Combine & conquer : genetic algorithm and CP for optimization

Nicolas Barnier, Pascal Brisset

Combine & Conquer: Genetic Algorithm and CP for Optimization

Nicolas Barnier, Pascal Brisset*

April 17, 1998

ABSTRACT We introduce a new optimization method based on a Genetic Algorithm (GA) combined 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 adaptation 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 experimentations adresses a naïve formulation of a Vehicle Routing Problem (VRP). The results are quite encouraging as we outperform CSP techniques and genetic algorithm alone on these formulations.
KEYWORDS: Optimization, Genetic Algorithms, Hybridization

1 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 determinist 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

*École Nationale de l'Aviation Civile, 7 avenue Édouard Belin, B.P. 4005, F–31055 Toulouse Cedex 4, France, E-mail: {barnier,brisset}@recherche.enac.fr

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 [Gol89] 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 adaptation, i.e. will not survive through the next generation).

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[DHS+88]). 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 higly constrained and have large search spaces. These two features exclude the direct and naïve use of a genetic algorithm or a CSP technique alone.

We suggest to take advantage of the two approaches by combining 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 X_1 and X_2 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 (subspace **a**) and two different individuals may correspond to the same solution (subspaces **d** and **e**). The ratio of the size of a subspace to the size of the whole search space (called ρ afterwards) is the essential parameter of the hybridization : one can continuously pass from a pure CSP search ($\rho = 1$) to a pure stochastic search ($\rho = 0$, *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 prob-

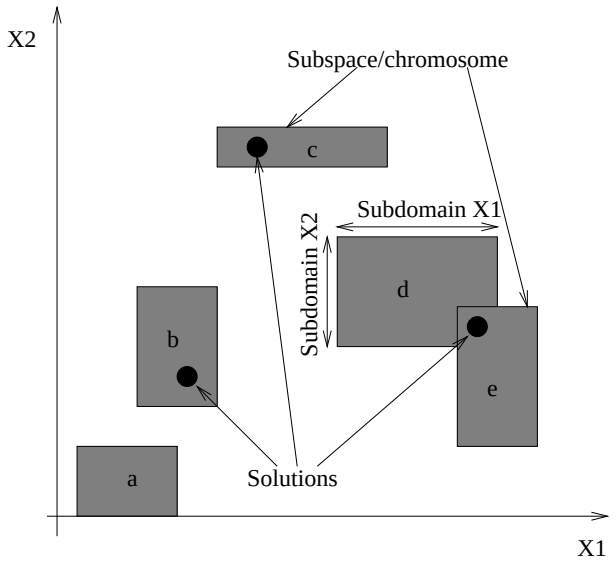

FIGURE 1. A two-dimensional search space. The grey rectangles (sub-domains) are the individuals of the GA. Solutions are searched inside these sub-domains by the CSP.

lem whose variables belong to X, assuming that a CLP(X) framework is provided. First, we recall briefly what CSP and genetic algorithms are, 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.

2 Context

We present in this section the two optimization techniques for which we introduce hybridization in the next section.

2.1 Constraints satisfaction problems (CSP)

We consider here a CSP formed by (X, D, C), X being the set of the n problem variables (X_1, X_2...), D the set of their respective finite domains ($D(X_1)$, $D(X_2)$...) and C a set of relations between these variables, *i.e.* the problem constraints. For an associated optimization problem, we also consider an objective function (or cost function) f and the constraint $f(X_1, X_2, ...) > c$ (for a maximization problem) where c is some constant which evolves according to the optimization strategy.

The CSP is formulated through the Constraint Logic Programming paradigm [Van89] and is implemented with the ECL^{i}PS^{e}system [ECL92] which provides all the "classic" constraints needed for CSPs: linear ones (`#=`, `#>`...) and many others (`alldistinct`, `element`...), and allows to easily define new ones (fine and direct handling of the domains, the coroutining...). The `maximize` predicate allows to optimize linear expressions by integrating the problem solving goal (most of the time the instantiation of the variables, *i.e.* the labeling) into a *branch & bound* algorithm.

Note that the choice of CLP is arbitrary and that any other CSP solver could be used instead. However, efficiency does not directly rely on this choice given the fact that CLP systems usually include high quality algorithms for constraint satisfaction.

2.2 Genetic algorithms

Genetic algorithms (GA) are a drastically simplified model of natural evolution in a given environment according to the Darwinian theory. They are described with terms very close to those of (biological) genetic vocabulary. We will therefore talk about a *population* of *individuals*. Each individual is made up of one *chromosome* whose n *genes* correspond to the n variables of the problem and hold its hereditary features. *Selection*, *crossover* and *mutation* operators are inspired by the homonymous biological processes.

For a given optimization problem, an individual stands for a point of the search space and is associated with its adaptation or *fitness* to a particular environment, *i.e.* the corresponding value of the (possibly scaled) objective function. The algorithm iteratively builds new generations of the population which evolves through selection, crossover (which is a recombination operator) and mutation (figure 2). Selection favours high fitness individuals whereas crossover and mutation ensure an efficient exploration of the search space.

The algorithm first randomly generates a population of some size, which is one of its parameter. To go from the current generation k to the next generation $k+1$, the following steps are repeated for all the members of the population. Couple of parents P_1 and P_2 are selected among the population according to their fitnesses. The crossover operator is applied on them with probability P_c and couples of children C_1 and C_2 are produced. Other individuals P are selected according to their fitnesses. The mutation operator is applied on them with probability P_m (P_m is generally one order of magnitude less than P_c) and mutants P' are produced. The fitnesses of the offspring are then evaluated before insertion in the new population. Different termination criteria may be used to end the algorithm: number of generations (constant time), convergence of the population...

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 through-

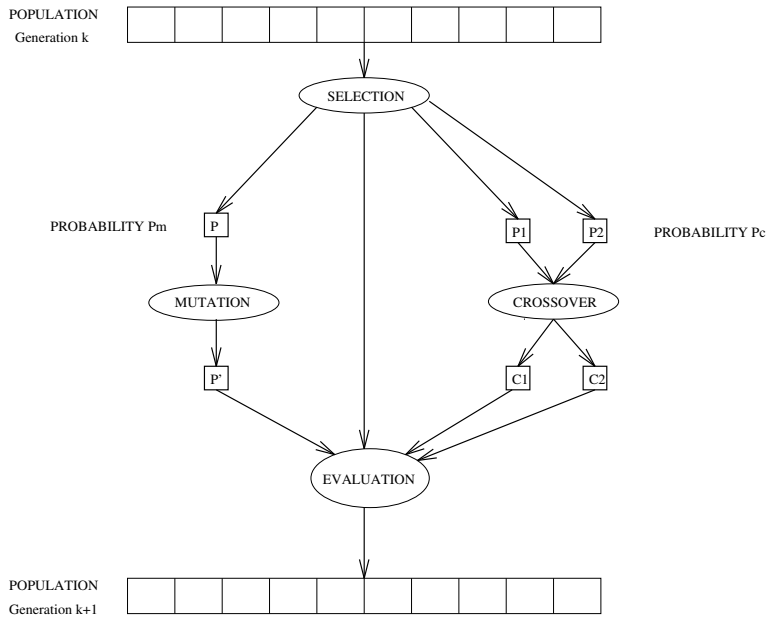

FIGURE 2. General principle of genetic algorithms

out the search space) and operators allowing to diversify the population (consequently to explore the search space) and to focus on the fittest individuals.

3 A mixed approach

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

3.1 Chromosome

An individual of the genetic algorithm is formed by a gene strings $G_1...G_n$, *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 possibly 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 ρ ($0 \leq \rho \leq 1$) 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 ρ 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 ρ 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.

3.2 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 (or not standard) way by the labeling of the variables. If a solution is found, the fitness is simply computed by

[1] One value is at least provided to each gene with this formula. Empty sub-domains would be of little interest.

TABLE 1. Classic mutation and crossover

X_1	X_2	X_3	Variables
1..7	1..9	1..5	Initial domains
1,2,3	4,7,8	1,3,4	Individual P_1
1,5,7	1,4,8	1,2,5	Individual P_2
1,2,3	1,7,8	1,3,4	Mutated individual P_1
1,2,3	4,7,8	1,2,5	First child of P_1 and P_2
1,5,7	1,4,8	1,3,4	Second child of P_1 and P_2

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 null). The same kind of penalty is applied when the CSP becomes inconsistent as the $X_i \in G_i$ constraints are added.

However, to prove that a sub-space cannot provide any solution might be very time consuming and the maximum 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 optimization during the solving of the sub-CSPs as described in section 2.1 because the genetic algorithm handles it.

3.3 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 [Gol89] can be used with very little changes to handle various data representations. We describe here the transposed classical operators implemented in our algorithm.

Crossover

As described in table 1, 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.

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[2]. 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 ρ is not close from 1/2. Nevertheless, next section describes a "set oriented" operator which can outperform classic ones on some problems.

3.4 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[3] operator (as described in table 2):

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

This recombination operator "shuffles" the genes of the parents in a way similar to the *uniform* crossover (n-point crossover with chromosomes of

[2]Assuming $\rho = 1/2$ and the variables are boolean, "complementary" mutation is strictly identical to bit strings mutation.

[3]Recombination operator would be a more suitable terminology, as our operator is far from biological crossover.

TABLE 2. Set oriented crossover

1,2,3	4,7,8	1,3,4	Parent
1,5,7	1,4,8	1,2,5	Parent
1,2,3,5,7	1,4,7,8	1,2,3,4,5	Union
2,3,5	1,4,8	2,3,4	First child
1,7	7	1,5	Rest
1,7,5	7,4,8	1,5,2	Second child

size n) and provides children as different as possible, re-using all the parents genes data and keeping constant sub-domains sizes.

3.5 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.

3.6 Other operators

Further refinements can be provided to the operator:

- 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.

3.7 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:

`ga_maximize(`*Goal* , *Variables* , *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 hybrization parameter ρ.

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 3.2).

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 ρ which specifies the relative size of the sub-domains. All these parameters have default values in our implementation and can be easily modified.

4 Application

We haved tested out 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 ingeneers 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.

4.1 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_is (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):

$$\begin{aligned} & E_i = E_j && (1)\\ \& \quad & T_i + d_i + t \leq T_j && (2)\\ \& \quad & T_j + d_j + t \leq T_i && (3)\\ \& \quad & (1) \Leftrightarrow ((2) \vee (3)) && (4) \end{aligned}$$

  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.

Note that this formulation is "not" a good one to solve this problem in CSP. It cannot compete with approaches using global constraints (provided by Ilog Scheduler for example). We use it only to compare results obtained with or without the combination with a GA.

4.2 Results

Figure 3 illustrates the influence of the degree of hybridization ρ on a 10-engineer 30-task instance of the VRP for the classic crossover and the set oriented crossover with a population size of 60 evolved during 50 generations, and probability 0.4 and 0.2 for crossover and mutation respectively. The graph represents the cost of the best individual for this minimization problem.

The infinite costs stand for the inability of the genetic algorithm to generate feasible individuals for low values of ρ ($\rho < 0.15$) corresponding to single valued domains (GA alone). The best solutions are obtained for ρ around 0.2 and the algorithm has low efficiency for $\rho > 0.4$ 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 4 shows the influence of crossover and mutation probabilities (P_c and P_m respectively) for the same problem with the two crossover operators and $\rho = 0.2$. The parts of the graphs corresponding to very low mutation probabilities are truncated because the population hardly evolves

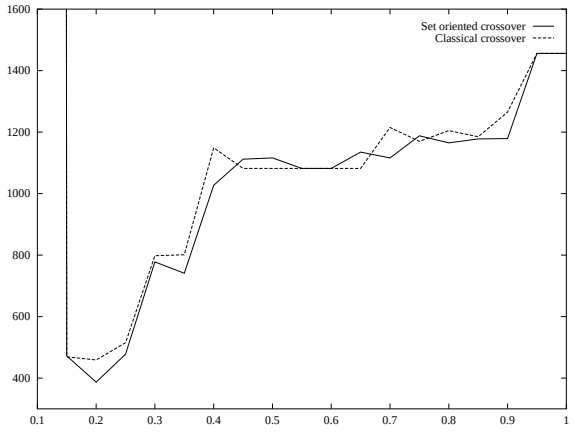

FIGURE 3. Influence of ρ on the quality of the solution

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 P_c and P_m) for the lowest areas: $0.2 \leq P_m \leq 0.6$ and $0.0 \leq P_c \leq 0.4$. We can also notice that the best results are obtained for high values of P_m and low values of P_c, 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 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).

One more time, we do not (and we do not want to) show that this method is a good one to solve a VRP problem. We only prove that a simple CSP formulation of an optimisation problem can be executed more efficiently using a stochastic algorithm.

5 Similar approaches

Other GA/CSP hybridization approaches have been experimented: [Küc93] 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. [BEW95] solves timetabling problems with a similar method using very specific operators.

A similar idea is proposed by [PG96] in the context of local search: neighborhoods where a local move is possible are specified by constraints and explored by a classic branch-and-bound algorithm.

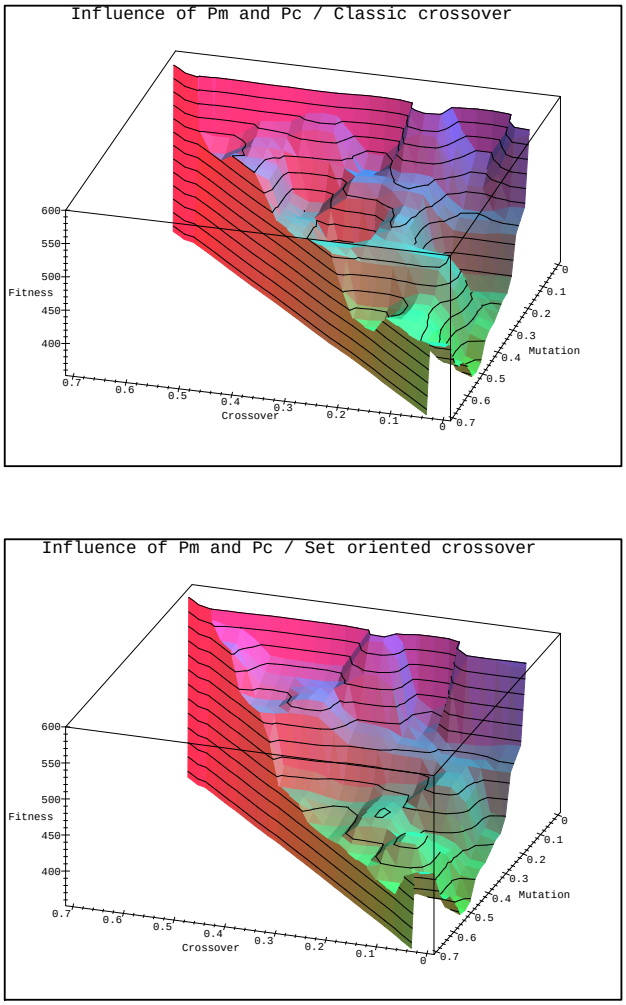

FIGURE 4. Influence of crossover and mutation probabilities

Our approach is more analogous to [THG+95] which hybridizes a CSP with a Simplex algorithm: the method is generic and problem independent.

6 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 [All96]. 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 [MJPL94] into the GA operators to produce feasible individuals.

7 References

[All96] Jean-Marc Alliot. Techniques d'optimisation stochastique appliquées aux problèmes du contrôle aérien. Thèse d'habilitation, Université de Toulouse Paul Sabatier, 1996.

[BEW95] Edmund K. Burke, Dave G. Elliman, and Rupert F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. Technical report, University of Nottingham, 1995. Technical Report NOTTCS-TR-95-8.

[DHS+88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. The constraint logic programming language CHIP. In *Int. Conf. Fifth Generation Computer Systems*, volume 1, pages 693–702, Tokyo, Japan, 1988.

[ECL92] ECL^{i}PS^e user manual (ECRC Common Logic Programming System), 1992.

[Gol89] David Goldberg. *Genetic Algorithms.* Addison Wesley, 1989.

[Küc93] Volker Küchenhoff. Novel search and constraints – an integration. Technical Report ECRC–CORE–93–9, European Computer-Industry Research Centre, 1993.

[MJPL94] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. In Eugene C. Freuder and Alan K. Mackworth, editors, *Constraint-based Reasoning*. MIT Press, 1994.

[PG96] G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Proc. of the 2nd international conference on Principles and Practice of Constraint Programming*, pages 353–366, Cambridge, Ma, USA, 1996.

[THG+95] Hajian M T, El-Sakkout H H, Wallace M G, Richards E B, and Lever J M. Towards a closer integration of finite domain propagation and simplex-based algorithms. 1995. AI Maths 96 Florida Atlantic University.

[Van89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, 1989.