

# A Dynamic Distributed Double Guided Genetic Algorithm for Optimization and Constraint Reasoning

Sadok Bouamama<sup>1</sup> and Khaled Ghedira<sup>2</sup>

<sup>1</sup> SOIE Laboratory, University of TUNIS  
*Sadok.Bouamama@ensi.rnu.tn*

<sup>2</sup> SOIE Laboratory, University of TUNIS  
*Ghedira.Khaled@isg.rnu.tn*

**Abstract:** D<sup>3</sup>G<sup>2</sup>A is a new multi-agent approach which addresses additive Constraint Satisfaction Problems ( $\Sigma$ CSPs). This approach is inspired by the guided genetic algorithm (GGA) and by the Dynamic distributed double guided genetic algorithm for Max\_CSPs. It consists of agents dynamically created and cooperating in order to solve problems with each agent performs its own GA. First, our approach is enhanced by many new parameters. These latter allow not only diversification but also escaping from local optima. Second, the GGAs performed agents will no longer be the same. This is stirred by NEO-DARWINISM theory and the nature laws. In fact our approach will let the agents able to count their own GA parameters. In order to show D<sup>3</sup>G<sup>2</sup>A advantages, the approach and the GGA are applied on the radio link frequency allocation problem (RLFAP) and on the randomly generated binary CSPs.

**Keywords:** Constraints satisfaction, genetic algorithms, guidance, NEO-DARWINISM theory.

## I. 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. This solution is costly to get and does not, necessarily, exist within every problem. In such cases, one has better assign a cost to each constraint and then search an instantiation of all variables that minimise the costs sum of violated constraints. Such problems are called additive CSPs and referred to as  $\Sigma$ CSPs[10],[12]. Unfortunately, real life problems can be more difficult in case we have to optimize an objective function and to satisfy all the problem constraints. These problems are referred to as CSOPs (for constraint satisfaction and optimisation problems)[11],[12]. CSOPs and  $\Sigma$ CSPs make up the framework to this paper.

$\Sigma$ CSPs and CSOPs are generally NP-hard. They have been dealt with by complete or incomplete methods. Complete methods are able to provide an optimal solution. Unfortunately, the combinatorial explosion thwarts this advantage. However incomplete methods, such as Guided Genetic Algorithms (GGA) [11], have the property of avoiding the trap of local optima, they also sacrifice completeness for efficiency.

There is an another incomplete but distributed GA based method from which arises the present work. This method is known as Distributed Guided Genetic Algorithm (DGGGA)[6]. It has been successfully applied to Max-CSP[1],[3]-[6]. This method outperforms the centralized Genetic Algorithms, which are especially known to be expensive in time. The aim of this paper is to adopt the same idea for the case of  $\Sigma$ CSPs and CSOPs. Our interest in GAs is also motivated by the fact that they have proven to be useful in hard optimization problems [2],[11],[12], solving multiprocessor scheduling problems[9],[13], etc.

This paper aims to present the new approach and it is organised as follows: The next subsection presents the CSOP and  $\Sigma$ CSP Formalisms. The following one presents the Dynamic Distributed Double Guided Genetic Algorithm; the basic concepts, the agent structure and the global dynamic. The fourth section recalls rapidly the RLFAP and the fifth one details both experimental design and results. Finally, concluding remarks and possible extensions to this work are proposed.

## II. CSOP and $\Sigma$ CSP Formalisms

Please check with A *constraint satisfaction problem*, or CSP, is a triple  $(X, D, C)$ ; whose components are defined as follows:

- $X$  is a finite set of variables  $\{x_1, x_2, \dots, x_n\}$ .

- $D$  is a function which maps each variable in  $X$  to its domain of possible values, of any type, and  $D_{xi}$  is used to denote the set of objects mapped from  $xi$  by  $D$ .  $D$  can be considered as  $D = \{D_{x_1}, D_{x_2}, \dots, D_{x_n}\}$ ;
- $C$  is a finite, possibly empty, set of constraints on an arbitrary subset of variables in  $X$ . these constraints are represented in Extension or in Intention.

A CSP, as defined above, is a hard constraint network. For this, many CSPs extensions, such as constraint satisfaction and optimisation problems (CSOP)[11],[12] and additive constraint satisfaction problems ( $\Sigma$ CSP)[10],[12], have been tackled in the literature.

A CSOP, is a quadruple  $(X, D, C, f)$ ; where  $(X,D,C)$  is a CSP and  $f$  an objective function which maps every instantiation to a numerical value. This function is specific to the problem to solve.

$\Sigma$ CSPs can be defined as special CSPs, in which each constraint has a cost. An objective function  $f$  maps every instantiation to a numerical value. The latter represents the costs sum of violated constraints.

### III. the Dynamic Distributed Double Guided Genetic Algorithm

#### A. Basic Principles

Our 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 provide good results [7]. certain number of methods have been settled in order to favor the building of ecological niches [6] in GAs.

So, the idea here is to partition the initial population into sub-populations and to assign each one of them to an agent called Species agent. A given sub-population consists of chromosomes having their fitness values in the same range. This range, said FVR (for fitness values range), is the specificity of the Species agent  $Species_{FVR}$ . Species agents are in interaction, in order to reach an optimal solution for the problem. For this reason, each Species agent performs its own GA. The latter is guided by both template[11] concept and min-conflict heuristic[16]. An intermediary agent is necessary between the society of Species 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 Species agents.

#### B. Min-Conflict-Heuristic and the Template Concept

Each chromosome is attached to a *template* [11] that is made up of weights referred to as  $template_{i,j}$  or  $\delta_{i,j}$ . Each one of them corresponds to  $gene_{i,j}$  where  $i$  refers to the

chromosome and  $j$  to the position. This fitness template is a map that defines undesirable genes.

When dealing with  $\Sigma$ CSPs,  $\delta_{i,j}$  represents the sum of costs of violated constraints by  $gene_{i,j}$ .  $\delta_{i,j}$  represents the sum of penalties applied on the gene when dealing with CSOPs. In all the cases, these weights are updated by means of the *penalty* operator (see sub-section 3.7). Templates will be used by GA in replacement.

As we use the min-conflict-heuristic, replacement have to be elitist, i.e a chromosome is replaced by a better chromosome. For this, heavier templates genes have more probability to be replaced.

#### C. CSP and GA Relationship

Relationship between both genetic algorithms and CSP formalisms is outlined as below; each chromosome (respectively gene) is equivalent to a CSP potential solution (respectively variable). Moreover, each allele corresponds to a value.

#### D. The fitness function

Central to the theme in GAs is the fitness function. The latter differs from  $\Sigma$ CSPs to CSOPs.

For a given  $\Sigma$ CSP  $(X,D,C,f)$  the objective function for a potential solution  $ps$  can be defined as;

$$f(ps) = \sum W_{C_i} \quad (1)$$

where  $C_i$  is a violated constraint and  $W_{C_i}$  its cost. Let us remember here that Genetic algorithms, always, maximize their chromosomes fitness. For this and by reference to [11], we can define the fitness function as below:

$$g(ps) = Sum_{tc} - f(ps) \quad (2)$$

Where  $Sum_{tc}$  represents the sum of costs of all the problem constraints.

In the case of CSOPs, we define an augmented fitness function (AFF)  $g$  which will be used by the optimization process.

$$g(ps) = f(ps) + \lambda * \sum_i (PC_i * I_i(ps)) \quad (3)$$

The Augmented Fitness function  $g$  can be defined as the function that is the sum of the objective function on a chromosome and the penalties of features that exist in it.

Where  $ps$  is a potential solution,  $\lambda$  is a parameter to the algorithm called Regularization parameter. It is a parameter that determines the proportion of contribution that penalties have in the fitness function value.

$I_i(ps)$  is an indicator for the solution. It is equal to 1 if  $ps$  satisfies all the constraints and it is equal to 0 otherwise.

#### E. 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.

### 1) Species Agent

A Specie agent has got as acquaintances the other Specie agents and the Interface agent. Its static knowledge consists of the  $\Sigma$ CSP or the CSOP data (i.e. the variables, their domains of values and the constraints), the specificity (i.e. the fitness function range) 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) Interface Agent

An Interface agent has as acquaintances all the Specie agents. Its static knowledge consists of the  $\Sigma$ CSP or the CSOP data. Its dynamic knowledge includes the best chromosome (i.e. the chromosome having the best fitness function value).

### F. Global Dynamic

The Interface agent randomly generates the initial population and then partitions it into sub-populations accordingly to their specificities i.e. the fitness value range FVR. After that, the former creates Species agents to which it assigns the corresponding sub-populations. Then the Interface agent asks these Species to perform their optimization processes. So, before starting its own optimization process, i.e. its own behaviour, each Species agent,  $\text{Species}_{\text{FVR}}$ , initializes all templates and penalties counters corresponding to its chromosomes. 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 algorithms, returns a sub-population “pop” that has been submitted to the crossing-over and mutating steps only once, i.e. corresponding to one generation. For each chromosome of pop,  $\text{Specie}_{\text{FVR}}$  computes their fitness function values (FV). Consequently, two cases may occur. The first one corresponds to a chromosome having an FV in the same range as its parents. In this case, the chromosome replaces one of the latter randomly chosen. In the second case, this value (FV) is not in the same range (FVR), i.e. the specificity of the corresponding  $\text{Species}_{\text{FVR}}$ . Then, the chromosome is sent to another  $\text{Species}_{\text{FV}}$  if such agent already exists, otherwise it is sent to the Interface agent. The latter creates a new agent having FV as specificity and transmits the quoted chromosome to it. Whenever a new Species agent is created, the Interface agent informs all the other agents about this creation and then asks the new Species to perform its optimization process. Note that message processing is given a priority. So, whenever an agent receives a message, it stops its behaviour, saves the context, updates its local knowledge, and restores the context before resuming its behaviour.

Here we describe the syntax used in the Figures:

- `sendMsg (sender, receiver, 'message')`: ‘message’ is sent by “sender” to “receiver”.
- `getMsg (mailBox)`: retrieves the first message in mailBox.

```

Messages-processing
1. m ← getMsg (mailBox)
2. Case (m) in
3. optimization-process (sub-population):
   apply-behaviour (sub-population)
4. take-into-account (chromosome):
   population-pool ← population-pool ∪ {chromosome}
5. inform-new-agent (SpecieFV):
   list-acquaintances ← list-acquaintances ∪ {SpecieFV}
6. stop-process: stop-behaviour

```

**Figure 1.** Message processing relative to  $\text{Specie}_{\text{FVR}}$

The behaviour of an agent describes its genetic sub-process. Each Species agent will apply its behaviour on its sub-population until the stopping criterion (a number of generation in our case) is reached.

```

Apply-behaviour (initial-population)
1. init-local-knowledge
2. for k := 1 to number-of-generations do
3.   template-updating (initial-population)
4.   pop ← genetic-process (initial-population)
5.   best-FV ← 0
6.   for each chromosomej in pop do
7.     FVj ← compute-fitness-value (chromosome)
8.     if best-FV ≤ FVj
9.       then best-FV ← FVj
10.      clear (LO-chromosomes-list)
11.      LO-chromosomes-list ← LO-chromosomes-
list ∪ {chromosomej}
12.      if (FVj ∈ rangei)
13.        then replace-by (chromosomej)
14.        else if exist-agent (SpeciesFV)
15.          then sendMsg (Speciesi, SpeciesFV, 'take-
into-account (chromosomej)')
16.          else sendMsg (Speciesi, Interface, 'create-
agent (chromosomej)')
17.    end For
18.   if best-FV = last-FV
19.     then stat-counter ← stat-counter + 1
20.     else last-FV ← best-FV
21.   if stat-counter = LODi
22.     then last-stat-FV ← last-FV
23.     penalize(LO-chromosomes-list)
24.   sendMsg (Speciesi, Interface, 'result (one-chromosome,
specificity)')

```

**Figure 2.** Behaviour relative to  $\text{Species}_{\text{FVR}}$

### G. Guided Cross-over and Guided Mutation

Out of each pair of chromosomes, the cross-over operator produces a new child as described in Figures 3 and 4. The child inherits the best genes, i.e. the “lighter” ones, from its parents. The probability, for a parent chromosome  $i$  ( $i=i_1$  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 best FV, are more likely than the other to be passed to the child.

For each one of its chromosomes selected according to the mutation probability  $P_{mut}$ ,  $Species_{FVR}$  uses the min-conflict-heuristic first to determine the gene (variable) involved in the worst FV, secondly to select from this gene domain the value giving the lightest template value and finally to instantiate this gene with this value (see figure 5).

If all the  $Species$  agents did not meet any better chromosome at the end of their behaviour or they attain the stopping criterion, they successively transmit one of their best chromosomes, linked to its FV to the Interface agent. The latter determines and displays the best chromosome namely the one which have the best FV.

```

Cross-over ( $chromosome_{i1}, chromosome_{i2}$ )
1. for  $j := 1$  to size ( $chromosome_{i1}$ ) do
2.    $sum \leftarrow template_{i1,j} + template_{i2,j}$ 
3.   if (random-integer  $[0, sum - 1] < template_{i1,j}$ )
4.     then  $gene_{i3,j} \leftarrow gene_{i2,j}$ 
5.   else  $gene_{i3,j} \leftarrow gene_{i1,j}$ 
6. Return  $chromosome_{i3}$ 

```

**Figure 3.** Cross-over operator

```

Crossing (mating-pool)
1. if (mating-pool size < 2)
2. then return mating-pool
3. for each pair in mating-pool do
4.   if ( $random [0,1] < P_{cross}$ )
5.   then offspring  $\leftarrow$  cross-over (first-pair, second-pair)
6.     FV  $\leftarrow$  compute-fitness-value (offspring)
7.   offspring-pool  $\leftarrow$  offspring-pool  $\cup$  {offspring}
8. return offspring-pool

```

**Figure 4.** Crossing process relative to  $Species_{FVR}$

```

Min-conflict-heuristic ( $chromosome_i$ )
1.  $\delta_{i,j} \leftarrow \max (template_i) / \delta_{i,j}$  is associated to  $gene_{i,j}$  which is in turn
   associated to the variable  $v_j$ */
2.  $FV^* \leftarrow 0$ 
3. for each value in domain of  $v_j$  do
4.   FV  $\leftarrow$  compute-fitness-value (value)
5.   if (FV >  $FV^*$ )
6.     then  $FV^* \leftarrow$  FV
7.     value*  $\leftarrow$  value
8.   value ( $gene_{i,j}$ )  $\leftarrow$  value*
9. update ( $template_i$ )
10. return  $FV^*$ 

```

**Figure 5.** Min-conflict-heuristic relative to  $chromosome_i$

#### H. Penalty Operator and Local Optima Detector

To enhance the approach, we add another GA's parameter that we call LOD (for local optima detector). The latter represents the number of generation in which the neighboring does not give improvement, i.e. if the FV of the best chromosome remains unchanged for a specific number of generations; and so we can conclude that the agent optimization sub-process is trapped in a local optimum. In fact if the unchanged FV is lesser than the last stationary FV

then automatically LOD have to be equal to one (see figures 11). Otherwise the LOD will remain unchanged i.e. LOD is a parameter to the whole optimization process and it will be dynamically updated by every agent (see figures 2).

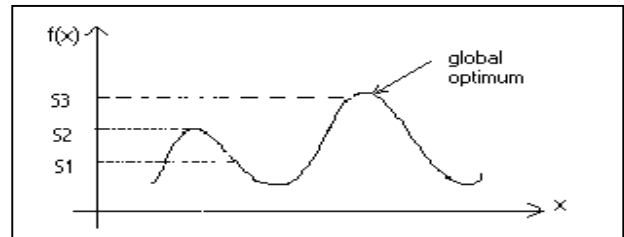
Let us mention that every  $Species_{FVR}$  have to save its best FV for the next generations. This will be very useful not only in the case of stationary fitness values, but also to select the best chromosome in the species. In fact, if the best FV remain unchanged for LOD generation, the process will be considered as trapped in a local optimum. Thus for all the chromosomes having this FV, the related template of all its genes is incremented by one. Penalties will, then be used to update the chromosome templates.

#### I. Mutation and Guidance Probability

The approach, as described until now, can not be considered as a classic GA. In fact, in classic GAs the mutation aims to diversify a considered population and then to avoid the population degeneration [7]. In this approach, mutation operator is used improperly since it is considered as a betterment operator of the considered chromosome. However, if a gene value was inexistent in the initial population, there is no way to obtain it by cross-over process. Thus, it is sometimes necessary to have, a random mutation in order to generate the possibly missing gene values. Our approach is a local search method. The first known improvement mechanism of local search is the diversification of the search process in order to escape from local optima [10]. No doubt, the simplest mechanism to diversify the search is to consider a noise part during the process. Otherwise the search process executes a random movement with probability  $p$  and follows the normal process with a probability  $1-p$  [10].

In figure 7, an example of local optima attraction basin is introduced; in a maximization case,  $S2$ , which is a local maximum, is better than  $S1$ . The passing through  $S1$  from  $S2$ , is considered as a solution destruction but give more chance to the search process to reach  $S3$ , the global optimum.

For all these reasons, the new proposed approach is enhanced by a random providing operator which we call guidance probability  $P_{guid}$ . Thus the approach will possess (in addition to the cross-over and mutation operators, to the generation number and to the initial-population size) a guidance operator.



**Figure 6.** An example of attraction basin of local optima

The mutating sub-process will change; for each selected chromosome following mutation probability  $P_{mut}$ , the mutation will be random with a probability  $1-P_{guid}$  and guided with a probability  $P_{guid}$  (see Figure 7 line 2). So that in the proposed mutating sub-process it's possible to destroy a given solution in order to enhance exploration. This process is illustrated in Figures 7,8 and 9.

```

Mutating (offspring-pool)
1. for each chromosome in offspring-pool do
2.   if (random [0,1]< Pmuti) then if (random [0,1]< Pguid)
3.     then Guided_Mutation (chromosomei)
4.     else Random_Mutation(chromosomei)
5.   FV* ← compute-augmented-fitness-value(chromosomei)
6.   offspring-pool-mutated ← offspring-pool-mutated
   ∪ {chromosomei}
7. return offspring-pool-mutated
    
```

**Figure 7.** Mutating process relative to Species<sub>FVR</sub>

```

Random_Mutation (chromosomei)
1. Choose randomly a genei,j
2. Choose randomly a value vi in domain of genei,j
3. value(genei,j) ← vi
4. Return chromosomei
    
```

**Figure 8.** Random Mutation relative to chromosome<sub>i</sub>

```

Guided_Mutation (chromosomei)
1. min-conflict-heuristic (chromosomei)
2. Return chromosomei
    
```

**Figure 9.** Guided Mutation relative to chromosome<sub>i</sub>

#### J. Dynamic Approach

The main interest of the second improvement is based on the NEO-DARWINISM theory Charles Darwin in 1859 and on the laws of nature « *The Preservation of favoured races in the struggle for life* » [14]. This phenomenon can be described, in nature, by an animal society in which the strongest members are luckier to be multiplied (so their crossing-over probability is high). The power of these elements allows them not to be infected by illnesses (mutation is then at a lower rate). On the contrary case the weakest limbs of these animals are frequently ill or unable to combat illnesses (mutation is frequent), usually this kind of animals can't attract females (reproduction is limited). In fact to cross-over a strong species and to give more mutation possibility for a weak species can be very worthy.

So, from now on  $P_{cross}$  and  $P_{mut}$  will be function of fitness and of a newer operator called  $\epsilon$ . This operator is a weight having values from 0 to 1.

In the optimization process, described in Figures 10 and 11, each species agent proceeds with its own genetic process. Indeed before starting the optimization process agents have to count their parameters  $P_{cross}$  and  $P_{mut}$  on the basis of their fitness values. For a given Species agent three

cases are possible as described by the new genetic process detailed in figures 10 and 11.

```

Genetic process
1. mating-pool ← matching (population-pool)
2. (Pcrossi, Pmuti, LODi) ← count_operator (Pcross, Pmut, LOD)
3. template-updating (mating-pool)
4. offspring-pool-crossed ← crossing (mating-pool)
5. offspring-pool-mutated ← mutating (offspring-pool-crossed)
6. return offspring-pool-mutated
    
```

**Figure 10.** The Genetic process

```

Count_operator (Pcross, Pmut, LOD)
1. if FV < (max-attained-FV / 2) then
2.   Pcrossi ← Pcross /  $\epsilon$ 
3.   Pmuti ← Pmut *  $\epsilon$ 
4.   LODi ← LOD /  $\epsilon$ 
5. if FV > (max-attained-FV / 2) then
6.   Pcrossi ← Pcross *  $\epsilon$ 
7.   Pmuti ← Pmut /  $\epsilon$ 
8.   LODi ← LOD *  $\epsilon$ 
9. if FV = (max-attained-FV / 2) then
10.  Pcrossi ← Pcross
11.  Pmuti ← Pmut
12.  LODi ← LOD
13. if FV ≤ last-stat-FV
14.   then LODi ← 1
15. return (Pcrossi, Pmuti, LODi)
    
```

**Figure 11.** The operator count process

## IV. The Radio Link Frequency Assignment Problems (RLFAP)

The French "Centre d'Électronique de l'Armement" (CELAR) has made available, in the framework of the European project EUCLID CALMA (Combinatorial Algorithms for Military Applications) set of Radio Link Frequency Assignment benchmark problems (RLFAP) (RLFAP is available in Centre d'Electronique et de l'Armement (France), via <http://www.inra.fr/bia/T/schiex/Doc/CELAR.shtml>) build from a real network, with simplified data. These benchmarks had been previously designed by the CELAR to assess several different Constraint Programming languages.

These benchmarks are extremely valuable as benchmarks for the CSP community and more largely for constraint programming:

- The constraints are all binary (involving no more than two variables), non linear and the variables have finite domains.
- These are real-world size problems, the larger instances having round one thousand variables and more than five thousand constraints. All these instances have been built from a unique real instance with 916 links and 5744 constraints in 11 connected components [16],[17].

The Radio Link frequency Assignment Problem consists in assigning frequencies to a set of radio links defined between pairs of sites in order to avoid interferences. Each

radio link is represented by a variable whose domain is the set of all frequencies that are available for this link. The essential constraints involve two variables  $F_1$  and  $F_2$ :

$$|F_1 - F_2| > k_{12}$$

The two variables represent two radio links which are "close" one to the other. The constant  $k_{12}$  depends on the position of the two links and also on the physical environment. It is obtained using a mathematical model of electromagnetic waves propagation which is still very "rough". Therefore, the constants  $k_{12}$  are not necessarily correct (it is possible that the minimum difference in frequency between  $F_1$  and  $F_2$  that does not yield interferences is actually different from  $k_{12}$ ). In practice,  $k_{12}$  is often overestimated in order to effectively guarantee the absence of interference. For each pair of sites, two frequencies must be assigned: one for the communications from  $A$  to  $B$ , the other one for the communications from  $B$  to  $A$ . In the case of the CELAR instances, a technological constraint appears: the distance in frequency between the  $A \rightarrow B$  link and the  $B \rightarrow A$  link must be exactly equal to 238. In all CELAR instances, these pairs of links are represented as pairs of variables numbered  $2k$  and  $2k+1$ . The possibility of expressing constraints such as  $|F_1 - F_2| > k_{12}$  suffices to express the graph coloring problem and it is therefore clear that the RLFAP is NP-hard [16],[17].

In order to facilitate the later addition of new radio links, one tries to build solutions that leave room for these new links. The pure satisfaction problem is therefore not crucial in itself (even if...). Two essential criteria are usually considered for feasible instances:

1) *Minimization of the maximum frequency used*: the frequencies above the last frequency used can be tried for new radio links.

2) *Minimization of the number of frequencies used*: the different frequencies unused can be tried for new radio links. Here, it is likely that the various frequencies available will be more distant one from the other and therefore it is more likely that a new radio link can be inserted without any modification of the previous setup. In practice, this criterion seems therefore more interesting than the previous one.

The first criterion, a Min-Max type criterion, is usually less expensive to optimize from an algorithmic view point. A third type of criteria is used in practice when the problem is unfeasible (inconsistent): Minimization of a weighted sum of the violated constraints. As we said before, in constraints such as  $|F_1 - F_2| > k_{12}$ , the constant  $k_{12}$  is usually overestimated and it is possible that such a constraint be violated without any real interference occurring in practice. The minimization of a weighted sum of the constraints violated corresponds to a minimization of costs induced to check the quality of the communication (and possibly a modification of the position of the antennas). The criteria used on unfeasible instances being different from the criteria used on feasible instances, this gives back some interest to the pure satisfaction problem which, in practice, will allow selecting the optimization criteria to use.

When new links are added to an existing communication network, some variables are already assigned and it is either

impossible to modify the value of these variables (hard constraints) or, again, one must minimize a weighted sum of the variables being modified. The size of the modification is not important. What is important is that the variable value has changed which means that somebody must apply this change. One can imagine that the cost associated with a variable is related to the cost of having the change done. Let us mention here that we will take the same formulation as that presented by [16]. Note that, we will formalize the non solvable instances (instances 6,7,8,9 and 10) as  $\Sigma$ CSPs. The solvable ones (instances 1,2,3,4,5 and 11) are easily formalized as CSOPs.

Inspired by the algorithm of Bellicha [21], we propose an algorithm of generation of random  $\Sigma$ CSPs. Let us note that all the constraints are binary and that their relations are expressed in extension.

1. For each couple of variables (I, J) such as  $1 < i < j < n$  do
2.   choose randomly a number  $x$  between 0 and 1
3.   if  $x \leq p$
4.   then the constraint  $R_{ij}$  is initially-created empty
5.   For each couple of values  $((i,a), (j,b)) \in D_i \times D_j$  do
6.     choose randomly a number between 0 and 1
7.     if  $y \leq q$
8.     then the couple  $((I, a), (j,b))$  is added to the constraint  $R_{ij}$
9. for each constraint  $R_{ij}$  do
- 10 randomly-assign a cost
11. To return the CSP obtained

**Figure 12.** The Random  $\Sigma$ CSPs Generator Algorithm

This generator requires four parameters which are as follows:

- the number of variables  $N$ ,
- the number of values per Domain  $D$ ,
- the probability  $p$  of existence of a constraint between two variables: connectivity,
- the probability  $q$  that there is not a tuple within a constraint: tightness.

One speaks about problems slightly constrained or not very dense for the small values of  $p$ , and hard problems for the great values of  $q$ . In addition, the algorithm which we propose takes into account the cost of the constraints. Indeed it assigns randomly a cost for each constraint of the problem (algorithm 11-1 line 10). This makes that the result is a  $\Sigma$ CSP.

## V. Experimentation

### A. RLFAP experimentations

#### 1) Experimental Design

The goal of our experimentation is to compare a distributed implementation with a centralized one of guided genetic algorithm. The first implementation is referred to as Dynamic Distributed Double Guided Genetic Algorithm ( $D^3G^2A$ ) whereas the second one as Guided Genetic

Algorithm (GGA)[16]. The implementation has been done with ACTALK [15], a concurrent object language implemented above the Object Oriented language SMALLTALK-80. This choice of Actalk is justified by its convivial aspect, its reflexive character, the possibility of carrying out functional programming as well as object oriented programming , and the simulation of parallelism.

On the basis of [18], in our experiments we carry out 30 times the algorithms and we take the average without considering outliers. Concerning the GA parameters, all the experimentations employ a number of generations (NG) equal to 10, a size of initial population equal to 1000, a cross-over probability equal to 0,5, a mutation probability equal to 0,2, an LOD equal to 3,  $\lambda$  equal to 10 and  $\epsilon$  is equal to 0.6. Not that these values has been proved to be the best in [5],[6],[16].

The performance is assessed by the two following measures:

- Run time: the CPU time requested for solving a problem instance,
- Fitness function value: the solution given by the algorithm.

The first one shows the complexity whereas the second recalls the quality. In order to have a quick and clear comparison of the relative performance of the two approaches.

2) Experimental results

In order to have a quick and clear comparison of the relative performance of the two approaches, we compute ratios of GGA and D<sup>3</sup>G<sup>2</sup>A performance using the Run time and the fitness value as follows:

- CPU time Ratio=GGA CPU time / D<sup>3</sup>G<sup>2</sup>A CPU time.
- Fitness ratio= FV of D<sup>3</sup>G<sup>2</sup>A / FV of GGA

Thus, GGA performance is the numerator when measuring the CPU time ratios, and the denominator when measuring fitness value ratio. Then, any number greater than 1 indicates superior performance by D<sup>3</sup>G<sup>2</sup>A.

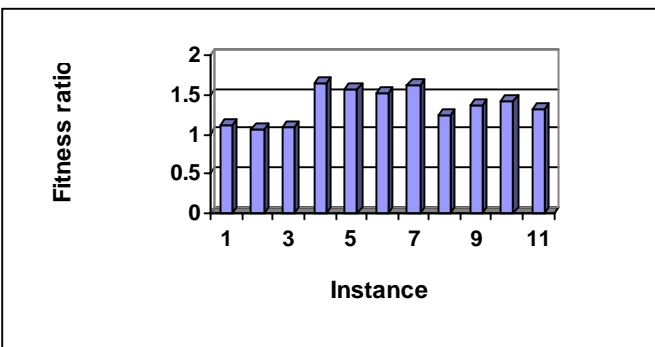


Figure 13. The fitness ratios

From the solution quality point of view shown in figure 13, the D<sup>3</sup>G<sup>2</sup>A always finds better solution than GGA or same one. This ratio is more significant for instances 6 and 7 (see peaks in figure 13). The fitness ratios average is

1.36022182. This is the result of the diversification and the intensification used in our approach.

Form the CPU time point of view (figure 14), D<sup>3</sup>G<sup>2</sup>A requires less time for all the instances of the problem. It requires until 5.122 less time for the same example. In average the CPU time ratio is equal to 3.13165.

We have come to these results thanks to agent interaction reducing GA temporal complexity. In fact, the communication between agents helps them to in the solution investigation. The CPU time is in the other hand reduced thanks to the new proposed genetic process. In the latter, both diversification (by random mutations and by LOD) and guidance are used. The first one helps the optimization process to escape from local optima. The second one, intensifies the search helping it to attain, rapidly, better fitness function values.

Note that compared to benchmarks of the CALMA algorithms, GGA has been shown to be the better in terms of compromise between the solution quality and the temporal complexity[16]. This was the mainn reanon that let us compare our approach to the GGA.

Let us mention that our approach was able to solve all the soluble and unsoluble instances (some of the CALMA project Algorithms was unable to solve them [16]).

It is tempting to note that the experimentations did not show a clear dependency between, the number of generation of the genetic process on the one hand, and on the other hand the evolution of both CPU-time and fitness ratios [6].

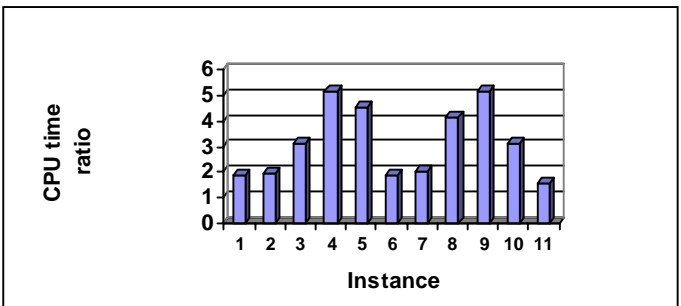


Figure 14. The CPU time ratios

Attention is next focused on the guidance probability study in order to determine its best values. To accomplish this task, we have assembled the CPU-time averages and fitness values averages for different values of  $P_{guid}$ .

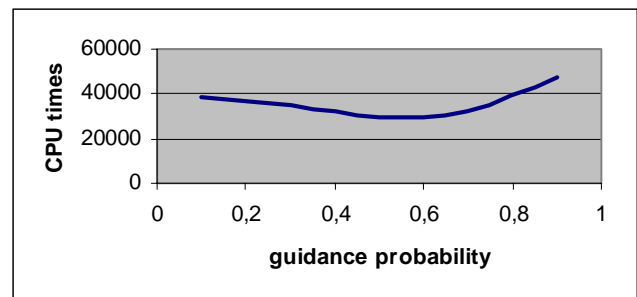
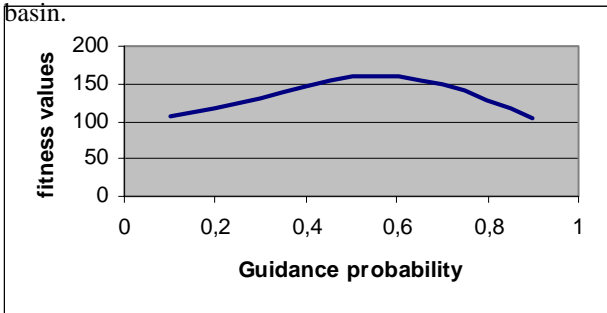


Figure 15. CPU times relative to different values of  $P_{guid}$

Both Figures 15 and 16 show that the best value of  $P_{guid}$  is about 0.5. In fact the best fitness value is reached in this area. In addition to that the least CPU time is also given by this  $P_{guid}$  value. This value can be explained by the fact that not only random mutations are important but also guided mutations. In fact the guided mutating sub-process allows guidance and so helps the algorithm to reach optima, and the random one helps it to escape from local optima attraction



**Figure 16.** Fitness values relative to different values of  $P_{guid}$

## B. Application on Random CSPs

### 1) 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. Therefore, we have 750 examples. It is tempting to note that random costs have been assigned to each constraint. Moreover and considering the random aspect of genetic algorithms, we have performed 10 experimentations per example and taken 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. LOD have an initial value equal to 3, and  $\epsilon$  is equal to 0.6.

Note that by experiments we did show that these values of parameters are those giving the best CPU times and the best fitness values. It's tempting to note that the use of the same  $\epsilon$  has been proven, to be better than the use of different values[1].

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.

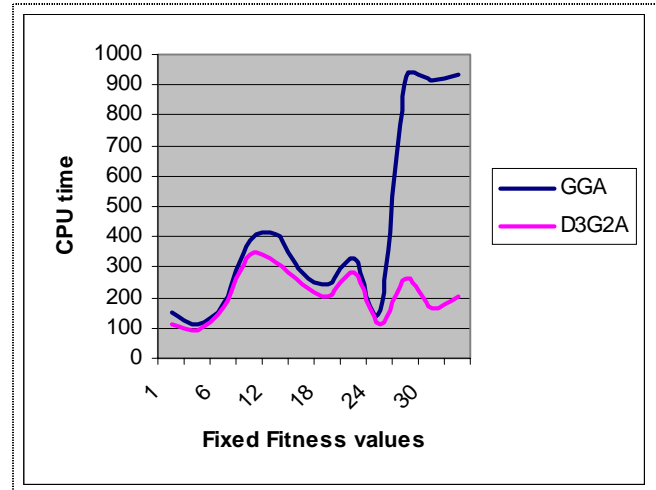
The first one shows the complexity whereas the second recalls the quality. In order to have a quick and clear comparison of the relative performance of the two approaches.

### 2) Experimental results

The performance of the two approaches will be compared as follow:

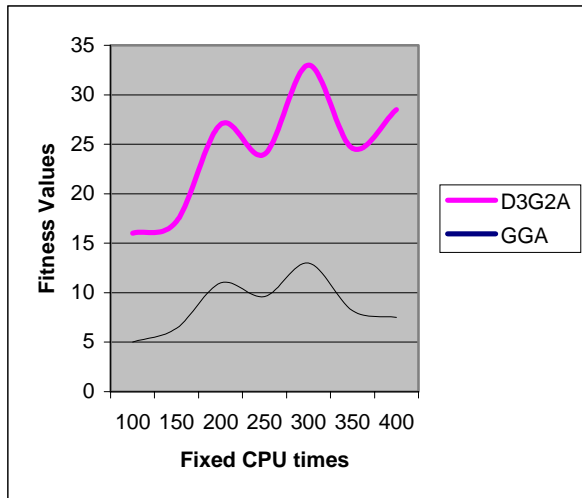
- For the same value of fitness, we compute the time elapsed by each approach to attain it.
- For the same CPU time, we compute the fitness value attained by the two approaches.

Form the CPU time point of view (figure. 17),  $D^3G^2A$  requires less or the same time for the most weakly set of examples i.e. weak FV. Nevertheless, and in some problems especially for the over-constrained and for the most strongly tight set of examples,  $D^3G^2A$  requires less time.



**Figure 17.** CPU times given for fixed value of fitness

From the solution's quality point of view (Figure 17), the  $D^3G^2A$  always finds better solution than GGA. It finds about 3 times more for the most strongly constrained and most tight set of problems. This is the result of the diversification and the intensification used in our approach.



**Figure 17.** fitness values given for fixed CPU times

Note that the approach may scale up well especially in the case of relatively difficult problems. In fact the difficult is the problem the better is the solution given by our approach. This could be explained by the agents' interaction in the used multi-agent system. This interaction will let the optimisation process faster. However a good solution quality is guaranteed.

## VI. Conclusion and Perspectives

within the framework of constraint satisfaction and optimization, we have developed a newer approach called  $D^3G^2A$ . This approach is a dynamic distributed double guided genetic algorithm enhanced by three new parameters called guidance probability  $P_{\text{guid}}$ , the local optima detector LOD and the weight  $\epsilon$ . The latter is a weight used by Species agents to determine their own genetic process parameters on the basis of their chromosomes Fitness values. Compared to the centralized guided genetic algorithm and applied for a set of literature known problems, our new approaches have been experimentally shown to be always better in terms of fitness value and CPU time. The guidance operator has been tested, too. Its best given value has been about 0.5.

The improvement is due to both diversification and guidance. The first increases the algorithm convergence by escaping from local optima attraction basin. The latter helps the algorithm to attain optima. Consequently  $D^3G^2A$  gives more chance to the optimization process to visit all the search space. We have come to this conclusion thanks to the proposed mutation sub-process. The latter is sometimes random, aiming to diversify the search process, and sometimes guided in order to increase the best of the fitness function value. The genetic sub-process of  $D^3G^2A$  Species agents will no longer be the same depending on their fitness values. This operation is based on the species typology. The sub-population of a species agent can be considered as

strong or weak with reference to its fitness value. For a strong species, it's better to increase cross-over probability and to decrease mutation probability. However, when dealing with a weak species, cross-over probability is decreased and mutation probability is increased. The occurrence of these measures not only diversifies the search but also explore wholly its space.

No doubt further refinement of this approach would allow its performance to be improved. Further works could be focused on applying these approaches to solve real hard life problems and distributed CSPs.

## References

- [1] Bouamama S, Ghédira K.: "ED<sup>3</sup>G<sup>2</sup>A: an enhanced Version of the dynamic Distributed double Guided Genetic Algorithms for Max\_CSPs", In *Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI'04)*, 2004, Orlando, Florida, USA.
- [2] DeJong K. A., Spears W. M.: "Using Genetic Algorithms to solve NP-Complete problems" George Mason University, Fairfax, VA, 1989.
- [3] Freuder E.C, Wallace R.J., "Partial Constraint Satisfaction", *Artificial Intelligence*, vol. 58, p 21-70, 1992.
- [4] Bouamama S, Ghédira K.: "D<sup>2</sup>G<sup>2</sup>A and D<sup>3</sup>G<sup>2</sup>A: a new generation of Distributed Guided Genetic Algorithms for Max\_CSPs", In *proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI'03)*, 2003, Orlando, Florida, USA.
- [5] Bouamama S, Ghédira K: "D<sup>2</sup>G<sup>2</sup>A: a Distributed double Guided Genetic Algorithm for Max\_CSPs", In *Proceedings of the Seventh International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'03)*, 2003, Oxford, UK.
- [6] Ghédira K & Jlifi B. : "A Distributed Guided Genetic Algorithm for Max\_CSPs", *Journal of sciences and technologies of information (RSTI)*, journal of artificial intelligence series (RIA), volume 16 N°3/2002
- [7] Goldberg D.E.: "Genetic algorithms in search, Optimization, and Machine Learning", Reading, Mass, Addison-Wesley, 1989.
- [8] Holland J.: "Adaptation in natural and artificial systems", Ann Arbor: The University of Michigan Press, 1975.
- [9] Michael B., Frank M., Yi P.: "Improved Multiprocessor Task scheduling Using Genetic Algorithms", In *Proceedings of the twelfth International Florida AI Research Society Conference FLAIRS'99*, 3-5 May 1999, AAAI press, p. 140-146.
- [10] Schiex T., Fargier H. & Verfaillie G.: "Valued constrained satisfaction problems: hard and easy problems", In *proceeding of the 14<sup>th</sup> IJCAI*, Montreal, Canada august 1995.

- [11] Tsang E.P.K, Wang C.J., Davenport A., Voudouris C., Lau T.L.: "A family of stochastic methods for Constraint Satisfaction and Optimization", *technical report* University of Essex, Colchester, UK, November 1999.
- [12] Tsang EPK, *Foundations of Constraints Satisfaction* Academic Press Limited, 1993
- [13] Tsujimura Y., Gen M.: "Genetic algorithms for solving Multiprocessor Scheduling Problems", In *Proceedings of the 1st Asia-Pacific Conference on Simulated Evolution and Learning*, LNAI, November 9-12 1997, Springer, vol. 1285, p. 106-115.
- [14] Darwin C.: "*The Origin of Species*", 1859, Sixth London Editions, 1999.
- [15] Minton S. "Minimizing conflicts a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence*, volume 58 pages 161-205,1992.
- [16] Lau T.L and Tsang E.P.K. Solving The Radio Link Frequency Assignment Problem With The Guided Genetic Algorithm, In *Proceedings of NATO symposium on radio length frequency assignment starting and conservation systems*, UK. Albrg, Denmark, October 1998.
- [17] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, J. P. Warners. Radio Link Frequency Assignment, *Constraints* 4(1): 79-89 (1999).
- [18] Craenen B., Eiben A.E., and J.I. van Hemert, 2003. Comparing Evolutionary Algorithms on Binary Constraint Satisfaction Problems, *IEEE Transactions on Evolutionary Computation*, 7(5):424-444.
- [19] Bouamama S, Ghédira K.: "D<sup>3</sup>G<sup>2</sup>A: a new Dynamic Distributed Double Guided Genetic Algorithms for  $\Sigma$ CSPs", In *proceeding of the 2005 IEEE congress on evolutionary computation CEC2005*, 2-5 September 2005 Edinburgh, Scotland , p795.
- [20] Bouamama S, Ghédira K.: "D<sup>3</sup>G<sup>2</sup>A the Dynamic Distributed Double Guided Genetic Algorithms and its application for the RLFAP", In *proceedings of the 2005 IEEE congress on evolutionary computation CEC2005*, 2-5 September 2005 Edinburgh, Scotland, p1298.
- [21] Bellicha A., Capelle C., Habib M., Kökény T. and Vilarem M.: "Exploitation de la relation de substitutalité pour la résolution des CSPs". *Revue d'intelligence artificielle RIA* n°3, 1997.