Optimization by hybridization of a genetic algorithm with constraint satisfaction techniques

Nicolas Barnier, Pascal Brisset

To cite this version:

Nicolas Barnier, Pascal Brisset. Optimization by hybridization of a genetic algorithm with constraint satisfaction techniques. IEEE 1998, World Congress on Computational Intelligence, May 1998, Anchorage, United States. pp 645 - 649, 10.1109/ICEC.1998.700115 . hal-00937716

HAL Id: hal-00937716
https://hal-enac.archives-ouvertes.fr/hal-00937716
Submitted on 17 Apr 2014
Optimization by hybridization of a genetic algorithm with constraint satisfaction techniques

Nicolas Barnier, Pascal Brisset

Abstract—We introduce a new optimization method based on a Genetic Algorithm (GA) mixed with Constraint Satisfaction Problem (CSP) techniques. The approach is designed for combinatorial problems whose search spaces are too large and/or objective functions too complex for usual CSP techniques and whose constraints are too complex for conventional genetic algorithm. The main idea is the handling of sub-domains of the CSP variables by the genetic algorithm. The population of the genetic algorithm is made up of strings of sub-domains whose fitness are computed through the resolution of the corresponding “sub-CSPs” which are somehow much easier than the original problem. We provide basic and dedicated recombination and mutation operators with various degrees of robustness. The first set of experimentalizations addreses a naive formulation of the Vehicle Routing Problem (VRP) and the Radio Link Frequency Assignment Problem (RLFAP). The results are quite encouraging as we outperform CSP techniques and genetic algorithm alone.

Keywords—Optimization, Constraint Satisfaction, Hybridization

I. INTRODUCTION

Solving an optimization problem consists in exploring a search space to maximize a given objective function. The relative structural or size complexities of the search space and the objective function lead to use drastically different strategies. Roughly, we can assume that a deterministic method is suited to a small and/or complex search space whereas a stochastic search strategy (simulated annealing, genetic algorithm...) is fitted to a large one.

In most cases, an optimization problem is naturally divided into two phases: the search of feasible solutions and then the search of the solution with the lowest cost among them. This division is more or less obvious during the search according to the choice of the optimization method.

Genetic algorithms [1] are well suited to the quick and global exploration of a large search space to optimize any objective function (even a “black box” one, i.e. no hypothesis is required on the function) and are able to provide several solutions of “good quality”. In the case the set of the feasible solutions is complex (i.e. finding a feasible solution is difficult), the feasibility can be intrinsic to the chosen representation or integrated within the creation of the chromosomes (initialization, mutation and crossover) or within the objective function (an unfeasible solution will be given a low fitness).

Constraints satisfaction techniques are fitted to highly constrained problems for which the exhaustive exploration of their search spaces are conceivable. Such a method provides naturally feasible solutions. By adding a dynamic constraint on the cost of the currently found solution, the search can provide an optimal solution (cf. the maximize predicate of Constraint Logic Programming systems like CHIP[2]). This method ensures optimality of the solution (possibly with a given percentage).

However, there is no such simple dichotomy among the set of optimization problems: many problems are highly constrained and have large search spaces. These two features exclude the direct and naive use of a genetic algorithm or a CSP technique alone.

We suggest to take advantage of the two approaches by hybridizing them:

• use of constraint satisfaction to compute feasible solutions on a subspace of the search space;
• use of a genetic algorithm to explore the space formed by the set of these subspaces and perform the optimization.

The underlying idea is illustrated in figure 1 (in the particular case of a problem with two variables X1 and X2 constrained in interval domains): the dark areas are individuals of the population of the genetic algorithm which correspond to subspaces of the search space; for each subspace, a solution is computed with the associated “sub-CSP”. An individual does not necessarily correspond to a solution and two different individuals may correspond to the same solution. The ratio of the size of a subspace to the size of the whole search space (called $p$ afterwards) is the essential parameter of the hybridization: one can continuously pass from a pure CSP search ($p = 1$) to a pure stochastic search ($p = 0$, i.e a subspace is reduced to a single value).

We introduce in this article a generic method to implement this hybridization for any CSP on finite domains with the help of Constraint Logic Programming CLP(FD), but its use may be widespread to any problem whose variables belong to $X$, assuming that a CLP(X) framework is provided. First, we recall briefly what CLP(X) framework is, then we describe the components of our hybrid genetic algorithm: initialization of the population, operators (mutation and crossover) and evaluation. We conclude with an encouraging example and compare our method with similar approaches.

II. CONTEXT

We present in this section the two optimization techniques for which we introduce hybridization in the next section.
Thus, the use of a genetic algorithm to solve an optimization problem requires a data encoding to build gene strings (i.e. chromosomes), some mechanism to initialize the population (usually uniformly distributed throughout the search space) and operators allowing to diversify the population (consequently to explore the search space) and to focus on the fittest individuals. All these components are described in the next section.

### III. A mixed approach

We present in this section the components of our hybrid genetic algorithm designed to CSPs on finite domains.

#### A. Chromosome

An individual of the genetic algorithm is formed by a gene strings $G_1G_2...G_n$, i.e. its chromosome. Each gene of the chromosome is associated with a variable of the CSP and an individual represents a sub-problem as well as a solution. The gene $G_i$ corresponding to the variable $X_i$ is a sub-domain (or a subset) of $D(X_i)$ and $|G_i|$ is its cardinality. The essential parameter of the algorithm $\rho$ ($0 \leq \rho \leq 1$) is defined by the ratio of $G_i$ to $D(X_i)$ cardinals:

$$\rho = \frac{|G_i|}{|D(X_i)|}$$

The degree of hybridization can thus “continuously” vary from a pure genetic algorithm, assuming that $\rho$ is chosen small enough for the sub-domains $G_i$ to contain only one value, to a pure CSP solving with $\rho = 1$, i.e. $G_i = D(X_i)$. It may however be useful to provide different $\rho$ ratios for each of the variables if the sizes of the initial domains are very disparate.

The individuals of the first generation are build during the initialization of the genetic algorithm: a sub-domain of the needed size computed by the following formula\(^1\)

$$|G_i| = \max\{1, \text{round}(\rho |D(X_i)|)\}$$

is randomly generated for each gene of each individual.

#### B. Valuation

The fitness of an individual is computed during the solving of the CSP restricted to the corresponding sub-domains: for an individual defined by the chromosome $G_1G_2...G_n$ the constraints $X_i \in G_i$ are added; the sub-CSP is then solved in a standard way by the labeling of the variables. If a solution is found, the fitness is simply computed by applying the objective function to the values of the instantiated variables. Otherwise, i.e. the sub-space $G_1 \times G_2 \times ... \times G_n$ does not contain any solution, the individual can be rejected or given a low fitness (possibly 0). The same kind of penalty is applied when the CSP becomes inconsistent as the $X_i \not\in G_i$ constraints are added.

However, to prove that a sub-space cannot provide any solution might be very time consuming and the maximum

\(^1\)One value is at least provided to each gene with this formula. Empty sub-domains would be of little interest.
time taken by the evaluation of the fitness is a critical data for a genetic algorithm. So a mechanism is provided within our algorithm to stop the labeling of the variables after some given delay.

In the case of a true hybridization ($\rho < 1$), it is not necessary to perform the optimisation during the solving of the sub-CSPs as described in section II-A because the genetic algorithm handles it.

C. Classic operators

Genetic algorithms have traditionally used domain independent representation, namely bit strings, to encode individuals chromosomes. However, for practical reasons of efficiency, many different representations are used which provide much better results, like real strings for instance. But the classic operators designed for bit strings [1] can be used with very little changes to handle various data representations. We describe here the transposed classical operators implemented in our algorithm.

C.1 Crossover

As described in table I, the classic slicing crossover (or n-point slicing crossover) can be directly used with our encoding. Actually this operator is not related to the gene representation, i.e. boolean values are not required.

C.2 Mutation

Classic mutation performs local moves on the individuals, and a similar process can be applied on our sub-domain strings.

Our mutation operator alters a randomly chosen gene by changing some values of its sub-domain or by replacing it by a randomly generated new one. A gene is then able to explore its search space, i.e. the parts of size $|G_i|$ of $D(X_i)$ (assuming that $G_i$ is the mutated gene), and an individual is consequently able to explore all its search space, which is an essential property of mutation.

However, classic mutation may be interpreted in a more semantic way by relating the alteration of a bit to the complementary of a sub-domain in its domain. But such an operator would not keep the size of the sub-domains constant, increasing the difficulty of the analysis of the algorithm, and possibly produce mutants which could be very far from their parents if $\rho$ is not close from $1/2$. Nevertheless, next section describes a “set oriented” operator which can outperform classic ones on some problems.

D. Set oriented operators

Classic operators translated from the bit strings ones are very robust but not always very efficient. It is therefore of interest to design more semantic ones which care about the set-like structure of our genes. Classic set operators like union, intersection and complementation may drastically change the size of the genes and are consequently hardly suitable for our algorithm.

We introduce a new “set oriented” crossover operator (as described in table II):

\begin{itemize}
  \item for each locus $i$, the union of the fathers sub-domains is computed $G_{i,\text{father}}^U = G_{i,\text{father}}^1 \cup G_{i,\text{father}}^2$;
  \item then a subset of the right size is randomly taken from $G_{i,\text{father}}^U$ to build the first child $G_{i,\text{child}}^1$;
  \item the rest of $G_{i,\text{father}}^U$, i.e. $G_{i,\text{father}}^U - G_{i,\text{child}}^1$, makes up the first part of the second child $G_{i,\text{child}}^2$;
  \item if the size of $G_{i,\text{child}}^2$ is too small, a subset of the complementary size is randomly taken from $G_{i,\text{child}}^1$ to fill $G_{i,\text{child}}^2$.
\end{itemize}

This recombination operator “shuffles” the genes of the parents in a way similar to the uniform crossover (n-point crossover with chromosomes of size $n$) and provides children as different as possible, re-using all the parents genes data and keeping constant sub-domains sizes.

E. Operators guided by valuation

The previous recombination operator keeps the parents domain values within the offspring but “forgets” the actual solution computed by the solving of the CSP. Crossover efficiency may be increased for some problems by keeping the values corresponding to the CSP solution within the children. This mechanism can also be applied to the mutation operator in the same way.

F. Other operators

Further refinements can be provided to the operator:

\begin{itemize}
  \item Assuming $\rho = 1/2$ and the variables are boolean, “complementary” mutation is strictly identical to bit strings mutation.
  \item Recombination operator would be a more suitable terminology, as our operator is far from biological crossover.
\end{itemize}
• heuristics can sometimes be deduced from the objective function, like a mutation which alters genes by substituting values for greater ones taken from their initial domains if the objective function increases with the problem variables;
• interval sub-domains and operators that keep their structures can be used if the notion of interval has a meaning for the treated problem;
• combination of all the previous techniques might be used with some problems.

Performances can be increased with these kinds of mechanisms but too specific operators lack robustness and may only be used on small classes of problems, and too determinist ones reduce the exploration of the search space.

G. The ga_maximize procedure

The hybridization is generic: only a CSP formulation on finite domains of the problem is needed; no other assumption is required. Within the CLP framework, the optimization process can thus be provided to the user by a predicate analogous to the standard maximize predicate:

\[
g_{\text{maximize}}(\text{Goal}, \text{Variables}, \text{Eval}, \rho)
\]

where Goal stands for the CSP searching procedure, Variables for the list of finite domain variables of the problem and Eval for the evaluation of the solution computed by Goal. As well as for the standard maximize predicate, Goal is simply the labeling of the variables. \(\rho\) stands for our hybridization parameter \(\rho\).

G.1 Implementation

The novelty of the implementation of our hybrid genetic algorithm lies in the evaluation of the individuals: as an individual is made up of sub-domains, the domain of each variable is restricted by adding the constraint \(X_i \in G_i\) before Goal is called. If Goal is successful, Eval is the evaluation of the individual by the objective function, otherwise, its fitness is penalized (see section III-B).

G.2 Parameters setting

Beside the classic parameters of a GA (size of the population, number of generations or termination criterion, crossover and mutation probabilities, ...), our algorithm is parametrized by the degree of hybridization \(\rho\) which specifies the relative size of the sub-domains. All these parameters have default values in our implementation and can be easily modified.

IV. Application

We have tested our algorithm on a VRP (Vehicle Routing Problem) problem. VRP is a concoction of TSP (Travelling Salesman Problem) and scheduling problem: several tasks must be done at distinct locations and within given time windows; each task can be executed by some skilled engineers; some tasks must be performed before or at the same time as others; the problem lies in the production of a timetable for each engineer minimizing the time spent in travelling and waiting. We have chosen this problem for its intrinsic complexity and the huge size of its search space.

A. Formulation

We have naïvely formulated this problem into a CSP in the following way:

- Two domain variables are associated to each task \((i)\) one for the engineer who executes the task \((E_i)\) and the other for the date corresponding to the beginning of the task \((T_i)\); the domains of the variables are specified (skilled engineers and time windows).
- The precedence and synchronisation constraints are expressed on the \(T_i\)'s (equality and inequality built-in constraints).
- With this formulation, the fact that an engineer cannot execute two tasks at the same time is tricky to express. The following constraint is therefore added for each couple of tasks \((i,j)\):

\[
E_i = E_j \quad (1)
\]

\[
T_i + d_i + t \leq T_j \quad (2)
\]

\[
T_j + d_j + t \leq T_i \quad (3)
\]

\[
(1) \Leftrightarrow ((2) \lor (3)) \quad (4)
\]

where constraint 1 is true if both tasks are executed by the same engineer; \(d_i\) (resp. \(d_j\)) is the duration of task \(i\) (resp. \(j\)); \(t\) is the time spent in travelling between the locations of the tasks; Constraint 4 stipulates that either the two tasks are not performed by the same engineer, either only one of constraints 2 and 3 is true.

B. Results

![Fig. 2. Influence of \(\rho\) on the quality of the solution](image-url)
and branch&bound strategy provides a solution with cost 1300 in roughly the same time as our hybrid GA and its best solution for a 50 times greater duration is only 1177 (versus 387, best solution for the hybrid GA with fine tuned parameters).

V. Similar approaches

Other GA/CSP hybridization approaches have been experimented: [5] integrates CSP into the GA operators to generate only feasible solutions and reports good results for the TSP with a mutation operator performing local improvement (with CSP) and a crossover operator generating new individuals whenever the children are not feasible solutions. [6] solves timetabling problems with a similar method using very specific operators.

Our approach is more analogous to [7] which hybridizes a CSP with a Simplex algorithm: the method is generic and problem independent.

VI. Conclusion

We have introduced a novel optimization method which hybridizes CLP techniques within a genetic algorithm. Our algorithm can be applied to any CSP on finite domains. The first results are given and are encouraging enough to validate the approach.

This work will be followed by other experimentations on various combinatorial optimization problems and in particular on airflow traffic related problems [8]. Other approaches are also envisaged:

- for some problem with two distinct “dimensions”, one of them can be treated by a GA and the other by a CSP (timetabling problem for instance, dates with the GA and rooms with the CSP);
- integration of a “repairing” CSP method [9] into the GA operators to produce feasible individuals.

REFERENCES


ρ (ρ < 0.15) corresponding to single valued domains (GA alone). The best solutions are obtained for ρ around 0.2 and the algorithm has low efficiency for ρ > 0.4 because the solving of the CSP does not optimize its solutions and individuals are much more alike for high values of ρ. The graph shows that the set oriented crossover operator outperforms the classic one on most of ρ values.

Figure 3 shows the influence of crossover and mutation probabilities (Pc and Pm respectively) for the same problem with the two crossover operators ρ = 0.2. The parts of the graphs corresponding to very low mutation probabilities are truncated because the population hardly evolves with crossover alone. The overall superiority of the set oriented crossover is noticeable as the corresponding graph is smoother than the other one (more independent of Pm, Pc) for the lowest areas: 0.2 ≤ Pm ≤ 0.6 and 0.0 ≤ Pc ≤ 0.4. We can also notice that the best results are obtained for high values of Pm and low values of Pc, which is not conventional for genetic algorithms.

Finally, we may also notice that the same CSP formulation of this VRP problem solved with a standard search