Improving genetic algorithm using arc consistency technic

Meriem Zouita, Sadok Bouamama, Kamel Barkaoui

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 1387–1396

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Improving genetic algorithm using arc consistency technic

Meriem Zouita [a*], Sadok Bouamama [a], Kamel Barkaoui [b]

[a] *ENSI,University of Manouba, Manouba, Tunisia*
[b] *CNAM,Paris, France*

Abstract

We studied in this article a topic that focused on two areas of research: Constraint Satisfaction Problems (CSP) and genetic algorithms. The problem is that this type of algorithm is recognized to be greedy in terms of CPU time. To solve this problem, we tried to integrate the arc consistency (AC) at the initial population in a way that it would be the result of this filtering. First, we generated the genetic algorithm without integrating the arc consistency. Then, we considered that each chromosome is a CSP, each gene is a variable of the problem and each allele represents the taken value. We randomly generated the CSP to obtain the inconsistent values of each pair of variables. To remove these values, we used the technique of arc consistency as a technique for solving this type of problem, that means we have worked to eliminate from each variables domain the values which violate the constraint specific and make the CSP inconsistent. The aim of this work is to reduce performance in terms of execution time of the genetic algorithm.

Keywords: Genetic algorithm, CSP, arc consistency (AC), time CPU.

1. Introduction

In everyday life, we all face more or less complex optimization problems. That can start, for example, when we try to put our offices together. Taking as an example the domain of the industry in which the business environment is mainly characterized by competition from markets where the

* Corresponding author. *E-mail address:* zouitazouita90.m@gmail.com

10.1016/j.procs.2019.09.309

demands and expectations of customers are becoming stronger and more important in terms of quality, cost and time of provision. This is why we can say that the success of a project or the excellent development of a company are mainly linked to the competence and ability of engineers to solve this type of problem (optimization problem) while minimizing Costs and maximizing production. In order to solve such a problem, we must give good values to the various parameters in order to reach an optimal solution. These values must respect the constraints associated with the problem. Generally, there are two distinct approaches to solving an optimization problem: There are the exact methods that allow certain problems to find the best solution or optimal solution. But most of the problems studied in optimization belong to the class of NP- Complex problems. This class brings together problems for which there is no algorithm that provides the optimal solution. There is also the stochastic methods which content themselves with looking for a "good quality" solution. To this end, different approaches have been developed. Among them, we can cite simulated annealing, taboo research and genetic algorithms. The principle of the latter is inspired by the laws of natural selection, described by Darwin. The evolution of a species is simply a successive sequence of improvement so that it is best adapted to the environment in which it evolves. This adaptation is achieved through natural selection and the mechanisms of reproduction: crosses, mutations and selections. This algorithm is now very successful. We use it in complex problem solving, requiring high CPU (Central Processing Unit) processing times. In order to minimize this execution time, we had the idea to put the problem in the form of a constraint satisfaction problem (CSP) and to enrich it by using a technique of filtering "the consistency of arc" . Constraint satisfaction is a powerful tool for modeling and solving combinatorial problems associated with a set of powerful resolution techniques.

This modeling approach has proved successful in the areas of planning, configuration, resource allocation and others.

A constraint network consists of a set of variables, their domains of values and a set of constraints between these variables. This formalism is very simple it allows to model easily many problems. One of the fundamental tasks of constraint networks is to find a solution to the modeled problem, that is to say an assignment of values to the variables of the network which satisfies the set of constraints. The work supported in this document is that the integration of a technique of solving a problem of constraint satisfaction in a genetic algorithm can eventually bring a satisfactory solution to a problem solved with genetic algorithm also to reduce the time d CPU execution. This article consists of four parts: The first speaks about the CSPs. The second speaks of the arc consistency The third presents the genetic algorithms. The last part discusses the obtained results.

2. The CSP (Constraint Satisfaction Problems) formalism:

2.1. Definition

Formally, a CSP [2] is a modelized problem in the form of a set of constraints placed on variables, each taking its values in a domain. We define a CSP by a triple (X, D, C) such that:

- X = (x1, x2, ..., xn) the set of n variables of the problem.
- D = (d1, d2, ..., dn) the set of n discrete and finite domains of the problem: each variable $x_i \in X$ is associated with a domain of values $d_j \in D$.
- C = (c1, c2, ..., cm) the finite set of m constraints related to the problem. A CSP can be of different types:

Unary: the variable bears only on itself

Binary: the variables are in pairs

Ternary: the variables are in three to three

N-aire: relation between all variables of the problem

2.2. Binary CSP

A binary CSP can be represented by a Constraint Network G = (X, Y) where each node of this graph represents a single variable and each arc connecting two nodes represents a constraint.

2.3. Notion of constraint

A constraint is a relation or property that must be checked between the (unknown) variables. It restricts all values of domains that can take variables simultaneously. We denote by var (c) the set of variables involved in this constraint c. The arity of a constraint c∈C represents the number of variables on which it bears. It is said that the constraint is:

Unary if its arity is equal to 1.

Binary if its arity is equal to 2.

N-aire if its arity is equal to n. The objective of a CSP is to find a set of values to assign to variables in a way that all constraints are satisfied.

2.4. Instantiation / assignment

The assignment is a set of pairs (variables- values), in effect it assigns to each variable a value of its domain: $A = \{(xi \leftarrow vi)\}$ with $xi \in X$ and $vi \in D(xi)$.

2.5. Solution of a CSP

A solution of a CSP is a total and consistent assignment.

3. Concept of consistency

Consistency is a property related to the compatibility between domain values and constraints. It must be ensured at every stage of the search for the solution. Indeed, it removes domains from variables, values that do not participate in any solution.

3.1. *Arc consistency(AC)*

The arc consistency **[1]** remains one of the fundamental properties of constraint satisfaction problems. It is the oldest and most used of the local consistencies. It is concerned with binary constraints and guarantees that any value of the domain of a variable has a support in any constraint. There is, however, a simple way to make a coherent arc constraint: removing domains from variables that do not belong to any correct instantiation of the constraint. Let a CSP (X, D, C) be a binary constraint $c \in C$ on the variables x and y with their respective domains Dx and Dy. We will say that this constraint is consistent if:

- $\forall a \in Dx, \exists b \in Dy, (a, b) \in c,$
- $\forall b \in Dy, \exists a \in Dx, (a, b) \in c.$

3.1.1. The arc consistency algorithms

There are different versions of these algorithms: AC or called REVISE, AC1, AC3, ..., each version being more efficient than the previous one.

3.1.1.1REVISE procedure

The Revise **[2]** procedure removes from the domain of the variable i all values without support for the constraint Cij and returns a Boolean indicating whether the domain of xi has been modified.

- Insufficient of REVISE:

To ensure that each arc is consistent, only one application of the Revise procedure is insufficient. Indeed, the deletion of a value of a domain for a given constraint can have repercussions on the other constraints on this domain.

- AC1 algorithm

The algorithm AC-1 **[2]** calls the procedure Revise (x, y) for all pairs of variables (x, y) linked by a binary constraint, and repeats this loop as long as one of the calls returns true. One obvious cause of the inefficiency of AC-1 is that it is enough for a single Revise-arc call to perform a single deletion so that all the arcs are re- examined during the next iteration.

- AC3 algorithm

More efficient than AC-1, Mackworth presents a variant, AC-3 [2], which uses a queue to re-check only those constraints in which one of the domains has been modified by avoiding unnecessary tests. The improvement provided by AC-3 consists in not reapplying Revise-arc to all the arcs, but only to those likely to be affected by the deletion of the value of a domain. The arcs to be processed are put on hold in the Q list, and of course, an array is not reinserted if it already exists. AC3 (y) holds the arc consistency after an assignment to y. Initially, Q = {y}. Then as long as Q is not empty: y = Q.deepile (). For any variable x linked to y, call Revise (x, y). If the call returns true, then add x to Q if it is not already there.

- Other Aci

Since the proposal of AC-3, several algorithms establishing the arc-consistency have been proposed, these algorithms aim to improve the theoretical complexity and the actual calculation time. **AC-4**: An improvement in AC-3 that reduces theoretical complexity by using sophisticated data structures. AC-4 avoids redundant testing related to AC3. AC-4 has two steps: the first is to save information about values and its supporters in other areas in a data structure; The second step is to delete the values that do not have supporters. When a value is deleted, it is added to a queue to propagate the effect of its deletion.

- **AC-6 [3]** which mixes principles from AC-3 and AC-4. The main goal of AC-6 is to reduce the spatial complexity generated by AC-4. AC-6 only backs up a media for each value. The values are assumed to be ordered.

4. Genetic algorithm

4.1. Basic principle of a genetic algorithm

The principle of a GA is simple, it is a question of simulating the evolution of a population of individuals up to a criterion of arrest.

4.2.2 The operators of a GA

4.2.2.1. The coding

The coding of the data is a very important part of genetic algorithms because its quality conditions the success of the algorithm, to this effect we say that the choice of the coding is delicate. The coding makes it possible to present the individual in the form of a chromosome. Binary encodings were widely used at the outset. Afterwards, the real codings are now widely used, in particular in the case where the maximum of a real function is sought.

4.2.2.2 *Random generation of the initial population*

The choice of the initial population of individuals is important because it determines the speed of the algorithm by making the convergence towards the global optimum more or less rapid. This mechanism must be able to produce a heterogeneous population of individuals. After we have to randomly generate individuals by making random draws making sure that the produced individuals respect the constraints. If, on the other hand, a priori information on the problem is available, it seems to generate the individuals in a particular sub- domain in order to accelerate the convergence.

4.2.2 A function to optimize

Since the genetic algorithm is an optimization algorithm then we must define a function that we call the adaptation function or fitness f to optimize which will quantify the adaptability of each individual to the environment that surrounds it. This function of evaluating the individual returns a positive real value.

4.2.2.1 **The reproduction operators**

- **The selection**

The purpose of breeding is to identify and select individuals in the population as a whole to reproduce. This operator does not create new individuals but identifies the best ones on the basis of their adaptation function, the best adapted individuals are selected while the least adapted ones are discarded.

- **Croisment**

The crossing or reproduction operation allows the exchange of information between the chromosomes. This operator aims to enrich the diversity of the population while manipulating the structure of the chromosomes. It combines the genes of the two parent individuals who are already selected to get one or two new children. In order to make a cross over chromosomes consisting of M genes, we randomly draw a position in each of the parents. We then exchange the two terminal subchains of each of the two chromosomes, which produces the descendants. The crossing is based on the idea that two performing parents will produce better children.

- **Mutation**

The role of this operator is to randomly modify, with a certain probability, the value of a component of the individual. The following figure shows an example of a chromosome mutation.

5. Contribution

Proposal of an enriched genetic algorithm with arc consistency (AC):

The basic idea of this work is to integrate the arc consistency at the level of the genetic algorithm whose objective is to reduce the CPU execution time. In these experiments, first we tested the standard genetic algorithm until finding the solution. Indeed, this type of algorithm is recognized to be greedy in terms of CPU time (sometimes they take a lot of time during the execution of a given problem). To solve this problem, we considered that each individual in the initial population is a constraint satisfaction problem that we seek to solve with the arc consistency technique, so the initial population will represent the result. Using this method, we have been able to reduce the execution time while eliminating from each domain the values that violate the constraint specific to their variables and which make the CSP inconsistent.

5.1. Experiments and validation

All experiments are done with the java programming language using the eclipse software. This choice of java is justified by its speed, security and reliability. We used an Intel® Core ™ i3-2330M CPU @ 2.20 GHZ 220 GHZ. The aim of this work is to minimize the CPU execution time of a genetic algorithm without first using the arc consistency and after its enrichment with this filtering technique. For the creation of the genetic algorithm we used the pseudo code below:

Algorithm 1 A Genetic Algorithm Pseudo-Code

```
1: Choose an initial random population of individuals
2: Evaluate the fitness of the individuals
3: repeat
4:     Select the best individuals to be used by the genetic
       operators
5:     Generate new individuals using crossover and mu-
       tation
6:     Evaluate the fitness of the new individuals
7:     Replace the worst individuals of the population by
       the best new individuals
8: until some stop criteria
```

Algorithm 1 : Pseudo code of the genetic algorithm

For the parameters of the genetic algorithm, we used, during all the experiments for the size of the individual (the number of variables), 100 for the population size, a probability of crossing Pcrois of value equal to 0.5 and Probability of mutation Pmut equalto0.15. After generating the algorithm and finding the solution. We have switched to the implementation of a random CSP [4][5]. This type of generator used for constraint satisfaction problems is one of those commonly used in the CSPs community. Generation is guided by classical CSP parameters: the number of variables of the problem (n), the size of their domain (d) (as numerical values, we use n = 20, d = 2) the stress density P (This number can be specified either as an integer or as a fraction between 0 and 1

```
Algorithme Générateur-Aléatoire
1- Début
2- Pour chaque couple de variables (i, j) tel que 1< i <j < n Faire
        a. Tirer aléatoirement un nombre x entre 0 et 1
        b. Si x ≤ p
                alors La contrainte Rij est créée-initialement vide
                Pour chaque couple de valeurs ((i, a), (j,b))∈Di x Dj Faire
3. Tirer aléatoirement un nombre y entre 0 et 1
4. Si y ≤ q
        a. Le couple ((i, a), (j,b)) est ajouté à la contrainte Rij
5- Renvoyer le CSP obtenu
6- Fin
```

and indicating the ratio between the number of effectiveconstraints of the problem and the number of possible constraints, that is, a complete graph Of constraints) and the hardness q (this number can be specified either as an integer or as a fraction between 0 and 1, it refers to the number of pairs of values that are not allowed by the constraint. Are chosen randomly with the generator).

Algorithm 2: Algorithm (6): Algorithm of the random generator of the CSPs

1.1. Experiments

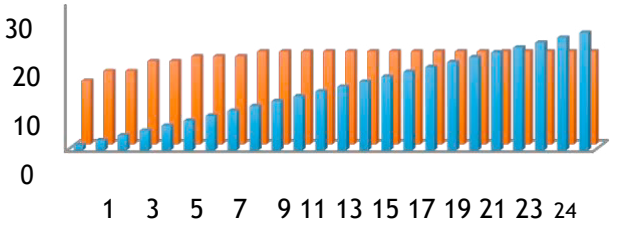
The experiments are carried out in two parts:

First: the implementation of the genetic algorithm before enriching it with the technique of arc consistency.

Second: the implementation of the same algorithm with this filtering technique. In order to have a clear comparison of the execution between step 1 and step 2, the performances are evaluated by the following measure:

- Runtime: TIME-CPU requested toresolve a problem.

The genetic algorithm without AC: At the beginning of our experiments, we generated the genetic algorithm without integrating the arc consistency. This algorithm comprises a population of 100 individuals in which each individual expresses itself in the form of a binary string of length 20. The AG seeks to find the binary string of the solution. After having applied the operators of the algorithm we were able to find the generation and the competence of the desired solution.



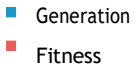


Diagram 1: Genetic algorithm without AC

After finding the solution, we calculated the execution time of the algorithm. The second diagram is interested in showing us the consumed time (CPU) during the execution of the algorithm to find the solution. The latter was found after 15 ms. This represents the time consumed in milliseconds by the genetic algorithm throughout the generation in order to achieve the goal.

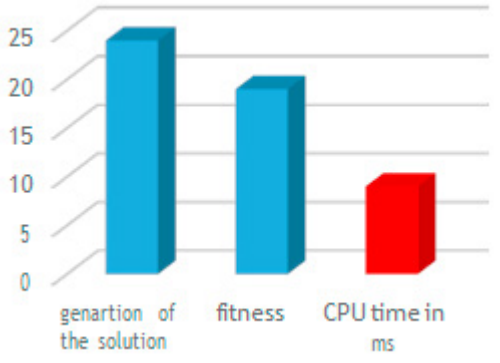
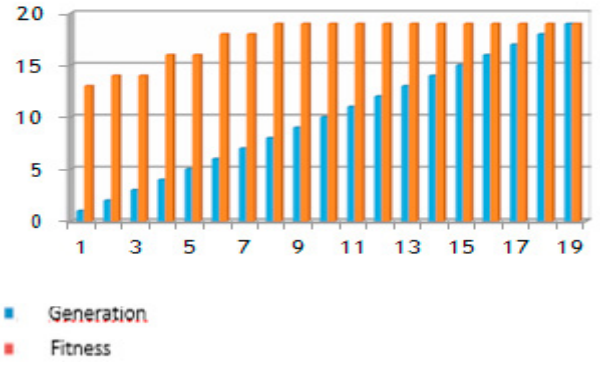


Diagram 2: CPU execution time before AC integration

The genetic algorithm enriched with AC: In order to minimize the execution time found during the first experiment, we considered that each chromosome of the initial population is a problem of constraint satisfaction and that each gene represents a variable. Then, we randomly generated the CSP. The latter, could randomly indicate the pairs of variables involved in the same constraint. It displays in front of each pair the forbidden values. So for the resolution of this CSP, we used the arc consistency to remove the forbidden values from the domains of the variables. We then rectified the values of the variables and placed them back into the initial population to regenerate the genetic algorithm again in order to obtain better results in terms of execution time. After randomly generating the CSP and removing inconsistent values from the variable domains, we were able to find the results indicated by the following diagrams. The third diagram shown below shows that the desired solution is found at the level of the 19th generation with a competence equal to



19.

Diagram 3: Genetic algorithm with AC

The fourth diagram shows us the time consumed during the new execution of the genetic algorithm after using consistent values at the level of the variables. This diagram shows that generation number 19 was found after about 9ms. That is, the solution was found after 9ms.

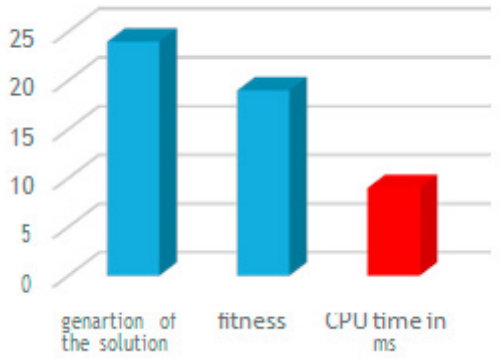


Diagram 4: CPU runtime after AC integration.

Comparison in terms of CPU time between the genetic algorithm without AC and the genetic algorithm enriched with AC:

The diagram above helps us to better visualize the results. Indeed, we can clearly distinguish the difference in terms of the time between the execution of the non- enriched genetic algorithm with the arc consistency and that corresponding to the enrichment of the algorithm with this filtering technique. By removing the prohibited values of the individuals of the initial population, we have been able to decrease the time of execution of the genetic algorithm, we have gained a 6 ms gain from 15ms to 9ms. This gain also explains the reduction in the generation number of the 24 generation to 19 generations, ie the algorithm has made fewer iterations to find the solution.

5.3 Results interpretation

According to the first diagram we have found that the number of generations made by the genetic algorithm to reach the solution before the integration of the consistency arc is equal to 24 with a skill equal to 19. The second diagram showed the CPU execution time that the genetic algorithm consumed to reach the chromosome of the solution before the enrichment with the arc consistency ie at this level we are not yet interested To the inconsistent values, at this stage we had a CPU time = 15 milliseconds.

After enriching the algorithm using the filtering technique (AC) we have found as diagram 3 shows a generation number equal to 19 with a better skill equal to 19 as well. And we were able to decrease the execution time as shown in Diagram 4 and 5 from 15 ms to 9 ms. This gain in terms of time could also give a gain in terms of the number of iterations (generations) performed.

6. Conclusions

Genetic algorithms and the resolution of a problem of constraint satisfaction by the consistency of arc. The aim was to reduce the CPU time by using the arc consistency at the level of constraints that have been violated. Our findings will be divided into two sections. In The first, we set the framework and the problematic of our work. In the second, we summarize our contribution in terms of our subject (a general synthesis of our work) and then we will return to the research perspectives that we have been able to highlight during the work.

Part 1: Framework of work and issue

Our work is positioned within the framework of the integration of one of the filtering methods (the arc consistency) at the level of the initial population of a genetic algorithm. This work was based on a reasoning based on satisfaction of constraints. In the first part of this manuscript, we have given importance to the presentation of the CSP, to explain well the method arc consistency as a filtering method (this method has the advantage of prove to be a good alternative for the Resolution of CSP) so we tried to speak well of GAs. So the framework of our work combines the two research axes: the resolution of a CSP using the arc consistency and the genetic algorithm. In fact, this last one is known to be gourmant in term of CPU time therefore we have interest of diminishing this objective consists the problematic of our work.

♣ Part 2: General Work Synthesis

Using the arc consistency technique as a filtering method we have succeeded in decreasing the time consumed by the genetic algorithm and with this result we have been able to achieve the objective of the problem.

REFERENCES

[1] Christian Bessière. (1991). “Arc-consistency in dynamic constraint satisfaction problems. “ In Proceedings of the ninth National conference on Artificial intelligence - Volume 1 (AAAI'91), Vol. 1. AAAI Press 221-226.

[2] A.K Mackworht, (1977). “Consistency in Networks of Relations”. Artificial Intelligence, Volume 8, Issue 1, February 1977, Pages 99-118

[3] C. Bessière, (1994). “Arc-consistency and arc-consistency again.” Journal Artificial Intelligence, Volume 65 Issue 1, Jan.1994, pages 179-190.

[4] Bouammama.S, Ghedira K, “A new generation of distributed guided genetic algorithms for Max_CSPs resolution.” RSTI - TSI. Volume 27 – n° 1-2/2008, pages 111 à 142.

[5] Dali Narjes, Bouamama Sadok, (2016) “A New Hybrid GPU-PSO Approach for Solving Max-CSPs”: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion Pages 119-120