4. How to determine MAS effects on GAs

4.1 Introduction

DGGA uses a society of dynamically created agents cooperating in order not only to provide an optimal solution for Max-CSP but also to minimize temporal complexity of Genetic Algorithms (GAs). Each agent is responsible for a sub-population of chromosomes, that it handles by a local GA guided by both the template concept (tc) and the min-conflict-heuristic (mch). All the chromosomes of a given Specie agent violate the same number of constraints (having the same fitness). Experimentally compared with a centralized implementation of “GA + tc + mch” in terms of quality and CPU time, DGGA has been always proven to be better [11].

It’s clear that DGGA uses simultaneously three improvement factors that are:
- Distribution using multi-agent systems,
- Guidance with "tc" and "mch",
- Use of the same fitness.

In order to analyze which improvement factor gave the DGGA its progress and what the surplus given to by MAS use, we proposed to study the effect of each one of these factors. The idea is, then, to compare results given once trying the following approaches on the same Max_CSPs (randomly generated);
- DGGA (MAS + tc + mch + same fitness)
- DGGA_DF: which is a distributed GA where agents are responsible of a subset of chromosomes having different fitness. The idea is to send chromosomes randomly, at the end of every generation, to one of the existing subset. In this case the species theory will not be considered.
- DGA: which is a not guided distributed GA so both crossover and mutation mechanism are random ones i.e only mutation operator manages mutation and only crossover operator intervene in crossover.
- GGA (centralized GA having heterogynous chromosomes)

In each attempt we will take two approaches having only one difference, so that
- DGGA versus DGGA_DF determines the effect of using the same fitness
- DGGA versus DGA gives the effect of guidance
- DGGA_DF versus GGA treats distribution effects

4.2 Experimental results

● Same fitness using effects:

In this part, DGGA (MAS + tc + mch + same fitness) and DGGA_DF (chromosomes having different fitness) will be compared. Using chromosomes having the same fitness or not using them is the only difference between the two approaches. We compute ratios of DGGA and DGGA_DF performance using the Run time and the satisfaction, as follows:
CPU-time-ratio = DGGDF -Run-time / DGGF-Run-time  
Satisfaction-ratio = DGGF-Satisfaction / DGGDF -Satisfaction

Both figure 10 and figure 11 show the performance ratios from which we draw out the following results:

- From the CPU time point of view (figure 10), DGGA requires up to 1.8 times less for the most weakly constrained and most weakly tight set of examples. Nevertheless, and in some problems especially for the over-constrained and for the most strongly tight set of examples, the CPU time ratio is less than 1. CPU-time ratios average value is about 1.25.

- From the satisfaction point of view (figure 11), the DGGA always finds more satisfaction than DGGA_DF or the same one. It finds about 1.2 times more for the most strongly constrained and most tight set of problems. Satisfaction ratios value is about 1.3 and can attain 3 for the most weakly constrained and most weakly tight set of examples.

These experimentations had shown that the use of chromosomes having the same fitness by a genetic algorithm gives better results than the use of heterogeneous ones.

**Guidance effects:**

As we did in the last experimental part, we will try to find out which is better than the other: a GA using guidance with "tc" and "mch" or which does not. For this reason, DGGA versus DGA experimentation results will be treated. Let’s mention here that only guidance (with "tc" and "mch") makes the difference between the two algorithms; in fact DGA is a not guided distributed GA. Using the Run time and the satisfaction, Ratios of DGA and DGGA performance are computed as follows:

CPU-time-ratio = DGA-Run-time / DGGA-Run-time  
Satisfaction-ratio = DGGA-Satisfaction / DGA-Satisfaction.

Figure 12 shows that, the CPU-time ratio is greater than 1. This demonstrates that DGGA is better than DGA for the majority of the token problems set. The average value is equal to 1.17. This ratio reaches its maximum (1.7) for problems which densities are located around 0.1 and have some values less than 1.

**Distribution effects:**

Our main concern in this part is to determine distribution effects. Then a distributed approach and a centralized one will be compared; DGGA_DF versus GGA will treat distribution effects. It is clear that what differs the only the two GAs is only whether to use the distribution by multi-agent systems or not. Ratios of GGA and DGGA performance ratios will be considered as follows:

CPU-time-ratio = GGA -Run-time / DGGA-Run-time  
Satisfaction-ratio = DGGF-Satisfaction / GGA-Satisfaction.
Figure 14. CPU-time ratio

Figure 14 illustrate the CPU time point of view, and shows that DGGA_DF furnish results are twice as good as those given by the centralized version of the GA. Let us mention that this ratio attains it maximum (3.2) in case of the most strongly constrained (i.e. over-constrained) set of examples. So the distributed approach advantage is more significant for problems which tightness are located around 0.5 (see peaks in figure 14). The location and the sharpness of this area depend on the density parameter. This area corresponds to the transition phase. The transition covers the passage from under-constrained problems which are relatively easy to solve, to over-constrained problems that are relatively easy to prove insoluble [BARBARA 94]. The average value of these ratios is about 1.63.

From the satisfaction point of view shown in figure 15, the DGGA_DF always finds more satisfaction than GGA or same one. It finds about 2 times more for the most weakly constrained set of problems which are located in the area corresponding to the transition phase. Satisfaction ratios average is 1.13.

● The way to the best approach:

Last series of experimentations are done considering the same set of problems for the four algorithms at every trial. Thus we can determine the most efficient algorithm. We will consider figure 15 and figure 16 which represent the CPU-time average and the satisfaction average for the four approaches as well as for different values of density.

Figure 16 shows clearly that the DGGA CPU-time is the shortest one, otherwise this approach needs lesser time than the others. The centralized GA comes in the last rank as the approach having the greatest CPU-time. These results are clearly shown in area corresponding to the transition phase (i.e density value is 0.5). In figure 15 attention is focused on satisfaction; the number of satisfied constraints, when using DGGA approach is the greatest among all the others. Figure 16 and figure 17 demonstrate that the DGGA approach gave better results comparing to those tested approaches.

5. Conclusion

This paper has shown that Distribution by MAS, guidance by Min-conflict-heuristic and template concept and using species having same fitness chromosomes are improvement factors. So when used, these factors could better the result given not only in CPU-time but also in satisfaction.

While each one of these improvement factors can improve the result until 3 time, DGGA requires up to six times less than GGA[11]. Therefore, one may conclude that the combination of improvement factors gives better results than their separation. This is, in fact, the result of the interaction of agents in the used multi-agent system. Thanks to this interaction complexity time is reduced and better solution is given. This agents interaction intensifies the effect given by the other improvement factors, allowing, then, the reduction of CPU-time and the rise of satisfied constraints. No doubt further refinement of this work would allow its performance to be improved.
6. References


