On Hybridization of Local Search and Constraint Propagation

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

[1] LINA, Université de Nantes, France
eric.monfroy@lina.univ-nantes.fr
[2] LERIA, Université d'Angers, France
{frederic.saubion,tony.lambert}@univ-angers.fr

Abstract. Hybridization of local search and constraint programming techniques for solving Constraint Satisfaction Problems is generally restricted to some kind of master-slave combinations for specific classes of problems. In this paper we propose a theoretical model based on K.R. Apt's chaotic iterations for hybridization of local search and constraint propagation. Hybrid resolution can be achieved as the computation of a fixpoint of some specific reduction functions. Our framework opens up new and finer possibilities for hybridization/combination strategies. We also present some combinations of techniques such as tabu search, node and bound consistencies. Some experimental results show the interest of our model to design such hybridization.

1 Introduction

Constraint Satisfaction Problems (CSP) [15] provide a general framework for the modeling of many practical applications (planning, scheduling, time tabling,...). A CSP is usually defined by a set of variables associated to domains of possible values and by a set of constraints. We only consider here CSP over finite domains. Constraints can be understood as relations over some variables and therefore, solving a CSP consists in finding tuples that belong to each constraint (an assignment of values to the variables that satisfies these constraints). To this end, many resolution algorithms have been proposed and we may distinguish at least two classes of general methods: 1) *complete methods* aim at exploring the whole search space in order to find all the solutions or to detect that the CSP is not consistent. Among these methods, we find methods based on constraint propagation, one of the most common techniques from constraint programming [4] (CP) for solving CSP; and 2) *incomplete methods* (such as Local Search [1] (LS)) 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.

A common idea is to build more efficient and robust algorithms by combining several resolution paradigms in order to take advantage of their respective assets (e.g., [5] presents an overview of possible uses of LS in CP). The benefit of the hybridization LS+CP is well-known and does not have to be proven (see e.g., [8, 13, 12, 14]). Most of the previous works are either algorithmic approaches which define a kind of master-slave combination (e.g., LS to guide the search in CP, or CP to reduce interesting area of the search space explored in LS), or ad-hoc realizations of systems for specific classes of problems.

In this paper, we are concerned with a model for hybridization in which local search [1] and constraint propagation [4] are broken up into their component parts. These basic operators can then be managed at the same level by a single mechanism. In this framework, properties concerning solvers (e.g., termination, solutions) can be easily expressed and established. This framework also opens up new and finer possibilities of combination strategies.

Our model is based on K.R. Apt's chaotic iterations [2] which define a mathematical framework for iteration of a finite set of functions over "abstract" domains with partial ordering. This framework is well-suited for solving CSPs with constraint propagation: domains are instantiated with variable domains (possible values of variables), and functions with domain reduction functions to remove inconsistent (w.r.t. constraints) values of domain variables (reduction functions abstract the notion of constraint in this mechanism).

Moreover, to get a complete solver (a solver which is always able to determine whether a CSP has some solutions), constraint propagation is generally associated with a splitting mechanism (a technique such as enumeration or bisection) to cut the search space into some smaller search spaces from which one can hope to perform more propagation. Propagation and splitting are interleaved until the solutions are reached.

In our model, Local Search [1] is based on 3 notions: samples which are particular points or sets of points of the search space, neighborhood that defines which samples can be reached from a specific sample, and a fitness function that defines the "quality" of a sample. Then, LS explores the search space by moving from sample to sample guided by the fitness function in order to reach a local optimum.

For our purpose, we introduce in our model the notion of sCSP (sampled CSP) which is an extension of CSP with a path (list) of samples (generally points of the search space). We also integrate the splitting as some reduction functions. This way, the "abstract" domains of chaotic iteration are instantiated with union of sCSPs. Usual domain reduction functions (used for constraint propagation) are extended to fit this new domain. Some new functions (the LS functions) are also introduced to jump from samples to samples: these functions have the sufficient properties required to be used in the chaotic iteration algorithm.

Thus, in the chaotic iteration framework, constraint propagation functions, local search moves, and splitting functions are considered at the same level and all apply to unions of sCSPs. Since interleaving and order of applications of these functions are totally free, this framework enables one to design finer strategies for hybridization than the usual master-slave combinations. Moreover, termination of the realized solvers is straight forward, i.e., fixpoint of the reduction functions.

In order to illustrate our framework, we realized some hybrid solvers using some well-known techniques and strategies such as tabu search (for LS), node and arc consistencies (for propagation), and bisection functions (for splitting).

We obtained some experimental results that show the interest of combination strategies compared to a single use of these methods. Moreover, the combination strategies can easily be designed and specified, and their properties can be proven in our model.

This paper is organized as follows. Section 2 describes constraint propagation and local search in terms of basic operators. We present our framework for hybridization in Section 3. Some experiments are given in Section 4 before concluding in Section 5.

2 Constraint Satisfaction Problems

In this section we recall basic notions related to Constraint Satisfaction Problems (CSP) [15] together with their resolution principles. Complete resolution is presented using the theoretical model developed by K.R. Apt [2, 3]. We then briefly describe the main lines of a local search process.

A CSP is a tuple (X, D, C) where $X = \{x_1, \cdots, x_n\}$ is a set of variables taking their values in their respective domains $D = \{D_1, \cdots, D_n\}$. A constraint $c \in C$ is a relation $c \subseteq D_1 \times \cdots \times D_n$ [1]. In order to simplify notations, D will also denote the Cartesian product of D_i 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 $\forall c \in C, d \in c$.

2.1 Solving a CSP with Complete Resolution Techniques

As mentioned in introduction, complete resolution methods are mainly based on a systematic exploration of the search space, which corresponds obviously to the set of possible tuples. To avoid the combinatorial grow up of this exploration, these methods use particular heuristics to prune the search space. The most popular of these techniques (i.e., constraint propagation) is based on local consistency properties. A local consistency (e.g., [9, 11]) is a property of the constraints which allows the search mechanisms to delete values from variables domains which cannot lead to solutions. We may mention node consistency and arc consistency [10] as famous examples of local consistencies. Complete search algorithms use constraint propagation techniques and splitting. Constraint propagation consists in examining a subset (usually a single constraint) C' of C, deleting some inconsistent values (from a local consistency point of view) of the domains of variables appearing in C' and to propagate this domain reduction to domains of variables appearing in $C \setminus C'$. When no more propagation is possible and the solutions are not reached, the CSP is split into sub-CSPs (generally, the domain of a variable is split into two sub-domains, leading to two sub-CSPs) on which propagation is applied again, and so on until the solutions are reached.

K.R. Apt proposed in [2, 3] a general theoretical framework for modeling such reduction operators. In this context, domain reduction corresponds to the

[1] Note that for sake of simplicity, 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 [2, 3] can be used to denote sequences of variables.

computation of a fixpoint of a set of functions over a partially ordered set. These functions, called *reduction functions*, abstract the notion of constraint.

The computation of the least common fixpoint of a set of functions F is achieved by the following algorithm:

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 G is the current set of functions still to be applied ($G \subseteq F$), d is a partially ordered set (the domains in case of CSP), and 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 $g \in update(G, g, d)$

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 [2]).

Note that in the following we consider only finite partial orderings. Constraint propagation is now achieved by instantiating the **GI** algorithm:

- the $\sqsubseteq$ ordering is instantiated by $\supseteq$, the usual set inclusion,
- $d := \perp$ corresponds to $d := D_1 \times \ldots \times D_n$, the Cartesian product of the domains of the variables from the CSP,
- F is a set of monotonic and inflationary functions (called *domain reduction functions*) which abstract the constraints to reduce domains of variables. For example, one of the domain reduction functions to reduce Boolean variables using a $and(X, Y, Z)$[2] constraint is defined by: if the domain of Z is $\{1\}$, then the domains of X and Y must be reduced to $\{1\}$.

The result is the smallest box (i.e., Cartesian product of domains) w.r.t. the given domain reduction functions that contains the solutions of the CSP.

At this point, in order to get the solutions of the CSP, one has to explore the reduced domains by enumeration or splitting techniques (and then, again, propagation, and so on). This usually implies an algorithmic process interleaving splitting and propagation phases. However, in the following, we will integrate splitting as a reduction function inside the **GI** algorithm, and we will extend the notion of CSP to sampled CSP on which an other type of reduction functions will be applied to mimic basic operations of local search algorithms.

[2] $and(X, Y, Z)$ represents the Boolean relation $X \wedge Y = Z$.

2.2 Solving CSP with Local Search

Given an optimization problem (which can be minimizing the number of violated constraints and thus trying to find a solution of the CSP), local search techniques [1] aim at exploring the search space, moving from a configuration to one of its neighbors. These moves are guided by a fitness function which evaluates the benefit of such a move in order to reach a local optimum. We will generalize the definition of local search in next sections.

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 $\mathcal{N} : D \to 2^D$. This neighborhood function defines indeed the possible moves from a configuration (a tuple) to one of its neighbors 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 $t \notin c$ (t being a tuple from D).

In this case, the problem to solve is indeed a minimization problem. Given a configuration $d \in D$, two basic strategies can be identified in order to continue the exploration of D:

- intensification: choose $d' \in \mathcal{N}(d)$ such that $eval(d') < eval(d)$.
- diversification: choose any other neighbor d'.

The intensification process only performs improving moves while diversification strategy allows the process to move to a worst neighbor w.r.t. the *eval* function. Any local search algorithm is 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.

3 A Uniform Computational Framework

From these different CSP resolution approaches, our aim is to integrate the various involved computation processes in a uniform description framework. The purpose of this section is to instantiate the general computation scheme presented in Section 2.1.

Our idea is to extend the set of usual functions used in the generic iteration algorithm with splitting operators and local search strategies. Then, these search methods can be viewed as the computation of a fixpoint of a set of functions on an ordered set. Therefore, the first step of our work consists in defining the main structure.

3.1 Sampling the Search Space

As we have seen, domain reduction and splitting operate on domains of values while 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). *Given a CSP (X, D, C), we define a sample function $\varepsilon : D \to 2^D$. By extension, $\varepsilon(D)$ denotes the set $\{\varepsilon(d) | \, d \in D\}$.*

Generally, $\varepsilon(d)$ is restricted to d and $\varepsilon(D) = D$, but it can also be a scatter of tuples around d, an approximation or a box covering d (e.g., for continuous domains). Moreover, it is reasonnable 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 $\varepsilon(D)$ and the set of visited samples for each local search path composed by a sequence of moves. Given a neighborhood function $\mathcal{N}: \varepsilon(D) \to 2^{\varepsilon(D)}$, we define the set of possible local search paths as $\mathcal{LS}_D =$

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

since 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 according to a stop criterion which can be a fixed maximum number of iterations or, in our context of CSP resolution, the fact that a solution is reached. For this concern, according to Section 2.2, we consider an evaluation function $eval: \varepsilon(D) \to \mathbb{N}$ such that $eval(s)$ represents the number of constraints unsatisfied by s and $eval(s)$ 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.

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

$$(s'_1, \ldots, s'_m) \sqsubseteq_{ls} (s_1, \ldots, s_n) \textit{ iff } eval(s_n) = 0 \textit{ or } n \geq m.$$

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, they all correspond to possible results of a local search of size 2, and they are equivalent w.r.t. to Definition 2.

3.2 Computation Structure

We now instantiate the abstract framework of K.R. Apt described in Section 2.1.

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

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

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 $sCSP$ and we define now an ordering relation on the structure $(SCSP, \sqsubseteq)$.

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

$$\psi \sqsubseteq \psi' \;\; \textit{iff} \;\; D' \subseteq D \; \textit{or} \; (D' = D \; \textit{and} \, p \sqsubseteq_{ls} p').$$

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

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

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

Note that this ordering on sCSPs could be extended by also considering an order on constraints; this would enable constraint simplifications.

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

3.3 Notion of Solution

Our framework is dedicated to CSP resolution and therefore we have to define precisely the notion of solution w.r.t. the previous computation structure. We should note that this notion is clear from each side of the resolution (i.e., complete and incomplete methods). From the complete resolution point of view, a solution of a CSP is a tuple which satisfies all the constraints. From the LS point of view, the notion of solution is related to the evaluation function *eval* which defines a solution as an element s of $\varepsilon(D)$ such that $eval(s) = 0$.

Given a sCSP $\psi = (D, C, p)$, these two points of view induce two sets of solutions $Sol_D(\psi) = \{d \in D | \forall c \in C,\, d \in c\}$ and $Sol_{\mathcal{LS}_D}(\psi) = \{(s_1, \cdots, s_n) \in \mathcal{LS}_D \,|\, eval(s_n) = 0\}$.

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

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

This notion is extended to any σCSP Ψ as $Sol(\Psi) = \bigcup_{\psi \in \Psi} Sol(\psi)$.

3.4 Reduction Functions Definitions and Properties

We have now to define the notion of function on ΣCSP. Given an element $\Psi = \{\psi_1, \cdots, \psi_n\}$ of ΣCSP, we have to apply functions on Ψ which correspond to domain reduction, domain splitting, and local search. These functions may operate on various elements of Ψ, and for each ψ_i on some of its components. We should note that since we consider here finite initial CSPs, our structure is a finite partial ordering.

Definition 6 (Domain reduction function). *A domain reduction function is a function red on ΣCSP s.t. for all $\Psi = \{\psi_1, \ldots, \psi_n\} \in \Sigma CSP$, $red(\Psi) = \{\psi'_1, \ldots, \psi'_n\}$ and $\forall i \in [1 \cdots n]$:*

– *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 condition insures that $\Psi \sqsubseteq red(\Psi)$ and that the function is inflationary and monotonic on $(\Sigma CSP, \sqsubseteq)$. It allows one to reduce several domains of several *sCSPs* of a σCSP at the same time. From a constraint programming point of view, 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 is a function sp on* ΣCSP *such that for all* $\Psi = \{\psi_1, \ldots, \psi_n\} \in \Sigma CSP$*:*

a. $sp(\Psi) = \{\psi'_1, \ldots, \psi'_m\}$ *with* $n \le m$,
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 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: none of the domain of the sCSPs composing Ψ' is not 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* $k + 1 \le N$.

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 can try to improve the sample of one or several sCSPs at once. 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. $n = N$: 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 [6]).

3.5 Solving σCSPs

The complete solving of a σCSP $\{(D_1, \ldots, D_n, C, p)\}$ now consists in instantiating the **GI** algorithm:

Computation Structure.

– the $\sqsubseteq$ ordering is instantiated by the ordering given in Definition 4,
– $d := \perp$ corresponds to $d := \{(D_1, \ldots, D_n, C, p)\}$, the Cartesian product of the variables domains and of the sample from the sCSP,
– F is a set of given monotonic and inflationary functions as defined in Section 3.4: domain reduction functions (extensions of usual domain reduction functions for CSPs), domain splitting functions (usual splitting mechanisms integrated as reduction functions), and local search functions (e.g., functions for descent, tabu search, ...).

Functions. We propose here 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. Therefore we have to consider functions driven by a selection operator. Given a selection function $select: A \rightarrow 2^B$ let us consider a function $f^{select}: A \rightarrow C$ such that $f^{select}(x) = g(y), y \in select(x)$ where $g: B \rightarrow C$. 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.

This corresponds to the fact that all possible functions are needed for each σCSP that can result from the application of some functions on the initial σCSP to model the different possible executions of the resolution process[3].

We first define functions on $SCSP$ w.r.t. selection functions to select the domains on which the functions apply. Similarly and 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 1).

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

[3] This is necessary in theory, however, in practice, only required functions are fed in the **GI** algorithm.

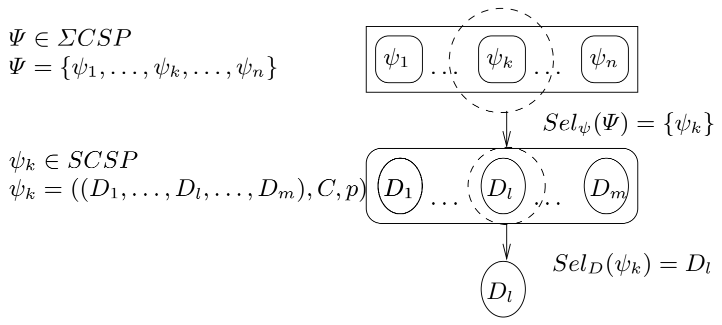
$\Psi \in \Sigma CSP$
$\Psi = \{\psi_1, \ldots, \psi_k, \ldots, \psi_n\}$
ψ_1
ψ_k
ψ_n
$Sel_\psi(\Psi) = \{\psi_k\}$
$\psi_k \in SCSP$
$\psi_k = ((D_1, \ldots, D_l, \ldots, D_m), C, p)$
D_1
D_l
D_m
$Sel_D(\psi_k) = D_l$
D_l

Fig. 1. Selection functions

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. $D = \{D_1, \cdots, D_n\}, D' = \{D'_1, \cdots, D'_n\}$ and $\forall 1 \leq i \leq n$
   - $D_i \in D \setminus Sel_D(\psi) \Rightarrow D'_i = D_i$
   - $D_i \in Sel_D(\psi) \Rightarrow D'_i \subseteq D_i$
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 $\varepsilon(D')$ [4]. This function is extended to ΣCSP as:

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

Splitting. We may first define a splitting operator on a single sCSP as:

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

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

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

[4] Note that we could keep $p' = (s_i)$ where s_i is the latest element of p which belongs to D', for instance, or a suitable sub-path of p. We have chosen to model here a restart from a randomly chosen sample after each reduction or splitting.

For the sake of readability we consider here the split of a single domain in the initial sCSP but it can obviously be extended to any selection function. The last condition is needed to satisfy the sCSP definition and corresponds to the fact that, informally, the samples associated to any sCSP belong to the box induced by their domains.

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

Local Search. As mentioned above, local search is viewed as the definition of a partial ordering $\sqsubseteq_{ls}$ which is used for the definition of the ordering $\sqsubseteq$ on ΣCSP. The remaining components to be defined are now: 1) the strategy to obtain 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 usually based on a fixed limited number of LS moves and, in this particular context of CSP resolution, the notion of reached solution. We first define a local search operator on $SCSP$ based on a function $strat\colon SCSP \rightarrow 2^{\varepsilon(D)}$ which defines the choice strategy of a given local search heuristics in order to move 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 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. $p' = (s_1, \cdots, s_n, s_{n+1})$ s.t. $s_{n+1} \in strat(\psi) \cap D$ otherwise

We provide here some examples of well known "move" heuristics.

- **Descent**: selects better neighbors
  $strat_d((D, C, (s_1, \cdots, s_n))) = \{s_{n+1} \in \varepsilon(D) \mid s_{n+1} <_{eval} s_n \wedge s_{n+1} \in \mathcal{N}(s_n)\}$
- **Strict Descent**: selects best improving neighbors
  $strat_{sd}((D, C, (s_1, \cdots, s_n))) = \{s_{n+1} \in \varepsilon(D) \mid s_{n+1} <_{eval} s_n \wedge s_{n+1} \in \mathcal{N}(s_n) \wedge \forall s' \in \mathcal{N}(s_n), s_{n+1} \leq_{eval} s'\}$
- **Random Walk**: selects all the neighbors
  $strat_{rw}((D, C, (s_1, \cdots, s_n))) = \{s_{n+1} \in \varepsilon(D) \mid s_{n+1} \in \mathcal{N}(s_n)\}$
- **Tabu of length l**: selects best neighbor not visited during the past l moves
  $strat_{tabu_l}((D, C, (s_1, \cdots, s_n))) = \{s_{n+1} \in \varepsilon(D) \mid \forall n - l \leq j \leq n, s_{n+1} \neq s_j \wedge s_{n+1} \in \mathcal{N}(s_n) \wedge \forall s' \in \mathcal{N}(s_n), s_{n+1} \leq_{eval} s'\}$

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

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

Combination. The combination strategy is now totally managed by the "choose function" of the **GI** algorithm: 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, but 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, different 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 w.r.t. the Definition 5 of solution of a σCSP.

Since we are in Apt's 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. Clearly, 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 $(D,C,p) \in Sol(\Psi)$ 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 represents the fact that all domain reduction and splitting 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, only some applications of the CP functions can reduce the search space (and thus, the possible moves). This corresponds to the hybrid nature of the resolution process and the tradeoff between a complete and incomplete exploration of the space.

4 Experimentation

In this section, we present a prototype, developed in C++, which allows us to test hybridization on different CSP examples.

4.1 Functions and Strategies

We choose $\varepsilon(D_1,\cdots,D_n)$ as the Cartesian product $D_1\times\cdots\times D_n$ ($\varepsilon(D) = D$). We consider the two selection functions $min(\Psi) = \{\psi\} \subseteq \Psi$ such that $\nexists \psi', \psi \sqsubseteq \psi'$ and $max(D) = \{D_i\}$ such that $\forall j \neq i, |D_i| \geq |D_j|$ (if there are several possible candidates choose the one with smallest index). A first set of domain reduction functions DR contains node and bound consistency operators (see [10]). The set SP contains the splitting operators $split_2^{min,max}$ and consist in cutting in two the largest domain of the minimal element of a σCSP. At last the set LS contains functions which corresponds to a tabu method: $\lambda_N^{min,strat_{tabu_l}}$.

The purpose of this section is not to test a high performance algorithm on large scale benchmarks but to prove the interest of our framework over various small problems.

According to the generic algorithm **GI**, note that one has to define a choose strategy at each iteration and to update the set of functions. Here we describe three different choose functions:

- $DR \cup SP$**:** in this case we consider the initial set of functions $F = DR \cup SP$ then the choose function is defined as: choose any $g \in G \cap DR$ or any $g \in G$ if $G \cap DR = \emptyset$ (G being the set of functions still to be applied in the `GI` algorithm). This simulates a complete backtracking algorithm.
- LS**:** here we consider $F = LS$. Note that in this case there is only one LS function for tabu search. We have also experimented a descent with random walk algorithm. In that case, reduction functions corresponding to descent and random walk strategies are applicable on each sCSP and one has to choose them alternatively with a certain probability.
- $DR \cup SP \cup LS$**:** we consider $F = DR \cup P \cup LS$. The choose function is: while $G \cap DR \neq \emptyset$ choose $g \in G \cap DR$; then choose any $g \in SP$; then while $G \cap LS \neq \emptyset$ choose $g \in G \cap LS$. In other terms, this strategy is: perform all possible domain reductions, then make a split, then a full local search; and iterate on this process.

Selected Problems. We propose various problems : **S+M=M**, a well-known crypto-arithmetic problem which consists in solving the equation $SEND + MORE = MONEY$ by assigning a different digit to each letter; **Marathon**, a problem of arrival in a race knowing particular information about the competitors (6 variables and 29 constraints); **Scheduling** problem (15 variables and 29 constraints); classical **Magic Square**, **Langford number** and **N-queens** benchmarks.

4.2 Experimental Results

The values given in the following table correspond to the truncated average of 50 independent runs. Concerning $DR \cup SP$, we count then the number of iterations of splitting operators which corresponds to the number of nodes in a classical backtracking algorithm. Concerning LS, we count the number of applied functions, which corresponds to the number of moves performed by the local search. We also mention the success rate. The computation time (t) is given in seconds. For $DR \cup SP \cup LS$, we limit the number of moves for each local search to $N = 100$, while for LS alone, a maximum of $500,000$ moves is allowed. Tabu list length l is set to 10.

We compare first the number of nodes with $DR \cup SP$ and $DR \cup SP \cup LS$ to get the first solution. Table 1 shows that $DR \cup SP \cup LS$ finds a solution with a smaller number of nodes compared to $DR \cup SP$ alone. In the combination, the relative efficiency of the LS part depends on the problem. For problems with one solution such as *S+M=M*, the benefit is less significant than for the other benchmarks (in particular *N-queens*).

Table 1. First solution ($DR \cup SP$ vs. $DR \cup SP \cup LS$ vs. LS)

Problem	$DR \cup SP$ (nodes | t)	$DR \cup SP \cup LS$ (nodes | moves | t)	LS (s. rate | moves | t)
S+M=M	(10 | 0.0)	(5.4 | 375.2 | 0.0)	(44% | 59294 | 5.6)
Marathon	(8 | 0.0)	(1 | 11.5 | 0.0)	(100% | 18 | 0.0)
Scheduling	(31 | 0.0)	(1 | 2.5 | 0.0)	(8% | 4726 | 59.1)
24-queens	(3329800 | 1631.2)	(1.2 | 62.8 | 0.6)	(100% | 201.7 | 2.1)
Magic-Square-4	(39 | 0.0)	(13.5 | 743 | 0.2)	(100% | 3423 | 2.0)
Langford 2 4	(4 | 0.0)	(1.6 | 95 | 0.0))	(100 % | 601 | 0.0)

Table 1 also shows the efficiency of the local consistency which guides the local search (see comparisons $DR \cup SP \cup LS$ vs. LS), in particular on *S+M=M* and *Scheduling.* One should remark that the success rate of LS is also an important parameter which is really improved by the combination since in that case, hybrid solving always succeeds in finding one solution.

Computation time is, of course, strongly related to the implementation of the different operators. We may remark that this computation is improved by using $DR \cup SP \cup LS$ instead of LS alone. The comparison between $DR \cup SP$ and $DR \cup SP \cup LS$ is not really significant except on *N-queens* where the hybridization provides an important saving of time.

Finally, we have calculated the computing cost needed to get several solutions with $DR \cup SP \cup LS$ on a *Magic-Square-3* problem which has 8 solutions. The mechanisms progress together to get a set of distinct solutions by computing fewer nodes (about 25 % less for each solution) than a classical backtracking algorithm simulated by $DR \cup SP$. To get several distinct solutions is a really good asset compared to LS alone (which was not designed for computing different solutions) and could be very interesting for number of problems.

At last, similar experiments have been performed with a hill climbing algorithm and provided similar conclusions.

5 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 presented a global model which is more suitable for integrating different strategies of combination and for proving some properties of these combinations. We have shown that this work can serve as basis for the integration of LS and CP methods in order to highlight the connections between complete and incomplete techniques and their main properties.

In the future, we plan to extend our framework in order to handle optimization problems. From a LS point of view, this will change the *strat* functions used to create the reduction functions. From a CP point of view, algorithms such as Branch and Bound requires adding new constraints during resolution: this could be done using a new type of reduction function in our model.

An other future extension is to provide some "tools" to help designing finer strategies in the GI algorithm. To this end, we plan to extend works of [7] where strategies are built using some composition operators in the GI algorithm. Moreover, this will also open possibilities of concurrent and parallel application of reduction functions inside our model.

At last, we plan to complete our prototype implementation (Section 4) into a "fully" generic implementation of our framework in order to design and test new and finer efficient strategies of hybridization.

Ackowledgement. The authors are grateful to Willem Jan van Hoeve for the interesting remarks he made on a preliminary version of this paper and to the anonymous referees for their useful comments.

References

1. E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley and Sons, 1997.
2. K. Apt. From chaotic iteration to constraint propagation. In *24th International Colloquium on Automata, Languages and Programming (ICALP '97*, number 1256 in LNCS, pages 36–55. Springer, 1997. invited lecture.
3. K. Apt. The rough guide to constraint propagation. In *Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 1–23, Springer, 1999. (Invited lecture).
4. K. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
5. F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In *Handbook of Metaheuristics.* Kluwer, 2002.
6. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.
7. L. Granvilliers and E. Monfroy. Implementing Constraint Propagation by Composition of Reductions. In C. Palamidessi, editor, *Proceedings of International Conference on Logic Programming*, LNCS 2916, pages 300–314. Springer, 2003.
8. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.
9. A. Mackworth. *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley, 1987.
10. K. Mariott and P. Stuckey. *Programming with Constraints, An introduction.* MIT Press, 1998.
11. R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
12. G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Proc. of the Second International Conference on Principles and Practice of Constraint Programming*, number 1118 in LNCS, pages 353–366. Springer, 1996.
13. 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*, number 1894 in LNCS, pages 337–352. Springer, 2000.
14. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98*, number 1520 in LNCS, pages 417–431. Springer, 1998.
15. E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.