Hybrid CSP Solving

Eric Monfroy[1,3], Frédéric Saubion[2], and Tony Lambert[2,3]

[1] Universidad Técnica Federico Santa María, Valparaíso, Chile
`Firstname.Name@inf.utfsm.cl`
[2] LERIA, Université d'Angers, France
`Firstname.Name@univ-angers.fr`
[3] LINA, Université de Nantes, France
`Firstname.Name@univ-nantes.fr`

Abstract. In this paper, we are concerned with the design of a hybrid resolution framework. We develop a theoretical model based on chaotic iterations in which hybrid resolution can be achieved as the computation of a fixpoint of elementary functions. These functions correspond to basic resolution techniques and their applications can easily be parameterized by different search strategies. This framework is used for the hybridization of local search and constraint propagation, and for the integration of genetic algorithms and constraint propagation. Our prototype implementation gave experimental results showing the interest of the model to design such hybridizations.

1 Introduction

The resolution of constraint satisfaction problems (CSP) appears nowadays as a very active and growing research area. Indeed, constraint modeling allows both scientists and practitioners to handle various industrial or academic applications (e.g., scheduling, timetabling, boolean satisfiability, ...). In this context, CSP are basically represented by a set of decision variables and a set of constraints among these variables. The purpose of a resolution process is therefore to assign a value to each variable such that the constraints are satisfied. We focus here on discrete CSP in which variables take their values over finite sets of integers. Discrete CSP are widely used to model combinatorial problems, and, by extension, combinatorial optimization problems, where the purpose is to find a solution of the problem which optimizes (minimizes or maximizes) a given criterion, usually encoded by an objective function.

The resolution of CSP involves many different techniques issued from different scientific communities: computer science, operation research or applied mathematics. Therefore, the principles and purposes of the proposed resolution approaches are very diverse. But, one may classified these methods in two families, which differ on a fundamental aspect of the resolution: complete methods whose purpose is to provide the whole set of solutions and incomplete methods which aim at finding one solution. On the one hand, complete methods, thanks to an exhaustive exploration of the search space, are able to demonstrate that a given problem is not satisfiable while incomplete methods will be ineffective in that case. On the other hand, incomplete methods, which explore only some parts of the search space with respect to specific heuristics, are often more efficient to obtain a solution and, moreover, for large instances with huge search space they appear as the only usable methods since resolution becomes intractable for complete methods.

A common idea to get more efficient and robust algorithms consists in combining several resolution paradigms in order to take advantage of their respective assets. Such combinations are now more and more studied in the constraint programming community [21,30,31,32].

Complete solvers usually build a search tree by applying domain reduction, splitting and enumeration. Local consistency mechanisms [24,27] allow the algorithms to prune the search space by deleting inconsistent values from variables domains. Such solvers have been embedded in constraint programming languages (Chip [2], Ilog Solver [19], CHOCO [22], ...) which provide a general framework for constraint modeling and resolution. Unfortunately, this approach requires an important computational effort and therefore encounters some difficulties with large scale problems. These performances can be improved by adding more specific techniques such as efficient constraint propagation algorithms, global constraints, ... We refer the reader to [5,12,25,9] for an introduction to constraint programming.

Incomplete methods mainly rely on the use of heuristics providing a more efficient exploration of interesting areas of the search space in order to find some solutions. Unfortunately, these approaches do not ensure to collect all the solutions nor to detect inconsistency. This class of methods, known as metaheuristics, covers a very large panel of resolution paradigms from evolutionary algorithms to local search techniques. We refer the reader to [1,29,18] for an overview of these different algorithms and their applications to combinatorial optimization problems. [11] presents an overview of possible uses of local search in constraint programming.

Due to their different algorithmic process, these approaches often differ in their representation of the search space and in the benefit they get from the structure of the problem. Therefore, hybridizations of these techniques have often been tackled through heterogeneous combinations of coexisting resolution processes, with a master-slave like management, and are often related to specific problems or class of problems. Such designs lead to intricate solvers whose behavior is sometimes hard to analyze and which offer few flexibility in order to handle other problems.

Our purpose is to present in this paper a general hybridization framework in order to combine usual complete constraint programming resolution techniques, namely constraint propagation and splitting, together with metaheuristics optimization techniques, namely local search and genetic algorithms. This framework is based on the original mathematical framework proposed by K.R. Apt in [4]. In this framework, basic resolution processes are abstracted by functions over an ordered structure. This allows us to consider the different resolution agents at a same level and to study more precisely various hybridization strategies.

In this paper, we first focus on hybridization of constraint propagation techniques (CP) and local search (LS) for constraint satisfaction problems, based on preliminary results [28,23], and then we present a new hybridization of CP and genetic algorithms (GA) for constraint optimization problems. As mentioned above, the main difference between these two classes of problems will consist of different evaluation or fitness function which have to take into account the satisfaction problem (minimization of the number of violated constraint) and eventually an optimization criterion.

This paper is organized as follows. In Section 2 we recall the basic notions related to CSPs, to complete methods (more especially constraint propagation based methods) and incomplete methods (local search and genetic algorithms) for solving CSPs. In Section 3, we present the uniform computational framework that we extend later for hybridization of CP and LS (Section 4) and hybridization of CP and GA (Section 5). Section 6 shows some experimental results of hybridization, obtained with our generic constraint system. Finally, we conclude and propose some perspectives in Section 7.

2 Constraint Satisfaction Problems

In this section, we first recall the basic notions related to Constraint Satisfaction Problems (CSP) [34]. We describe then, three important resolution approaches that we will use in our hybridization framework: complete resolution techniques based on constraint propagation, local search methods, and genetic algorithms.

A CSP is a tuple (X, D, C) where $X = \{x_1, \cdots, x_n\}$ is a set of variables that takes their values in their respective domains $D_1, \cdots, D_n$. A constraint $c \in C$ is a relation $c \subseteq D_1 \times \cdots \times D_n$. D denotes the Cartesian product of $D_1 \times \cdots \times D_n$ and C the union of its constraints.

A tuple $d \in D$ is a solution of a CSP (X, D, C) if and only if $\forall c \in C, d \in c$.

Note that, without any loss of generality, we consider that each constraint is over all the variables $x_1, \ldots, x_n$. However, one can consider constraints over some of the x_i. Then, the notion of scheme [4,3] or projections can be used to denote sequences of variables.

2.1 Solving CSP with Complete Resolution Techniques

Complete resolution techniques generally perform a systematic exploration of the search space which obviously corresponds to the set of possible tuples D. To avoid and reduce the combinatorial grow up of this extensive exploration, these methods use particular techniques to prune the search space. Constraint propagation, one of the most popular of these pruning techniques, is based on local consistency properties. A local consistency (e.g., [24,27]) is a property of the constraints and variables which is used by the search mechanisms to delete values from variables domains which violate constraints and thus, cannot lead to solutions. There are several levels of local consistencies that consider one or several constraints at a time: we may mention node consistency and arc consistency [24] as famous examples of local consistencies.

But constraint propagation is not sufficient for fully defining a complete solver and split techniques are added to obtain a complete search algorithm. Constraint propagation consists in examining a subset C' of C (generally C' is restricted to one constraint) to delete some inconsistent values (from a local consistency point of view) of the domains of variables appearing in C'. These domain reductions are then used to reduce variables appearing in $C \setminus C'$. Hence, reductions are propagated to the entire CSP. When no more propagation is possible and the solutions are not reached, the CSP is split into sub-CSPs on which propagation is applied again, and so on until the solutions are reached. Generally, the domain of a variable is split into two sub-domains leading to two sub-CSPs. One of the most popular strategy of splitting is enumeration that consists in restricting one of the sub-domain to one value, the other sub-domain being the initial domain without this value.

```
solve(CSP):
        while not solved do
            constraint propagation
            if not solved
              then split
                  search
            endif
        endwhile
```

Fig. 1. A simple constraint solving algorithm

Figure 1 shows a simple but generic solve algorithm based on constraint propagation. The "search" function consists in calls to the solve function: search manages the sub-CSPs created by split. Usual search is depth or breadth first search. "solved" is a Boolean that is set to true when the CSP is found inconsistent, or when the wish of the user is reached: one solution, all solutions, or an optimum solution have been computed.

2.2 Solving CSP with Local Search

Local search techniques usually aim at solving optimization problems and have been widely used for combinatorial problems [1,29,18]. In the particular context of constraint satisfaction, these methods are applied in order to minimize the number of violated constraints and thus to find a solution of the CSP. A local search algorithm, starting from a given configuration, explores the search space by a sequence of moves. At each iteration, the next move corresponds to the choice of one of the so-called neighbors of the current state. This neighborhood often corresponds to small changes of the current configuration. Moves are guided by a fitness function which evaluates their benefit from the optimization point of view, in order to reach a local optimum. In the next sections, we attempt to generalize the definition of local search.

For the resolution of a CSP (X, D, C), the search space can be usually defined as the set of possible tuples of $D = D_1 \times \cdots \times D_n$ and the neighborhood is a mapping $\mathcal{N} : D \rightarrow 2^D$. This neighborhood function defines indeed possible moves and therefore fully defines the exploration landscape. The fitness (or evaluation) function *eval* is related to the notion of solution and can be defined as the number of constraints c such that $d \notin c$ (d being a tuple from D).

As mentioned above, in the context of constraint satisfaction problems, the evaluation function corresponds to the minimization of the number of violated constraint. Therefore, given a configuration $d \in D$, representing an assignment, a basic local search move can either lead to an increase of the number of satisfied constraints (i.e., choose $d' \in \mathcal{N}(d)$ such that $eval(d') < eval(d)$) or to any other configuration which does not improve the evaluation function. These two possible steps can be interpreted as intensification or diversification of the search and local search algorithms are often based on the management of these basic heuristics by introducing specific control features. Therefore, a local search algorithm can be considered as a sequence of moves on a structure ordered according to the evaluation function.

2.3 Genetic Algorithms

Evolutionary algorithms are mainly based on the notion of adaptation of a population of individuals to a criterion using evolution operators like crossover [15].

Based on the principle of natural selection, *Genetic Algorithms* [17,20] have been quite successfully applied to combinatorial problems such as scheduling or transportation problems. The key principle of this approach states that, species evolve through adaptations to a changing environment and that the gained knowledge is embedded in the structure of the population and its members, encoded in their chromosomes. If individuals are considered as potential solutions to a given problem, applying a genetic algorithm consists in generating better and better individuals with respect to the problem by selecting, crossing, and mutating them. This approach reveals very useful for problems with huge search spaces. We had to adapt some basic techniques and slightly modify some definitions to fit our context but we refer the reader to [26] for a survey.

A genetic algorithm consists of the following components:

- a representation of the potential solutions: in most cases, individuals will be strings of bits representing its *genes*,
- a way to create an initial population,
- an *evaluation function eval*: the evaluation function rates each potential solution with respect to the given problem,
- genetic operators that define the composition of the children: two different operators will be considered: *Crossover* allows to generate new individuals(the offsprings) by crossing individuals of the current population

(the parents), *Mutation* arbitrarily alters one or more genes of a selected individual,
- parameters: population size p_{size} and probabilities of crossover p_c and mutation p_m.

In the context of GA, for the resolution of a given CSP (X, D, C), the search space can be usually defined with the set of tuples $D = D_1 \times \cdots \times D_n$. We consider a populations g, which is a subset of D, such that its cardinality is i. An element $s \in g$ is an individual and represents a potential solution to the problem.

Here, we will use the hybridization CP+GA in the context of constraint optimization problems. Therefore, evaluation functions (related to previous *eval* function but extended to optimization problems) provide information about the quality of an individual and so, of a population. Thus, these functions have to handle both the constraints of the problem and the optimization criterion.

A tuple in D is evaluated on an ordered set E whose lower bound corresponds indeed to the evaluation of an optimal solution. Therefore a fitness function $eval_{ind}\colon D \rightarrow E$ is such that $eval_{ind}(s)$ takes into account the number of unsatisfied constraints and the optimization criterion (abstraction of the objective function) for an individual s. We consider that E is ordered such that if s is a feasible solution (i.e. all constraints are satisfied) then $eval_{ind}(s)$ is restricted to its optimization evaluation. We denote $s <_{eval} s'$ the fact that $eval_{ind}(s) <_E eval_{ind}(s')$. When solving optimization problems we have to isolate the best solution yet found. Thus, s is the current solution for a population g if $\forall s' \in g$, $s \leq_{eval} s'$.

We extend this notion of fitness to population by $eval_{gen}\colon 2^D \rightarrow F$ such that $eval_{gen}(g)$ represents the evaluation of the individuals of the population g. The set F is ordered such that: g is a population solution if it contains an individual solution (i.e at least one of the components of g has an evaluation restricted to its optimization evaluation).

This $eval_{gen}(g)$ function, can represent for example, the sum of all the fitness of each individual, the sum of squares, or can be restricted to the best individual in the population. Furthermore, we denote $g <_{eval} g'$ the fact that $eval_{gen}(g) <_F eval_{gen}(g')$.

3 A Uniform Computational Framework

As described in the previous section, different techniques may be used to solve CSP (and many others which are not recalled here since they are out of the scope of this paper). Our purpose is to integrate the various involved computation processes in a uniform description framework. Since we want to combine all our resolution technique at a same level, the chaotic iterations model of K.R. Apt particularly fits our requirements. Therefore, the purpose of this section is to formalize the general computation scheme presented in Section 2.1, and to prepare it for hybridization of techniques.

In [4,3] K.R. Apt proposed a general theoretical framework for modeling constraint propagation. In this context, domain reduction corresponds to the computation of a fixpoint of a set of functions over a partially ordered set.

These *domain reduction functions* are monotonic and inflationary functions; they abstract the notion of constraint.

Example 1 (Domain reduction functions). Consider three Boolean variables X, Y, and Z and the Boolean constraint $and(X, Y, Z)$ such that $and(X, Y, Z)$ represents the Boolean relation $X \wedge Y = Z$. An example of reduction function for the constraint $and(X, Y, Z)$ can be defined by: if the domain of Z is $\{1\}$, then the domains of X and Y must be reduced to $\{1\}$.

Here is another example of reduction functions for linear equalities over integer numbers:

$$\begin{gathered}\text{if } x < y, x \in [l_x..r_x], y \in [l_y..r_y] \\ \text{we can reduce the domain of } x \text{ and } y \text{ as follows:} \\ x \in [l_x..min(r_x, r_y - 1)], y \in [max(l_y, l_x + 1)..r_y]\end{gathered}$$

The computation of the least common fixpoint of a set of functions F can be achieved by the Generic Iteration algorithm (GI) described in Figure 2. In the GI algorithm, G represents the current set of functions still to be applied ($G \subseteq F$), d is a partially ordered set (the domains in case of CSP).

GI: Generic Iteration Algorithm

```
$d := \perp$;
$G := F$;
While $G \neq \emptyset$ do
        choose $g \in G$;
        $G := G - \{g\}$;
        $G := G \cup update(G, g, d)$;
        $d := g(d)$;
endwhile
```

where for all G, g, d, the set of functions $update(G, g, d)$ from F is such that:

- $\{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \subseteq update(G, g, d)$.
- $g(d) = d$ implies that $update(G, g, d) = \emptyset$.
- $g(g(d)) \neq g(d)$ implies that $g \in update(G, g, d)$

Fig. 2. The Generic Iteration Algorithm

Suppose that all functions in F are inflationary ($x \sqsubseteq f(x)$ for all x) and monotonic ($x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$ for all x, y) and that $(D, \sqsubseteq)$ is finite. Then, every execution of the **GI** algorithm terminates and computes in d the least common fixpoint of the functions from F (see [4]).

Note that in the following we consider only partial orderings.

Constraint propagation is now achieved by instantiating and "feeding" the **GI** algorithm:

- the $\sqsubseteq$ partial ordering is instantiated by $\supseteq$, the usual set inclusion,
- $d := \bot$ corresponds to $d := D_1 \times \ldots \times D_n$, the Cartesian product of the domains of the variables from the initial CSP to be solved,
- F is a set of domain reduction functions which abstract the constraints in order to reduce domains of variables.

This algorithm allows us to compute the smallest box (i.e., Cartesian product of domains) with respect to the given domain reduction functions that contains the solutions of the initial CSP.

At this point, as shown in Figure 1, the exploration of the reduced domains is continued by interleaving splitting and again propagation phases.

In order to obtain a more uniform and generic framework, our purpose is to integrate the splitting process as a reduction function inside the **GI** algorithm. This is motivated by the fact that we want to manage constraint propagation, split and local search (respectively genetic algorithms) at the same level. To this end, we will extend the notion of CSP to sampled CSP (respectively CSP with genetic factor) on which an other type of reduction functions will be applied to mimic basic operations of local search algorithms (respectively basic operations of genetic algorithms).

Therefore, we have to introduce new functions in the generic iteration algorithm including splitting operators and local search strategies (respectively a genetic algorithm process). Then, these search methods can be viewed as the computation of a fixpoint of a set of functions on an ordered set. But, these new operators require also a new computation structure and the first step of our work consists in defining this main structure.

4 CP+LS

Extending the framework described above (Section 3), we propose here a computational structure taking into account both constraint programming and local search basic resolution processes. CSPs and search paths are embedded in this new computation structure. Some reduction functions that achieve constraint propagation, split, and local search are then introduced to compute over this structure.

4.1 Sampling the Search Space

Domain reductions and splits apply on domains of values: they transform Cartesian product of the domains. Local search acts on a different structure which usually corresponds to points of the search space. Here, we propose a more general and abstract definition based on the notion of sample.

Definition 1 (Sample). *Consider a CSP (X, D, C). A* sample function ε *is a function $\varepsilon : D \to 2^D$. By extension, $\varepsilon(D)$ denotes the set $\bigcup_{d \in D} \varepsilon(d)$.*

Generally, $\varepsilon(d)$ is restricted to d and the set of samples is thus the search space D ($\varepsilon(D) = D$). However, $\varepsilon(d)$ can also be defined as a scatter of tuples around d, an approximation covering d, or a box covering d (e.g., for continuous domains). Moreover, it is reasonable to impose that $\varepsilon(D)$ contains all the solutions. Indeed, the search space D is abstracted by $\varepsilon(D)$ to be used by LS.

In this context, a local search can be fully defined by:

– a neighborhood function on $\varepsilon(D)$ which computes the neighbors (i.e., a set of samples) for each sample of $\varepsilon(D)$;
– and the set of local search paths. Each path is composed of a sequence of visited samples and represents moves from neighbors to neighbors.

Given a neighborhood function $\mathcal{N}: \varepsilon(D) \to 2^{\varepsilon(D)}$, we define the set of possible local search paths as $\mathcal{LS}_D =$

$$\bigcup_{i>0} \{p = (s_1, \cdots, s_i) \in \varepsilon(D)^i \mid \forall j,\ 1 \le j < i-1,\ s_{j+1} \in \mathcal{N}(s_j) \text{ and } s_1 \in \varepsilon(D)\}$$

The fundamental property of local search relies on its exploration based on the neighborhood relation.

From a practical point of view, a local search is limited to finite paths with respect to a stopping criterion: this can be a fixed maximum number of iterations (i.e., the length of the path) or, in our context of CSP resolution, the fact that a solution has been reached.

For this concern, according to Section 2.2, we consider an evaluation function $eval: \varepsilon(D) \to I\!N$ such that $eval(s)$ represents the number of constraints unsatisfied by the sample s. Moreover, we impose that $eval(s)$ is equal to 0 iff s is a solution. We denote $s <_{eval} s'$ the fact that $eval(s) < eval(s')$.

Therefore, from a LS point of view, a result is either a search path leading to a solution or a search path of a maximum given size. According to this fact, we define an order on local search paths as follows:

Definition 2 (local search ordering). *We consider an order $\sqsubseteq_{ls}$ on $\mathcal{LS}_D$ defined by:*

$(s_1, \ldots, s_n) \sqsubseteq_{ls} (s_1, \ldots, s_n)$
$(s'_1, \ldots, s'_m) \sqsubseteq_{ls} (s_1, \ldots, s_n)$ *if* $n > m$ *and* $\forall j, 1 \le j \le m, eval(s'_j) \ne 0$ *and* $\forall i, 1 \le i \le n, eval(s_i) \ne 0$
$(s'_1, \ldots, s'_m) \sqsubseteq_{ls} (s_1, \ldots, s_n)$ *if* $eval(s_n) = 0, \forall i, 1 \le i \le n-1, eval(s_i) \ne 0$ *and* $\forall j, 1 \le j \le m, eval(s'_j) \ne 0$

The following example illustrates the notion of results from a LS process.

Example 2 (LS paths). Consider $p_1 = (a, b)$, $p_2 = (a, c)$ and $p_3 = (b)$ three elements of $\mathcal{LS}_D$ such that $eval(b) = 0$ (i.e., b is a solution). Then, these three paths correspond to possible results of a local search of size 2, they are not comparable with respect to Definition 2.

4.2 Computation Structure

We can now define the structure required for the hybridization of local search and constraint solving. To this end, we instantiate the abstract framework of K.R. Apt described in Section 2.1.

Definition 3 (Sampled CSP). *A* sampled CSP *(sCSP) is defined by a triple (D, C, p), a sample function ε, and a local search ordering $\sqsubseteq_{ls}$ where*

- $D = D_1 \times ... \times D_n$
- $\forall c \in C, c \subseteq D_1 \times \ldots \times D_n$
- $p \in \mathcal{LS}_D$

Note that, in our definition, the local search path p should be included in the box defined by $\varepsilon(D)$. We denote $SCSP$ the set of $sCSP$. We can now define an ordering relation on the sampled structure $(SCSP, \sqsubseteq)$.

Definition 4 (Order over sampled CSPs). *Given two sCSPs $\psi = (D, C, p)$ and $\psi' = (D', C, p')$,*

$$\psi \sqsubseteq \psi' \quad \mathit{iff} \quad D' \subseteq D \ \mathit{or}\ (D' = D \ \mathit{and}\ p \sqsubseteq_{ls} p').$$

This relation is extended on 2^{SCSP} as follows:

$$\{\phi_1, \ldots, \phi_k\} \sqsubseteq \{\psi_1, \ldots, \psi_l\} \quad \mathit{iff} \quad \forall \phi_i, (\exists \psi_j,\ \phi_i \sqsubseteq \psi_j \ \mathit{and} \ \nexists \psi_j,\ \psi_j \sqsubset \phi_i)$$

where $i \in [1..k], j \in [1..l]$.

Note that this partial ordering on sCSPs could also be extended by considering an order on constraints; this would enable constraint simplifications. But this is out of the scope of our hybridization.

We denote ΣCSP the set 2^{SCSP} which constitutes the key set of our computation structure. We denote σCSP an element of ΣCSP. A σCSP is thus a set of sCSPs. As in [4], we define the least element $\perp$ as $\{(D, C, p)\}$, i.e., the initial σCSP to be solved.

4.3 Solutions

Since our framework is dedicated to CSP solving, we must define precisely the notion of solution accordingly to the previous computation structure. These notions are well defined for complete methods and incomplete methods.

From the complete resolution point of view, a solution of a CSP is a tuple from the search space which satisfies all the constraints. For local search, the notion of solution is related to the evaluation function *eval* which defines a solution as an element s of $\varepsilon(D)$ such that $eval(s) = 0$.

Definition 5 (Solutions). *Given a sCSP $\psi = (D, C, p)$, the set of solutions of ψ is defined by:*

- *for constraint propagation (CP) based solvers:*

$$Sol_D(\psi) = \{d \in D | \forall c \in C,\ d \in c\}$$

- *for local search (LS):*

$$Sol_{\mathcal{LS}_D}(\psi) = \{(s_1, \cdots, s_n) \in \mathcal{LS}_D \,|\, eval(s_n) = 0\}$$

– *for a LS/CP hybrid solver:*

$$Sol(\psi) = \{(d, C, p) | d \in Sol_D(\psi) \text{ or } p \in Sol_{\mathcal{LS}_D}(\psi)\}$$

This notion is extended to any σCSP Ψ by:

$$Sol(\Psi) = \bigcup_{\psi \in \Psi} Sol(\psi)$$

4.4 Reduction Functions Definitions and Properties

The computation structure ΣCSP has been defined for integrating CP and LS and we have now to define our hybrid functions which will be used in the GI algorithm. Given a σCSP $\Psi = \{\psi_1, \cdots, \psi_n\}$ of ΣCSP, we need to define functions on Ψ which correspond to domain reduction, split, and local search. These functions may apply on several sCSPs ψ_i of Ψ, and for each ψ_i on some of its components. Since we consider here finite initial CSPs, note that our structure is a finite partial ordering.

Definition 6 (Domain reduction function). *A* domain reduction function *red is a function:*

$$\begin{aligned} red\colon \Sigma CSP &\to \Sigma CSP \\ \{\psi_1, \cdots, \psi_n\} &\mapsto \{\psi'_1, \cdots, \psi'_n\} \end{aligned}$$

such that $\forall i \in [1 \cdots n]$:

– *either $\psi_i = \psi'_i$*
– *or $\psi_i = (D, C, p)$, $\psi'_i = (D', C, p')$ and $D \supseteq D'$ and $Sol_D(\psi_i) = Sol_D(\psi'_i)$.*

Note that this definition insures that $\{\psi_1, \cdots, \psi_n\} \sqsubseteq red(\{\psi_1, \cdots, \psi_n\})$ and that the function is inflationary and monotonic on $(\Sigma CSP, \sqsubseteq)$. Note also that by definition $p' \in \mathcal{LS}_{D'}$. This definition allows one to specify a function that reduces several domains of several *sCSPs* of a σCSP at the same time. From a constraint programming point of view, a reduction function preserves the solution set of the initial CSP: no solution of the initial CSP is lost by a domain reduction function. This is also the case for domain splitting as defined below.

Definition 7 (Domain splitting). *A domain splitting function sp on ΣCSP is a function such that for all $\Psi = \{\psi_1, \ldots, \psi_n\} \in \Sigma CSP$:*

a. $sp(\Psi) = \{\psi'_1, \ldots, \psi'_m\}$ *with* $n \leq m$,
b. $\forall i \in [1..n]$,
- *either* $\exists j \in [1..m]$ *such that* $\psi_i = \psi'_j$
- *or there exist* $\psi'_{j_1}, \ldots, \psi'_{j_h}$, $j_1, \ldots, j_h \in [1..m]$ *such that*

$$Sol_D(\psi_i) = \bigcup_{k=1..h} Sol_D(\psi'_{j_k})$$

c. and, $\forall j \in [1..m]$,
- *either* $\exists i \in [1..n]$ *such that* $\psi_i = \psi'_j$
- *or* $\psi'_j = (D', C, p')$ *and there exists* $\psi_i = (D, C, p)$, $i \in [1..n]$ *such that* $D \supset D'$.

Conditions a. and b. ensure that some sCSPs have been split into sub-sCSPs by splitting some of their domains (one or several variable domains) into smaller domains without discarding solutions (defined by the union of solutions of the ψ_i). Condition c. ensures that the search space does not grow: every domain of the sCSPs composing Ψ' is included in one of the domain of some sCSP composing Ψ. Note that the domain of several variables of several sCSPs can be split at the same time.

Definition 8 (Local Search). *A local search function* λ_N *is a function*

$$\begin{aligned} \lambda_N \colon \Sigma CSP &\to \Sigma CSP \\ \{\psi_1, \cdots, \psi_n\} &\mapsto \{\psi'_1, \cdots, \psi'_n\} \end{aligned}$$

where

- *N is the maximum number of consecutive moves*
- $\forall i \in [1..n]$
  - *either* $\psi_i = \psi'_i$
  - *or* $\psi_i = (D, C, p)$ *and* $\psi'_i = (D, C, p')$ *with* $p = (s_1, \cdots, s_k)$ *and* $p' = (s_1, \cdots, s_k, s_{k+1})$ *such that* $s_{k+1} \in \mathcal{N}(s_k) \cap D$ *and* $k + 1 \leq N$.

The parameter N represents the maximum length of a local search path, i.e., the number of moves allowed in a usual local search process. A local search function may try to improve the sample of one or several sCSPs at once. Even when λ_N tries to reduce ψ_i, note that $\psi_i = \psi'_i$ may happen when:

1. $p \in Sol_{\mathcal{LS}_D}(\psi)$: the last sample s_n of the current local search path cannot be improved using λ_N,
2. the length n of the search path is such that $n = N$: the maximum allowed number of moves has been reached,
3. λ_N is the identity function on ψ_i, i.e., λ_N does not try to improve the local search path of the sCSP ψ_i. This might happen when no possible move can be performed (e.g., a descent algorithm has reached a local optimum or all neighbors are tabu in a tabu search algorithm [14]).

4.5 σCSPs Resolution

For the complete solving of a σCSP $\{(D_1 \times \ldots \times D_n, C, p)\}$ the **GI** algorithm must now be instantiated as follows:

- the $\sqsubseteq$ ordering is instantiated by the ordering given in Definition 4,
- $d := \perp$ corresponds to $d := \{(D_1 \times \ldots \times D_n, C, p)\}$,

– F is a set of given monotonic and inflationary functions as defined in Section 4.4: domain reduction functions (extensions of common domain reduction functions for CSPs), domain splitting functions (usual split mechanisms integrated as reduction functions), and local search functions (e.g., functions for descent, tabu search, ...).

We now propose an instantiation of the function schemes presented in the previous section. From an operational point of view, reduction functions have to be applied on some selected $sCSP$s of a given σCSP. More practically, we build the functions on sCSPs and then extend them on σCSPs. Thus, a function on ΣCSP will be driven by an operator selecting the sCSPs of a σCSP.

We now define these selection operators. Given a selection function $select$: $A \to 2^B$ let us consider a function $f^{select}: A \to C$ such that $f^{select}(x) = g(y), y \in select(x)$ where $g: B \to C$. Therefore, f^{select} can be viewed as a non deterministic function. Formally, we may associate to any function f^{select} a family of deterministic functions $(f^i)_{i>0}$ such that $\forall x \in A, \forall y \in select(x), \exists k > 0, f^k(x) = g(y)$. If we consider finite sets A and B then this family is also finite.

Indeed, each σCSP that can result from the application of some functions on the initial σCSP requires all reduction functions (defined for the initial σCSP) to model the different possible executions of the resolution process. In other terms, consider an sCSP ψ_i of a σCSP Ψ; a set F' of functions can apply on ψ_i through Ψ (through the sCSP selection process). If a new sCSP ψ_j can be created (e.g., by split), then the functions of F' are also required to be applied on ψ_j through Ψ (again, through the sCSP selection process). However, ψ_j will may be not be created. Note that in theory, it is necessary to consider all possible σCSP (and thus, all possible sets of all possible sCSP); however, in practice, only required functions are fed in the **GI** algorithm, induced by the σCSP under consideration in the resolution process.

We first define functions on $SCSP$ with respect to selection functions to select the domains on which the functions apply. In order to extend operations on $SCSP$ to ΣCSP, we introduce a selection process which allows us to extract particular $sCSP$s of a given σCSP (see Figure 3).

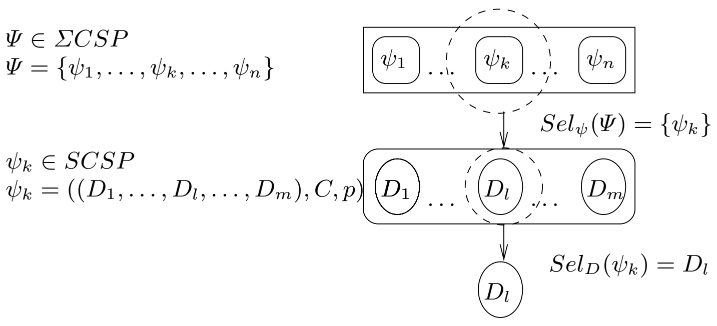
$\Psi \in \Sigma CSP$
$\Psi = \{\psi_1, \ldots, \psi_k, \ldots, \psi_n\}$
ψ_1 ... ψ_k ... ψ_n
$Sel_\psi(\Psi) = \{\psi_k\}$
$\psi_k \in SCSP$
$\psi_k = ((D_1, \ldots, D_l, \ldots, D_m), C, p)$
D_1 ... D_l ... D_m
$Sel_D(\psi_k) = D_l$
D_l

Fig. 3. Selection functions

Let us consider a domain selection function $Sel_D \colon SCSP \to 2^D$ and a *sCSP* selection function $Sel_\psi \colon \Sigma SCSP \to \Sigma SCSP$.

Domain Reduction. We may first define a domain reduction operator on a single sCSP as:

$$\begin{aligned} red^{Sel_D} &: SCSP \to SCSP \\ \psi &= (D, C, p) \mapsto (D', C, p') \end{aligned}$$

such that

1. $D = D_1 \times \cdots \times D_n, D' = D'_1 \times \cdots \times D'_n$ and $\forall i, 1 \le i \le n$
   - $D_i \notin Sel_D(\psi) \Rightarrow D'_i = D_i$
   - $D_i \in Sel_D(\psi) \Rightarrow D'_i \subseteq D_i$
2. $p' = p$ if $p \in \mathcal{LS}_{D'}$ otherwise p' is set to any sample chosen in $\varepsilon(D')$

Note that Condition 2. insures that the local search path associated to the sCSP stays in $\varepsilon(D')$. Note that we could keep $p' = (s_i)$ where s_i is the latest element of p which belongs to D' or we could keep a suitable sub-path of p. We have chosen to model here a restart from a randomly chosen sample after each reduction or split. The function red^{Sel_D} is extended to ΣCSP as:

$$\begin{aligned} red^{Sel_\psi, Sel_D} &: \Sigma CSP \to \Sigma CSP \\ &\Psi \mapsto (\Psi \setminus Sel_\psi(\Psi)) \bigcup_{\psi \in Sel_\psi(\Psi)} red^{Sel_D}(\psi) \end{aligned}$$

Split. We first define a split operator on a single sCSP as follows:

$$\begin{aligned} sp_k^{Sel_D} &: SCSP \to \Sigma CSP \\ &\psi \mapsto \Psi' \end{aligned}$$

with $\psi = (D_1 \times \ldots \times D_h \times \ldots \times D_n, C, p)$ where $\{D_h\} = Sel_D(\psi)$ and $\Psi' = \{(D_1 \times \ldots \times D_{h_1} \times \ldots \times D_n, C, p_1), \cdots, (D_1 \times \ldots \times D_{h_k} \times \ldots \times D_n, C, p_k)\}$ such that

1. $D_h = \bigcup_{i=1}^{k} D_{h_i}$
2. for all $i \in [1..k]$, $p_i = p$ if $p \in \mathcal{LS}_{(D_1 \times \cdots \times D_{h_i} \times \cdots \times D_n)}$ otherwise, p_i is set to any sample chosen in $\varepsilon(D_1 \times \ldots \times D_{h_i} \times \ldots \times D_n)$.

For the sake of readability we present a function that splits only one domain of the sCSP. But this can obviously be extended to split several domains at once. The last condition is needed to comply with the sCSP definition: it corresponds to the fact that, informally, the samples associated to any sCSP belong to the box induced by their domains. The function is extended to ΣCSP as follows:

$$\begin{aligned} sp_k^{Sel_\psi, Sel_D} &: \Sigma CSP \to \Sigma CSP \\ &\Psi \mapsto (\Psi \setminus Sel_\psi(\Psi)) \bigcup_{\psi \in Sel_\psi(\Psi)} sp_k^{Sel_D}(\psi) \end{aligned}$$

Local Search. As mentioned above, local search is viewed as the definition of a partial ordering $\sqsubseteq_{ls}$; this order is then used to define the ordering $\sqsubseteq$ on our hybrid structure ΣCSP. The components that remain to be defined are: 1) the strategy to compute a local search path p' of length $n+1$ from a local search path p of length n, and 2) the stop criterion which is commonly based on a fixed limited number of moves and, in our particular context of CSP resolution, the notion of computed solution.

First, we define a local search operator on $SCSP$ as a function $strat$: $SCSP \rightarrow 2^{\varepsilon(D)}$. This function specifies the choice strategy of a given local search heuristics for moving from a sample to one of its neighbors.

$$\begin{aligned} \lambda_N^{strat} \colon & SCSP \rightarrow SCSP \\ & \psi \mapsto \psi' \end{aligned}$$

where

- N is the maximum allowed number of moves
- $\psi = (C, D, p)$ and $\psi' = (C, D, p')$ with $p = (s_1, \cdots, s_n)$
  1. $p' = p$ if $p \in Sol_{\mathcal{LS}_D}$
  2. $p' = p$ if $n = N$
  3. $p' = (s_1, \cdots, s_n, s_{n+1})$ such that $s_{n+1} = strat(\psi)$ otherwise

Using this schema, we present here some examples of well known "move" heuristics. Consider a sCSP $\psi = (D, C, (s_1, \cdots, s_n))$. Each function consists in selecting one feasible neighbor (i.e., a sample of the neighborhood which is also in the current reduced search space D to comply with Definition 8) of a sample:

- **Random Walk:** the function $strat_{rw}$ randomly selects one sample of the neighborhood of the current sample
$$strat_{rw}(\psi) = s \ \text{ s.t. } \ s \in D \cap \mathcal{N}(s_n)$$
- **Descent:** the function $strat_d$ selects a neighbor improving the current sample with respect to the fitness function
$$strat_d(\psi) = s \ \text{ s.t. } \ s \in D \cap \mathcal{N}(s_n) \text{ and } s <_{eval} s_n$$
- **Strict Descent:** $strat_{sd}$ is similar to $strat_d$ but selects the best improving neighbor; $strat_{sd}(\psi) = s$ s.t.
$$s \in D \cap \mathcal{N}(s_n),\ s <_{eval} s_n, \text{ and } \forall s' \in D \cap \mathcal{N}(s_n), s \leq_{eval} s'$$
- **Tabu of length l:** selects the best neighbor not visited during the past l moves; $strat_{tabu_l}(\psi) = s$ s.t.
$$s \in \varepsilon(D) \cap \mathcal{N}(s_n) \text{ and } \forall j \in [n-l..n], s \neq s_j \text{ and } \forall s' \in D \cap \mathcal{N}(s_n), s \leq_{eval} s'$$

Note that, again, these functions satisfy the properties (inflationary and monotonic) required to be fed in the GI algorithm. Then, this function is extended to ΣCSP as:

$$\begin{aligned} \lambda_N^{Sel_\psi, strat} : & \ \Sigma CSP \rightarrow \Sigma CSP \\ & \Psi \mapsto (\Psi \setminus Sel_\psi(\Psi)) \bigcup_{\psi \in Sel_\psi(\Psi)} \lambda_N^{strat}(\psi) \end{aligned}$$

Combination. The "choose function" of the **GI** algorithm now totally manages the hybridization/combination strategy; different scheduling of functions lead to the same result (in term of least common fixpoint), but not with the same efficiency.

Note that in practice, we are not always interested in reaching the fixpoint of the **GI** algorithm. We can also be interested in solutions such as sCSPs which contain a solution for local search or a solution for constraint propagation. In this case, various runs of the **GI** algorithm with different strategies ("choose function") can lead to different solutions (e.g., in case of problems with several solutions, or several local minima).

Result of the GI Algorithm. We now compare the result of the GI algorithm with respect to Definition 5 for solution of a σCSP.

Since we are in chaotic iteration framework (concerning orderings and functions), given a σCSP Ψ and a set F of reduction functions (as defined above) the GI algorithm computes a common least fixpoint of the functions in F. Note that, this result is insured by the fact that our LS functions, which limit the size of search paths, induce a finite partial ordering in our computation structure. Clearly, this fixpoint $lfp(\Psi)$ abstracts all the solutions of $Sol(\Psi)$:

- $\bigcup_{(d,C,p)\in Sol(\Psi)} d \supseteq \bigcup_{(d,C,p)\in glfp(\Psi)} d$
- for all $(D, C, p) \in Sol(\Psi)$ s.t. $p = (s_1, \ldots, s_n) \in Sol_{\mathcal{LS}_D}(\Psi)$ there exists a $(d, C, p') \in glfp(\Psi)$ s.t. $s_n \in \varepsilon(d)$.

The first item states that all domain reduction and split functions used in GI preserve solutions. The second item ensures that all solutions computed by LS functions are in the fixpoint of the GI algorithm.

In practice, one can stop the GI algorithm before the fixpoint is reached. For example, one can compute the fixpoint of the LS functions; in this case, the search space may be reduced (and thus, the possible moves) by applying only some of the CP functions. This corresponds to the hybrid nature of the resolution process and the tradeoff between a complete and incomplete exploration of the space.

5 CP+GA

In this section, we describe the hybridization of a propagation based solver and genetic algorithms. We use the same approach as the one for local search. Thus, we try to keep the same progress, notations, and structure for this section.

5.1 Populations

Genetic algorithms aim at generating new populations using genetic operators, selection [6], (e.g. proportional selection [17], roulette-wheel selection [15], tournament selection, linear ranking [7], ...), recombination (e.g., elitist recombination [33], multiparent recombination like [10]), and mutation.

A new population is called an offspring and formally it is a mapping $\mathcal{O} : 2^D \rightarrow 2^D$. We define the set of possible genetic descendants, i.e., the set of sequences of populations as follows:

$$\mathcal{GA} = \bigcup_{k>0} \{p = (g_1, \cdots, g_k) \mid \forall j \in [1..k],\ g_j \in 2^D \text{ and } \forall j \in [2..k], g_j \in \mathcal{O}(g_{j-1})\}$$

where g_1 represents the initial population and k the length of the process. Note that, in practice, the size of th epopulation is fixed.

From a practical point of view, genetic algorithms are stopped by a criterion which is usually a fixed maximum number of iterations. Therefore, from a GA point of view, a result is either a population g which contains solutions or a genetic process of a maximum given size. Based on a fitness function (as presented in Section 2.3), we introduce the following order on sequences of populations of $\mathcal{GA}$:

Definition 9 (Order on sequences of populations). *Consider a fitness function eval together with its associate order. The order $\sqsubseteq_{ga}$ on $\mathcal{GA}$ is defined by:*

$$(g_1, \ldots, g_n) \sqsubseteq_{ga} (g'_1, \ldots, g'_m) \ \ \textit{iff} \ \ g'_m \leq_{eval} g_n$$

We have now to define the computation structure on which reduction functions will be applied and which includes the new component corresponding to the introduction of GA.

5.2 Computation Structure

In order to handle the different data structures associated to each technique of the hybrid resolution, we complete CSPs with *genetic factors.* Such a factor corresponds indeed to a GA process, and optimization will be done using them.

The resolution will be achieved according to the generic algorithm recalled in Section 2.3. We have here to define the computation structure devoted to this hybridization CP+GA.

Definition 10 (CSP with genetic factor). *A CSP with genetic factor (gcsp) for optimization is defined by a sequence (D, C, p, f) where*

- $D = D_1 \times ... \times D_n$
- $\forall c \in C, c \subseteq D_1 \times \ldots \times D_n$
- $p \in \mathcal{GA}$
- f*: objective function.*

GCSP denotes the set of gcsp, and $\Sigma GCSP$ denotes the set 2^{GCSP}

Note that, in the definition, the genetic algorithm process p should be included in the search space defined by D. Recall that the objective function f is taken into account in the *eval* function (see Section 2.3), and thus is also taken into account in the $\leq_{eval}$ and $\sqsubseteq_{ga}$ orderings (see above), and consequently in the ordered structure $(GCSP, \sqsubseteq)$ that we define below.

Definition 11 (Order on $GCSP$). *Given two gcsps $\psi = (D, C, p, f)$ and $\psi' = (D', C, p', f)$, $\psi \sqsubseteq \psi'$ iff*

- *$D' \subseteq D$*
- *or ($D' = D$ and $p \sqsubseteq_{ga} p'$).*

This relation is extended on 2^{GCSP}: $\{\phi_1; ...; \phi_k\} \sqsubseteq \{\psi_1; ...; \psi_l\}$, iff

$$\forall \phi_i, (\exists \psi_j,\ \phi_i \sqsubseteq \psi_j \text{ and } \nexists \psi_j,\ \psi_j \sqsubset \phi_i)$$

where $i \in [1..k], j \in [1..l]$.

$\Sigma GCSP$ (i.e., the set 2^{GCSP}) constitutes the key set of our computation structure. We use here σCSP to denote an element of $\Sigma GCSP$. The least element $\perp$ is $\{(D, C, p, f)\}$, i.e., the initial σCSP to be solved.

5.3 Solution

From the CP point of view, a solution of an gcsp $\psi = (D, C, p, f)$ is a tuple which satisfies all the constraints. From the GA point of view, the notion of solution is related to the evaluation function: a solution is defined as an element s of a population g of the sequence p such that s is the minimum (or maximum) of the objective function with respect to all such s' appearing in p. Given an gcsp $\psi = (D, C, p, f)$, these two points of view induce two sets of solutions:

- Feasible solutions: $Sol_{CP}(\psi) = \{d \in D \mid \forall c \in C,\ d \in c\}$
- Optimum solutions (minimization): $Sol_{\mathcal{GA}}(\psi) = \{s \mid p = (g_1, \ldots, g_m)$ and $\forall i \in [1..m], \forall s' \in g_i,\ s \leq_{eval} s'\}$.
- Optimum solutions (maximization): $Sol_{\mathcal{GA}}(\psi) = \{s \mid p = (g_1, \ldots, g_m)$ and $\forall i \in [1..m], \forall s' \in g_i,\ s' \leq_{eval} s\}$.

Based on this, we define the set of solutions in the hybrid model for a given gcsp ψ as:

$$Sol(\psi) = Sol_{CP}(\psi) \cap Sol_{\mathcal{GA}}(\psi)$$

Hence a solution of a given gcsp is a tuple d such that d satisfies the constraints and minimizes (respectively maximizes) the objective function. This notion of solution is generalized to the computation structure $\Sigma GCSP$.

Definition 12. *Given a σCSP $\Psi = \{\psi_1, \ldots, \psi_k\}$ according to*

- *a minimization problem: $Sol(\Psi) = Min(\{s_i\} \mid s_i \in sol(\psi_i))$*
- *a maximization problem: $Sol(\Psi) = Max(\{s_i\} \mid s_i \in sol(\psi_i))$*

5.4 A Function-Based Solving Process

At this step, we have to define the reduction functions on $\Sigma GCSP$. They describe the different components of the resolution process: constraint propagation by domain reduction and splitting, combined with genetic algorithms.

Given an element $\Psi = \{\psi_1, \cdots, \psi_n\}$ of $\Sigma GCSP$, we have to apply functions on Ψ which correspond to domain reduction, domain splitting, and genetic algorithm. These functions may operate on elements ψ_i of Ψ, and for each ψ_i on some of its components. We should note that since we consider here finite sets as initial gcsps, the structure is a finite partial ordering.

The following definitions introduce the fundamental properties of the different operators and their general purpose.

The definitions of a reduction function and of a split for the hybridization CP+GA are similar to the ones of CP+LS (Definitions 6 and 7) but this time they apply on $\Sigma GCSP$. The same remark is also valid concerning Definition 15 and Definition 8.

Definition 13 (Domain reduction function). *A* domain reduction function *red is a function:*

$$\begin{aligned} red\colon \Sigma GCSP &\rightarrow \Sigma GCSP \\ \{\psi_1, \cdots, \psi_n\} &\mapsto \{\psi'_1, \cdots, \psi'_n\} \end{aligned}$$

such that $\forall i \in [1 \cdots n]$*:*

- *either* $\psi_i = \psi'_i$,
- *or* $\psi_i = (D, C, p, f)$, $\psi'_i = (D', C, p', f)$ *and* $D \supseteq D'$ *and* $Sol(\psi_i) = Sol(\psi'_i)$.

This definition ensures that $\{\psi_1, \cdots, \psi_n\} \sqsubseteq red(\{\psi_1, \cdots, \psi_n\})$ and that the function is inflationary and monotonic on $(\Sigma GCSP, \sqsubseteq)$. From a constraint programming point of view, no solution of the initial $\sigma GCSP$ is lost by a domain reduction function. This is also the case for domain splitting as defined below.

Definition 14 (Domain splitting). *A domain splitting function sp on $\Sigma GCSP$ is a function such that for all $\Psi = \{\psi_1, \ldots, \psi_n\} \in \Sigma GCSP$:*

a. $sp(\Psi) = \{\psi'_1, \ldots, \psi'_m\}$ *with* $n \leq m$,
b. $\forall i \in [1..n]$,
   - *either* $\exists j \in [1..m]$ *such that* $\psi_i = \psi'_j$
   - *or there exist* $\psi'_{j_1}, \ldots, \psi'_{j_h}$, $j_1, \ldots, j_h \in [1..m]$ *such that*

$$Sol_D(\psi_i) = \bigcup_{k=1..h} Sol_D(\psi'_{j_k})$$

c. *and,* $\forall j \in [1..m]$,
   - *either* $\exists i \in [1..n]$ *such that* $\psi_i = \psi'_j$
   - *or* $\psi'_j = (D', C, p', f)$ *and there exists* $\psi_i = (D, C, p, f)$, $i \in [1..n]$ *such that* $D \supseteq D'$.

Conditions a. and b. ensure that some gcsps have been split without discarding solutions. Condition c. ensures that the search space does not grow (each new search space is included in one of the initial search space).

Finally we define genetic algorithm as a reduction function on $\Sigma GCSP$.

Definition 15 (Genetic algorithms). *A genetic algorithm function Γ_N is a function:*

$$\Gamma_N \colon \Sigma GCSP \to \Sigma GCSP \ , \\ \{\psi_1, \cdots, \psi_n\} \mapsto \{\psi'_1, \cdots, \psi'_n\}$$

where N is the maximum number of consecutive offsprings, and $\forall i \in [1..n]$

– *either $\psi_i = \psi'_i$*
– *or $\psi_i = (D, C, p, f)$ and $\psi'_i = (D, C, p', f)$ with $p = (g_1, \cdots, g_k)$ and $p' = (g_1, \cdots, g_k, g_{k+1})$ such that*
  $g_{k+1} \in \mathcal{O}(g_k) \cap D^m$ and $k + 1 \leq N$, where m is the size of the population .

N is the maximum length of a genetic algorithm, i.e., the number of offsprings allowed in a usual genetic search process. Note that $\psi_i = \psi'_i$ can happen when:

1. $n = N$: the maximum allowed number of operations has been reached,
2. Γ_N is the identity function on ψ_i, i.e., Γ_N does not try to improve the generation of the GCSP ψ_i. This might happen when no possible move can be performed (e.g., all individuals are equal and no mutation are allowed).

We now give some properties on some possible genetic algorithm functions.

Definition 16 (elitism). *A genetics algorithm is called elitist if at every step the current best individual survives, the best solution is never lost during the search. Formally consider a search path $p = (g_1, \ldots, g_k)$:*

$$\forall j \in [1..k-1], \textit{if there exists } s \in g_j \textit{ s.t. } \forall s' \in g_j, s \leq_{eval} s', \textit{ then } s \in g_{j+1}$$

Property 1 (Convergence). Suppose that the genetic algorithm is elitist. Suppose that for every population g there is a nonzero probability P that in the next generation the population is better:

$$\forall s \in g_k, \exists s' \in g_{k+1} \text{ s.t. } s' \leq_{eval} s$$

Then the fitness of the population at time t converges to the optimal value, for $t \to \infty$.

Thus, with the previous properties, GA optimizes the objective function, taking its values in a search space which is becoming locally consistent using CP. With successive constraint propagations and splits, the search space is progressively restricted to feasible solution, therefore GA finds the optimum.

5.5 $\sigma GCSP$s Resolution

As in the previous section, the **GI** algorithm is fed with the ordering on $\Sigma GCSP$; the least element $\perp$ is $\{(D, C, p, f)\}$, i.e., the initial $\sigma GCSP$ to be solved; and monotonic and inflationary functions: domain reduction, split, and genetic algorithms.

Similarly to Section 4, reduction functions can first be built over GCSP before being extended over $\Sigma GCSP$. In this case, a selection process is also needed in order to take into account each $\sigma GCSP$ that could be created during resolution. We do not detail here this process, since it is the same as for local search hybridization.

Result of the GI Algorithm. The result of the **GI** algorithm can be defined similarly as before. Given a $\sigma GCSP$ Ψ and a set F of reduction functions the GI algorithm computes a common least fixpoint of the functions in F. This fixpoint $glfp(\Psi)$ abstracts all the solutions of $Sol(\Psi)$:

- $\bigcup_{(d,C,p)\in Sol(\Psi)} d \supseteq \bigcup_{(d,C,p)\in glfp(\Psi)} d$
- for all $(D, C, p, f) \in Sol(\Psi)$ s.t. $p = (g_1, \ldots, g_n) \in Sol_{\mathcal{GA}_D}(\Psi)$ there exists a $(d, C, p', f) \in glfp(\Psi)$ s.t. $\exists i \in [1..n]$, $d \in g_i$.

The first item states that all domain reduction and split functions used in GI preserve solutions. The second item ensures that in all sequences of populations that are solution of the GA functions, there is a population containing a tuple which is in the fixpoint of the GI algorithm.

6 Experimentations

In this section, hybridizations CP+LS and CP+GA are tested using our constraint system (developed in C++). The purpose of this section is to highlight the benefit of our framework for hybridization and the benefit of hybrid resolution; note that our purpose is not to test a high performance implementation on large scale benchmarks. All tests are performed on a cluster with 22 processors running sequentially at 2.2 GHz with 1 Go of RAM each.

6.1 CP+LS for Constraint Satisfaction Problems

Problem Instances. We consider various classic CSP problems: *S+M=M* (Send + More = Money), *Magic Square*, *Langford numbers*, the *Zebra* puzzle, *Golomb ruler*, and the *Uzbekian problem*, issued from the CSPlib [13].

Experimental Process. Our basic functions are stored into three sets: a set of domain reduction functions dr, a set of split functions sp, and a set of local search functions ls. The *choose* function of the **GI** algorithm is defined as follows: we consider a tuple (α, β, γ) such that α, β, and γ represent respectively the percentage of reduction functions, split functions, and local search functions, that are applied; functions are fairly selected with respect to these ratios.

The reduction functions are defined as follows. A domain reduction corresponds either to a bound consistency operator or a global constraint filtering operator (e.g., $alldifferent$). A split function cuts the selected domain into two subdomains. A local search function is a basic LS move; LS functions are then instantiated by a tabu search strategy which selects the best neighbor not currently in a tabu list of length 10 (see Section 4.5).

In the following, we consider three types of strategies corresponding to selection function of sCSP (to select one $sCSP$ in a σCSP, i.e., Sel_ψ function as defined in Section 4.5), and selection function of domain (to select one domain in a CSP, i.e., Sel_D function as defined in Section 4.5) for domain reduction and split functions. Here, we do not formalize these functions, but we just describe their strategies:

- **random:** Sel_ψ selects any $sCSP$, and Sel_D selects any domain of the selected $sCSP$.
- **depth-first:** Sel_ψ selects the $sCSP$ containing the smallest domain, and Sel_D selects the smallest domain of the selected $sCSP$.
- **LS-forward checking:** forward checking consists in instantiating variables (split by enumeration) in a given order and to prevent future conflicts by reducing variables directly linked to the one freshly enumerated. Our LS-forward checking strategy is similar; ls functions will apply on the $sCSP$ that has just been split.
- **width-first:** Sel_ψ selects the $sCSP$ containing the largest domain, and Sel_D selects the largest domain of the selected $sCSP$.

Combining our reduction functions and the three above mentioned strategies, we obtain three packs (one for each strategy) of sets of functions (dr, sp, and ls).

Experimental Results. The evaluation and comparison criterion corresponds to the number of basic functions applied to reach a first solution. Such an application of function is either a step of local search, a split, or a domain reduction (reduction of one domain using one constraint). We focus here on small problems: thus, computation times represents less than one minute of CPU time (e.g., a solution for the Langford Number is found in one sec.).

Interaction between CP and LS. We study here the benefit of the hybridization CP+LS. Using various strategies we highlight the effect of different cooperations on solving efficiency.

We first focus on the problems Langford Number and S+M=M; the tests are performed by increasing the percentage α of propagation from 0 to 100%. To insure to reach a solution, we set the split ratio to $\beta = \alpha * 0.1$. For example, if $\alpha = 0.4$, we set β to 4% of split, and thus 56% of LS. These tests use the depth-first strategy above-mentioned.

Figure 4 shows that the best results for the Langford Number problem correspond to a range of propagation rate between 35% and 45%. As a matter of fact, when local search represents less than 10% of the search effort, reaching a solution means computing the fixpoint for constraint propagation (i.e., applying all

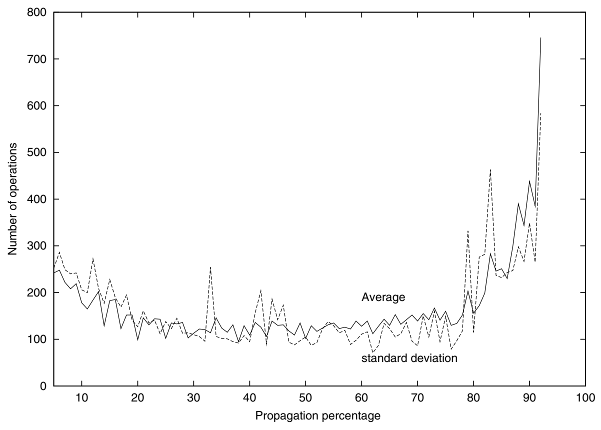

Fig. 4. Cost of a solution Langford Number (Depth-First)

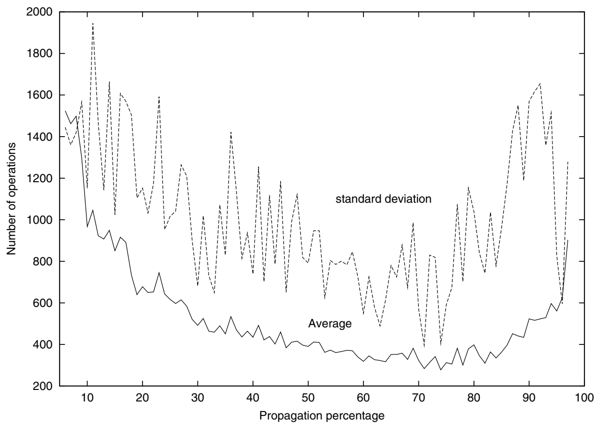

Fig. 5. Cost of a solution Send+More=Money (Depth-First)

propagation functions). Note that, for this problem, tabu search alone (Figure 4, left) provides better results than propagation with split (Figure 4, right).

Figure 5 shows the above-mentioned depth-first strategy to solve the S+M= M problem. The standard deviation is important: indeed, although sCSPs and domains are selected by the strategy, the choice of functions to apply is not fixed (random). However, the average performances are more regular, and the best range corresponds to 70%– 80% of propagation. Here, LS alone (Figure 5, right) appears less efficient than CP (Figure 5, left).

Thus, choosing the best settings for hybridization depends on the problem and on the strategies that are applied. Table 1 presents the best ranges using the LS-Forward-checking strategy for different problems.

Table 1. Best range of propagation rate (α) to compute a solution

Problem	S+M	LN42	Zebra	M. square	Golomb
Rate FC	70-80	15 - 25	60-70	30-45	30 - 40

These results point out that the incremental introduction of CP in LS (the same remark is valid for LS in CP) improves the general efficiency of resolution. These ratios of hybridization can thus be tuned to optimize performances.

Benefit of Hybridization with respect to LS and CP alone. In Table 2 we present a comparative study of the hybridization CP+LS, CP alone, and LS alone:

- the three strategies above-mentioned (random, depth-first, LS forward checking)
- CP+LS: the ratios (α, β, γ) are the best ratios selected in Table 1,
- CP (alone): the ratios are $(90\%, 10\%, 0)$,
- LS (alone): the ratios are $(0, 0, 100\%)$.

Table 2. Average number of operations to compute a first solution

Strategy	Method	S+M	LN42	M. square	Golomb
Random	LS	1638	383	3328	3442
	CP+LS	1408	113	892	909
	CP	3006	1680	1031	2170
D-First	LS	1535	401	3145	3265
	CP+LS	396	95	814	815
	CP	1515	746	936	1920
FC	LS	1635	393	3240	3585
	CP+LS	22	192	570	622
	CP	530	425	736	1126

Again, these results show that hybridization benefits from the interaction between the solving methods. Improvements occur on problems for which LS performs better than CP but also on problems for which CP is better than LS. Moreover, the improvement is strongly related to the problem structure (such as the density of solutions) and to the chosen strategy. Experiments with the Width-First strategy above-mentioned are not presented here but provided similar results.

6.2 CP+GA for Constraint Optimization Problems

Problem Instances. The BACP (Balanced Academic Curriculum Problem) is a problem class issued from the CSPlib [13]: it consists in organizing courses in order to balance the work load of students for each period of their curriculum. We

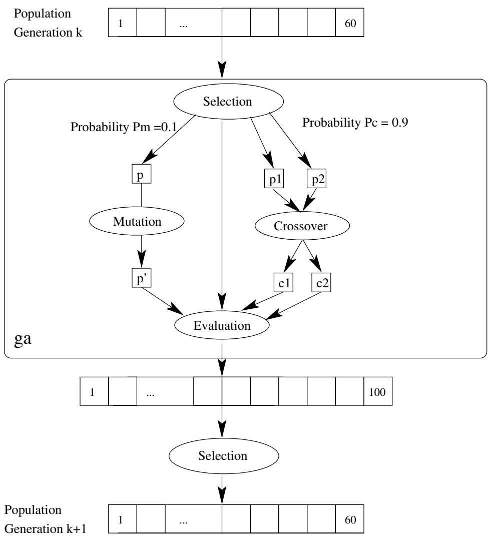

Fig. 6. *ga* functions

consider here various instantiations of the BACP: the bacp8, bacp10, and bacp12 problems issued from the CSPlib [13]; and finally data of this three curriculum are used to form a new problem (bacpall) for which some courses are shared by several curriculums.

Experimental Process. Similarly to CP+LS, our basic functions are organized into three sets: the set of domain reduction functions dr, the set of split functions sp, and the set of GA functions ga. In the following, the strategy is the depth-first strategy presented in the previous section.

Here, reduction functions correspond to arc consistency operators and reduction of global constraints (e.g., $period, load$) used to model the problems and to prune the search tree by detecting inconsistencies. The global constraint $period(i, \delta, \epsilon)$ computes the number of domains within the value i. If less than j occurrences of i are present in the m different domains, then the current CSP is locally inconsistent: $\delta \leq (\sum_{k=1}^{m} 1 \mid x_k = i) \leq \epsilon$. The global constraints $load(i, \beta, \gamma)$ counts the charge for a given period i of the current CSP: $\beta \leq (\sum_{k=1}^{m} c_k \mid x_k = j) \leq \gamma$.
sp are split operators which cut the selected domain into two subdomains.
ga are basic GA operators (see figure 6) which are instantiated by our genetic algorithm: from a population k, our genetic algorithm generates a new population k+1 of 60 individuals selected among 100 issued from k. When ga is called by the main algorithm, the following different cases may occur:

– the population $k+1$ has less than 100 individuals: an individual is selected randomly; then, either it is coupled with another parent to create 2 children

in the population $k+1$, either it is submitted to mutation, or it remains unchanged in the population $k+1$.
- the population $k+1$ has 100 individuals: the 60 best ones are selected according to the evaluation function which takes into account the objective function.

Experimental Results. For these experimentations, we integrated the GA module (i.e., *ga* functions) in our constraint based solving system for hybridization. We also added the notion of optimization to the single notion of solution.

In order to compare our results, we present the results of [8] using the linear programming solver lp_solve for the bacp8 and bacp10 problems (Table 3 shows the progress of the cost –evaluation– of the objective function w.r.t. the computation time). The results with our hybrid solver CP+GA are shown in Table 4. If lp_solve is able to find the optimal solution for the first problem, it is not the case for the second one.

Table 3. Results in seconds using lp_solve

Sol quality	bacp 8	Sol quality	bacp 10
24	137.08	33	9.11
23	218.23	32	25.38
21	218.43	30	25.65
20	712.84	29	1433.18
19	1441.98	27	1433.48
18	1453.73	26	1626.49
17 (optimum)	1459.73	24	1626.84

As mentioned above, we control the rates of each family of functions *dr*, *sp* and *ga* by defining the strategy (completing the depth-first strategy) as a tuple $(\%_{dr}, \%_{sp}, \%_{ga})$ of application rates. These values correspond indeed to a probability of application of a function from each family. In practice, we measure in Figure 7 the rate of effective applications, i.e., we only count the functions which are chosen according to the strategy and having a real impact on the resolution.

The most interesting in such an hybridization is the completeness of the association GA-CP, and the roles played by GA and CP in the search process (see Figures 7): GA optimizes the solutions in a search space progressively becoming locally consistent (and thus smaller and smaller) using constraints propagation and split. To evaluate the benefits of each of the methods we have measured:

- for CP: the number of effective reductions that are performed and the number of split,
- and for GA: the fact that the next generation is globally better than the previous one.

First of all, splits are limited to 1% of the total number of basic operations (reduction functions) because of the space complexity they generate.

Table 4. Results using GA+CP

Sol quality	bacp 8	bacp 10	bacp 12
24	0.47	4.71	2.34
23	0.54	4.67	2.40
22	0.61	3.68	2.48
21	0.61	4.36	2.76
20	0.69	4.63	3.20
19	0.83	4.95	4.25
18	1.20	5.13	35.20
17	15.05 (optimum)	5.60	
16		6.39	
15		8.53	
14		34.84 (optimum)	

Concerning the single problems (bacp8, bacp10, bacp12). At the beginning, CP represents 70% of the effort: constraint propagation narrows the search space. On the contrary, GA represents about 30%. During this period, not enough local consistency is enforced by constraint propagation, and GA only finds solutions (satisfying all constraints) with costs greater than 21. Then, at the beginning of the second half of the search process, CP and GA converge in terms of efficiency: most of the sub-GCSP have reach the local consistency and tests over constraints do not improve domain reduction. At the end, GA performs 70% of the search effort to find the optimal solution.

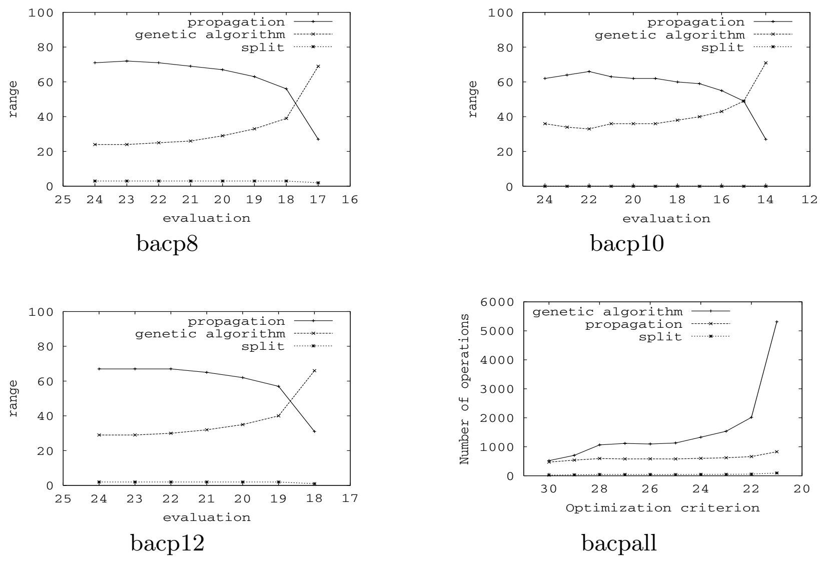

Fig. 7. Evolution of CP vs GA during the optimization process

Concerning strategies using GA and CP alone. In this implementation, CP is unable to find a feasible solution in 10 minutes cpu time. GA can find alone the optimal value but is 10 times slower than the hybrid resolution $GA + CP$. Therefore, we did not include these results in the tables.

In the figure for the all-period problem, CP and GA start searching with the same efficiency; but while CP seems to be stable, most of the operations are performed by the genetic process to improve the solution. This could be explained by the fact that, in this problem, constraints are too weak with respect to the number of variables and the size of the generated search space. But, in our hybrid resolution system, GA appears as a powerful method even if most of the constraint operators have not reached their fixpoints.

7 Perspectives and Conclusion

Most of hybrid approaches are ad-hoc algorithms based on a master-slave combination: they favor the development of systems whose efficiency is strongly related to a given class of CSPs. In this paper, we have used a more suitable general framework to model hybrid solving algorithms. We have shown that this work can serve as a basis for the integration of LS and CP methods, and the integration of GA and CP methods in order to highlight the connections between complete and incomplete techniques and their main properties.

We have shown how to integrate two techniques in the framework of chaotic iterations: CP+LS and CP+GA. However, this requires defining a new computation structure and orders on these structures. Moreover, the reduction functions have to be adapted to the new structures. Thus, integrating a new technique requires modifying the current structures and functions. We plan to modify our framework in order to be able to add a new technique without modifying the structure, simply by extending the existing structure. We also plan to modify function definition so that they can be defined independently. Some new types of functions operating the cooperation between the techniques. The first use of this new framework will be an hybrid solver CP+LS+GA.

A future extension will consists in providing "tools" to help designing and testing finer strategies in the GI algorithm in our particularly suitable uniform framework. To this end, we plan to extend works of [16] where strategies are built using some composition operators in the GI algorithm. Moreover, this will also open possibilities of concurrent and parallel applications of reduction functions inside the model.

At last, we plan to extend our prototype implementation (Section 6) into a complete generic implementation of our framework.

Acknowledgements. We would like to thank the program committee who gave us the opportunity of writing this paper and more especcially Christophe Ringeissen for his interesting remarks and useful comments.

References

1. E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley and Sons, 1997.
2. A. Aggoun and N. Beldiceanu. Overview of the chip compiler system. In K. Furukawa, editor, *Logic Programming, Proceedings of the Eigth International Conference*, pages 775–789. MIT Press, 1991.
3. K. Apt. The rough guide to constraint propagation. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 1–23, Springer, 1999. (Invited lecture).
4. K. Apt. From chaotic iteration to constraint propagation. In *Proceedings of ICALP '97*, volume 1256 of Lecture Notes in Computer Science, pages 36–55. Springer, 1997.
5. K. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
6. T. Bäck, J. M. de Graaf, J. N. Kok, and W. A. Kosters. Theory of genetic algorithms. In *Current Trends in Theoretical Computer Science*, pages 546–578. 2001.
7. J. E. Baker. *Adaptive Selection Methods for Genetic Algorithms.* ICGA, pages 101-111, 1985.
8. C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problems. In *Proceedings of 6th Workshop of the ERCIM WG on Constraints. CoRR cs.PL/0110007*, 2001.
9. R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.
10. A. E. Eiben, P-E. Raué and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation*, volume 866 of Lecture Notes in Computer Science, pages 78-87, Springer, 1994.
11. F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In *Handbook of Metaheuristics*, volume 57 of International Series in Operations Research and Management Science, Kluwer, 2002.
12. T. Fruewirth and S. Abdennadher. *Essentials of Constraint Programming.* Springer, 2003.
13. I. Gent, T. Walsh, and B. Selman. http://www.csplib.org, funded by the UK Network of Constraints.
14. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.
15. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., 1989.
16. L. Granvilliers and E. Monfroy. Implementing Constraint Propagation by Composition of Reductions. *Proceedings of International Conference on Logic Programming*, volume 2916 of Lecture Notes in Computer Science, pages 300-314. Springer, 2003.
17. J. H. Holland. *Adaptation in Natural and Artificial Systems.* 1975.
18. H. Hoos and T. Stülze. *Stochastic local search : foundations and applications.* Morgan Kaufmann, Elsevier, 2004.
19. ILOG. *ILOG Solver 5.0 User's Manual and Reference Manual*, 2000.
20. K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems.* Phd thesis, University of Michigan, 1975.
21. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21-45, 2002.
22. F. Laburthe. CHOCO: implementing a cp kernel. In *CP'00 Post Conference Workshop on Techniques for Implementing Constraint Programming Systems - TRICS*, 2000.

23. T. Lambert and E. Monfroy and F. Saubion. Solving Strategies using a Hybridization Model for Local Search and Constraint Propagation. In *Proceedings of ACM SAC'2005*, pages 398-403, ACM Press 2005.
24. A. Mackworth. *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley, 1987.
25. K. Mariott and P. Stuckey. *Programming with Constraints, An introduction.* MIT Press, 1998.
26. Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd, extended ed.).* Springer, New York, 1996.
27. R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
28. E. Monfroy, F. Saubion and T. Lambert. On Hybridization of Local Search and Constraint Propagation. In *Proceedings of ICLP'2004*, pages 299-313, volume 3132 of Lecture Notes in Computer Science, Springer, 2004.
29. P. Pardalos and M. Resende. *Handbook of Applied Optimization.* Oxford University Press, 2002.
30. G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, volume 1118 in Lecture Notes in Computer Science, pages 353–366. Springer, 1996.
31. S. Prestwich. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In *Principle and Practice of Constraint Programming - CP 2000*, volume 1894 in Lecture Notes in Computer Science, pages 337–352. Springer, 2000.
32. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98*, volume 1520 of Lecture Notes in Computer Science, pages 417–431. Springer, 1998.
33. D. Thierens and D. E. Goldberg. Convergence Models of Genetic Algorithm Selection Schemes. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation*, volume 866 of Lecture Notes in Computer Science, pages 119-129,Springer, 1994.
34. E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.