Search Trees for CSPs

Julien Arsouze, Gérard Ferrand, Arnaud Lallouet

Universitè d'Orléans — LIFO

Réalisation OADymPPaC D.1.1.2 première partie.

1 Constraint Satisfaction Problems

This paper only focus on the reduction and search paradigm for CSP solving. We refer to Tsang [10] for other methods to solve CSPs.

Let A be a set and I be a finite set of indices, we denote by $(a_i)_{i\in I}$ where $a_i \in A$ a family of elements of A indexed by I. Now let $A = (A_i)_{i\in I}$ be a family of sets. We denote $\Pi A = \Pi_{i\in I} A_i = \{(a_i)_{i\in I} \mid \forall i \in I, a_i \in A_i\}$ the *indexed* cartesian product of the family A. The elements of ΠA are called *tuples*. If all A_i are ordered sets, ΠA is ordered by the product ordering of all A_i. For $J \subseteq I$ and a family a on I, we denote by $a|_J = (a_i)_{i\in J}$ the restriction of a to the set of indices J. This is an element of the indexed cartesian product $\Pi(A|_J)$. We denote by $\mathcal{P}(A)$ the powerset of A.

Definition 1 (CSP). *A* CSP $\mathcal{C}$ *is given by a 5-uple* (V, C, var, D, ext) *composed of a syntactic part:*

- *a finite set V of variable symbols (variables in short)*
- *a finite set C of constraint symbols (constraints in short).*
- *a function $var : C \longrightarrow \mathcal{P}(V)$ that associates to a constraint symbol the set of the variables of the constraint.*

and a semantic part:

- *a family $D = (D_x)_{x\in V}$ of non-empty domains.*
- *a family of sets of tuples $ext \in \Pi_{c\in C}\mathcal{P}(\Pi_{i\in var(c)}(D_i))$ called the* extension *of the CSP. The family $ext = (ext_c)_{c\in C}$ associates a set of tuples on the appropriate variables domains to every constraint symbol in C. We call $ext_c \subseteq \Pi(D|_{var(c)})$ the* extension *of c (in other words, ext_c is the definition — or solution set — of the individual constraint c).*

CSP solving is mainly a matter of reduction of the constraints extensions [9, 1]. However, it is useful to consider that every variable x of V belongs to an unary constraint d_x, also called its domain by a slight abuse of language. While the (mathematical) domain D_x does not change, the "domain" d_x will be reduced as the reduction proceeds. Variable domains are most of the time the only reduced constraints. According to Van Emden [11], this scheme is called the *value constraint* subscheme and is of real importance in practice. This is in particular the basis of the CHIP system [12], the Gnu-Prolog system [6] and many others. To simplify and because non-unary constraint reduction is not needed in this paper, the value constraint subscheme is used here. Therefore, the set $\mathcal{SS} = \Pi_{x\in V}\mathcal{P}(D_x)$ is called the *search space* of the CSP (in the full constraint reduction scheme, it would have been $\Pi_{c\in C}\mathcal{P}(\Pi(D|_{var(c)}))$. $\mathcal{SS}$ is a lattice and is naturally ordered by $\subseteq$, the product of subset orderings for each variable domain. A pointwise intersection is defined on $\mathcal{SS}$. Let $I = \mathbb{N}$ or $I = [0..k]$ be a set of indices and $I^+ = I - \{0\}$. The limit of any decreasing sequence $e = (e_i)_{i\in I}$ where $e_i \in \mathcal{SS}$ exists and is denoted by $\lim(e) = \bigcap_{i\in I} e_i$. An element $d \in \mathcal{SS}$ is called a *search point* and includes a set of values for each variable name. The greatest element of $\mathcal{SS}$ exists and is $\max \mathcal{SS} = (D_x)_{x\in V}$.

The following definition particularizes a class of search points in which each component is a singleton.

Definition 2 (Singletonic Search Point). *A search point d is* singletonic *if* $\forall x \in V$, *d_x is a singleton.*

Let us call a family $u = (u_x)_{x \in V} \in \Pi(D|_V)$ a *global tuple*. The set of global tuples is denoted by $\mathcal{GT}$. Solutions of a CSP $\mathcal{C}$ are naturally global tuples which satisfy all constraints. They are defined from the notion of partial solution:

Definition 3 (Partial Solution). *For $S \subseteq C$, let $Sol(S) = \{u \in \mathcal{GT} \mid \forall c \in S, u|_{var(c)} \in ext_c\}$.*

In particular, the global solutions of the CSP are given by $Sol(C)$ (the classical definition of partial solution [10] involves a subset of the variables instead of a subset of the constraints).

Finding solutions of a CSP involves a combination of two techniques: search (also called labeling in general) and reduction which consists in pruning inconsistent values from variables domains in order to reduce the search effort. This is done by running a network algorithm up to reach a property called *consistency*. This is classically called filtering of the domains, but the word reduction is prefered here since a different notion of filter will be defined.

Example 4 (Local consistencies). In binary CSPs, a constraint $c(x, y)$ is arc-consistent if whenever a value v_x is present in the domain of x, there exists a value v_y in the domain of y such that $c(v_x, v_y)$ holds and conversely for y and x. A CSP is arc-consistent if all its constraints are arc-consistent. Many other consistencies have been considered in the literature. Among them can be cited hyperarc-consistency, path-consistency, k-consistency and directional arc-consistency, all of them being mentioned with exact references in [10].

These local consistencies can be modeled by using the common greatest fixpoint of a set of suitable operators. This feature has been introduced by Montanari and Rossi [9] and was extended by Apt [1]. These operators can be defined from rules which reduce the domain of one constraint according to the extension of some other constraints $U \subseteq C$:

Definition 5 (Local Consistency Operator). *Let $U \subseteq V$ and $x \in U$. A* local consistency operator *is a function $r : \Pi_{z \in U} \mathcal{P}(D_z) \longrightarrow \mathcal{P}(D_x)$ which is monotonic.*

Each local consistency operator r can be extended to a *reduction operator* $reduc_r$ on $\mathcal{SS}$.

Definition 6 (Reduction Operator). *Let $r : \Pi_{z \in U} \mathcal{P}(D_z) \longrightarrow \mathcal{P}(D_x)$ be a local consistency operator and $d \in \mathcal{SS}$. The reduction operator $reduc_r : \mathcal{SS} \longrightarrow \mathcal{SS}$ associated to r is defined by:*

- $\forall v \in V,\ v \neq x \Longrightarrow reduc_r(d)_v = d_v$
- $reduc_r(d)_x = r(d|_U) \cap d_x$

These operators are monotonic, contracting and idempotent. They are applied until a fixpoint is reached in order to obtain the expected consistency. Operators may affect the domain of more than one variable at a time. In this case, they are defined by (the closure of) more elementary operators which affect only one variable.

Example 7 (Gnu-Prolog). The finite domain CLP system *Gnu-Prolog* [6] implements such rules for constraints solving. These domain reduction rules follow a special syntax called "X in r" where X represents a variable domain to be reduced and r an expression involving domains of other variables. The semantics of this rule is $d_X \longleftarrow d_X \cap eval(r)$. An n-ary constraint is compiled into "X in r" rules, each of them defining an operator. For example, the constraint $X < Y$ is compiled to "X in $0..min(Y) - 1$" and "Y in $max(X) + 1..infinity$". A suitable composition of expressions allows to model various consistencies such as (hyper)arc-consistency or partial arc-consistency.

Definition 8 (Correct Reduction Operator). *A reduction operator* $reduc_r$ *is* correct *if forall* $d \in \mathcal{SS}$ *and for all global solution s of a CSP, the following implication is verified:*

$$\forall x \in V,\ s_x \in d_x \Rightarrow s_x \in reduc_r(d)_x.$$

In this paper, all reduction operators are supposed correct.

Let $\mathcal{F}$ be a set of reduction operators. The following definition allows to caracterize the points of the search space on which all the operators of $\mathcal{F}$ have no effect.

Definition 9 (Stable). *A search point d is* stable *by* $\mathcal{F}$ *if* $\forall f \in \mathcal{F}, f(d) = d$.

Let s be a solution of a CSP then, since reduction operators are correct, it is stable. In this paper, we suppose also the converse. So, a singletonic search point is a solution of the CSP if and only if it is stable by $\mathcal{F}$.

Let $C_{\mathcal{F}}(d) = max\{d' \in SS \mid d' \subseteq d$ and d' stable by $\mathcal{F}\}$ denote the closure of d by the operators of $\mathcal{F}$. $C_{\mathcal{F}}$ is called the *closure operator* of $\mathcal{F}$.

Lemma 10. *For all sets* $\mathcal{F}$ *of such reduction operators,* $C_{\mathcal{F}}$ *is monotonic, contracting and idempotent.*

Proof. Straightforward.

Here are some properties of closure operators.

Lemma 11.
Let $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *be two sets of reduction operators and d a point of the search space then:*

1. $C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d) = C_{\mathcal{F}_1 \cup \{C_{\mathcal{F}_2}\}}(d) = C_{\{C_{\mathcal{F}_1}, C_{\mathcal{F}_2}\}}(d)$ *("compositionnality" lemma).*
2. $\mathcal{F}_1 \subseteq \mathcal{F}_2 \Longrightarrow C_{\mathcal{F}_2}(d) \subseteq C_{\mathcal{F}_1}(d)$.
3. $C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d) \subseteq C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d))$.
4. *If* $C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d))$ *is stable by* $\mathcal{F}_2$*, then* $C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d)) = C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d)$.

Proof.

1. Let $A = C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d)$ et $B = C_{\mathcal{F}_1 \cup \{C_{\mathcal{F}_2}\}}(d)$. First, let show that $A \subseteq B$. By definition, A is stable by all the functions of $\mathcal{F}_1 \cup \mathcal{F}_2$. It is sufficient to have $\forall f \in \mathcal{F}_1 \cup \{C_{\mathcal{F}_2}\},\ A \subseteq f(A)$. This is true for $f \in \mathcal{F}_1$. So let $f = C_{\mathcal{F}_2}$, in order to have $A \subseteq C_{\mathcal{F}_2}(A)$, it is sufficient to have $\forall f' \in \mathcal{F}_2, A \subseteq f'(A)$, which is always true.
   For the reverse inclusion, it is necessary to show that $\forall f \in \mathcal{F}_1 \cup \mathcal{F}_2, B \subseteq f(B)$. B is stable by $\mathcal{F}_1$ and by $C_{\mathcal{F}_2}$, so $B = C_{\mathcal{F}_2}(B)$. This means that B is stable by all the functions of $\mathcal{F}_2$.
   In order to have the second equality, it is only necessary to apply the first one with $\mathcal{F}_2 = \{C_{\mathcal{F}_1}\}$.
2. Let $A = C_{\mathcal{F}_1}(d)$ and $B = C_{\mathcal{F}_2}(d)$. In order to have $B \subseteq A$, it is only necessary to have $\forall f \in \mathcal{F}_1, f(B) \subseteq B$. But $\forall f \in \mathcal{F}_2, f(B) = B$ and $\mathcal{F}_1 \subseteq \mathcal{F}_2$. So $B \subseteq A$.
3. $\mathcal{F}_2 \subseteq \mathcal{F}_1 \cup \mathcal{F}_2$, so, by lemma 11-2, $C_{\mathcal{F}_1 \cup \mathcal{F}_2}(s) \subseteq C_{\mathcal{F}_2}(d)$. $C_{\mathcal{F}_1}$ is monotonic (by lemma 10), so $C_{\mathcal{F}_1}(C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d)) \subseteq C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d))$. But $C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d)$ is stable by $\mathcal{F}_1$, so it is equal to $C_{\mathcal{F}_1}(C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d))$.
4. $C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d))$ stable by $\mathcal{F}_2$ by hypothesis and of course by $\mathcal{F}_1$ so $C_{\mathcal{F}_1}(C_{\mathcal{F}_2}(d)) \subseteq C_{\mathcal{F}_1 \cup \mathcal{F}_2}(d)$.

In order to compute the closure of a search point d by a set $\mathcal{F}$ of reduction operators, or at least an approximation if a transfinite iteration is required to reach the exact limit, two kinds of iteration can be considered. The first one is the *standard iteration*. It consists in combining all operators of $\mathcal{F}$ into one and iterating this new operator until stability.

Definition 12 (Standard Iteration). *Let $F_{\mathcal{F}}$ be the operator defined by: $\forall d \in \mathcal{SS},\ F_{\mathcal{F}}(d) = \bigcap_{f \in \mathcal{F}} f(d)$.*

The standard iteration *of $\mathcal{F}$ from $d \in \mathcal{SS}$ is the sequence $a = (a_i)_{i \in \mathbb{N}}$ of element of $\mathcal{SS}$ where $a_0 = d$ and $\forall i \in \mathbb{N}^+,\ a_i = F_{\mathcal{F}}(a_{i-1})$.*

To compute the next state, all operators have to be applied and, since this this involves a large work, *chaotic iteration* is a prefered method. It consists in applying each time only one operator $f \in \mathcal{F}$ instead of $F_{\mathcal{F}}$.

Definition 13 (Chaotic Iteration). *A* chaotic iteration *of $\mathcal{F}$ starting from $s \in \mathcal{SS}$ is a sequence $b = (b_i)_{i \in I}$ of elements of $\mathcal{SS}$ where $b_0 = s$ and $\forall i \in I^+,\ \exists f \in \mathcal{F}$ such that $b_i = f(b_{i-1})$.*

So, at each step, only one reduction operator is applied and each operator's application takes advantage of the reductions performed by the preceding ones. There is a particular class of chaotic iterations whose limit is stable.

Definition 14 (Complete Chaotic Iteration). *A chaotic iteration is* complete *if its limit is stable.*

The interest of this class of chaotic iterations is that they reach the closure $C_{\mathcal{F}}(d)$. Chaotic iterations have been introduced by Chazan and Miranker [4] in the context of linear algebra. It has been used as a general technique to fasten fixpoint computation by Cousot and Cousot [5] and many other works since then, before being identified as a foundation for constraint propagation by Apt [1] and Van Emden [11]. Note that Montanari and Rossi [9] also proved the main result but for more specific operators and the idea of computing a greatest fixpoint is already present in Benhamou [3]. Chaotic iterations may differ depending on the strategy used to schedule the operators. In this paper, we consider only complete chaotic iterations because only convenient strategies (fairness of the reduction operators scheduling strategy) are used. Thanks to this hypothesis, each chaotic iteration of a set $\mathcal{F}$ starting from the same search point has the same limit which is th closure by $\mathcal{F}$ of the starting search point.

Theorem 15 (Confluence, Apt [1]). *Let $\mathcal{F}$ be a set of reduction operators and $b1$, $b2$ be two complete chaotic iterations of $\mathcal{F}$ starting from $d \in \mathcal{SS}$ then:*

$$lim(b1) = lim(b2) = C_{\mathcal{F}}(d).$$

Proof. Straightforward.

2 γ-chaotic iterations

One of the drawbacks of chaotic iterations is that they rely on an external mechanism to detect stability (and thus termination of the program which computes them). This section introduces the γ-chaotic iterations which keep the properties of chaotic iterations and bring a solution to detect (operationnaly) stability.

γ-chaotic iterations are also a way to take advantage of certain properties of the reduction operators in order to accelerate the convergence. For example, operators used to enforce arc-consistency are usually idempotent, so once applied, it is unnecessary to consider them immediately after. This is why existing algorithms and systems implement a data structure to schedule the reduction operators: it may be a queue as in AC-3 [13], a priority queue as in Gnu-Prolog [6] or a more complex event parsing structure as in the *Choco* system [8] in which cheap propagations are done immediately while costly ones are further delayed. These structures can be modeled by a subset of $\mathcal{F}$ (the set of all reduction operators). For each point d of the search space $\mathcal{SS}$ (which correspond also to a computation state), several subset γ of $\mathcal{F}$ can be defined in which an operator is selected by the strategy. They can depend on the choosen strategy, on the computation state, on the way reduction operators are applied. Finally, the whole history of the computation can be taken into account. Because of the number of possible parameters, this subset which is composed of choosable reduction operators has a very general definition.

Definition 16 (Choice Set). *Let d be a point of the search space. A subset γ of $\mathcal{F}$ such that $\forall f \in \mathcal{F} - \gamma,\ f(d) = d$ is called a* choice set *for d.*

So, any operator on the outside of γ does not reduce the search point. Let $d \in \mathcal{SS}$ such that $\gamma = \emptyset$. So, $\forall f \in \mathcal{F},\ f(d) = d$. In this case, no more reduction can be done and so the consistency defined by $\mathcal{F}$ is verified. This property can be used operationnally to detect stability of a search point. Morever, at each search point d, there exists a minimal set $\gamma_d = \{f \in \mathcal{F} \mid f(d) \neq d\}$. But conversely, a computed γ may include non-reducing operators.

Example 17 (AC3 propagation queue). The implementation of the AC-3 algorithm requires a propagation queue which contains the operators to be applied. At the begining, the queue contains all the operators. At each step, an operator is chosen. If a component d_x is reduced, then all the operators which depend of the variable x are added in the queue on condition that they are not already in it. Since operators are indempotent, the chosen operator does not reduce any variable domain at the following step. Moreover, only the operators which depend on the variable whose domain has been reduced are added in the queue. So, all the operators which are not in the queue do not reduce any variable domain but, on the other side, there may be operators in the queue whose application reduces no variables domains. The empty queue means that the fixpoint has been optained.

A γ-chaotic iteration is a kind of chaotic iteration for which at each step, the reduction function is chosen in the choice set associated to the search point of the last state:

Definition 18 (γ-chaotic iteration). *A* γ- chaotic iteration *of $\mathcal{F}$ starting from $d \in \mathcal{SS}$ is a sequence $c = (c_i, \gamma_i)_{i \in I}$ with c_i element of $\mathcal{SS}$ and γ_i a choice set for to the current state c_i such that $c_0 = d$ and $\forall i \in I^+$, $\exists f \in \gamma_{i-1}, c_i = f(c_{i-1})$.*

At each γ-chaotic iteration $c = (c_i, \gamma_i)_{i \in I}$ can be associated the chaotic iteration $(c_i)_{i \in I}$. So, γ-chaotic iteration is a particular case of chaotic iterations. Reciprocally, chaotic iteration is a particular case of γ-chaotic iterations for which, at each step, the associated choice set γ_i is equal to the set of reduction operators $\mathcal{F}$.

The case where no reduction is done at a step of a γ-chaotic iteration is modeled by the fact that for some $i \in I,\ \gamma_i = \emptyset$. It is a sufficient condition to stop a chaotic iteration because no more reduction can be done.

Definition 19 (Complete γ-chaotic iteration). *A γ-chaotic iteration is* complete *if its associated chaotic iteration is complete.*

So, if there exists $i \in I$ such that $\gamma_i = \emptyset$, then the γ-chaotic iteration is *complete*. This condition is only sufficient because the limit can be sometimes reached at a step $j \leq i$. In this case, all the reduction operators of γ_j do not reduce the variables domains and the following steps of reduction are unnecessary.

Note that the choice set is dependent of the search point but not only. For example, the way how a search point is reached in a γ-chaotic iteration can be also a parameter for the computation of the choice set of this search point, and more generally, the whole history of the computation.

A modelization similar in intention to the set γ has been presented in [2] by mean of an *update* function which updates the set of "active " operators. Let $G \subseteq \mathcal{F}$ be the set of active reduction functions and d be the current state. After the choice of a function $g \in \mathcal{G}$, the algorithm replaces G by $G - \{g\} \cup update(G, g, d)$. A similar function also appear in [7] which takes as additional input the domain just reduced by the application of g. These functions allow to model AC-3 and other similar algorithms more directly than our set γ because of their operational nature (they all compute an approximation of γ for $d \in \mathcal{SS}$). However, the set γ is more declarative and better suited for our concerns. It allows to model any operational strategy, even more complex update strategies like taking into account the whole computation history instead of only the preceding state, or the removing of functions from the data structure. In [1], *semi-chaotic iterations* are presented. An iteration is semi-chaotic if all the operators which do not appear an infinite number of time from the rank i of the iteration, do not reduce any domain at any step $j \geq i$. This case is a particular one of γ-chaotic iterations because since an operator do not reduce any domain, it may not be in the defined choice set and so can not be applied.

3 Search Trees

3.1 Computation Scheme

Let $\mathcal{C} = (V, C, var, D, ext)$ be a CSP. As it was explained in the beginning of the CSP part, finding solutions of $\mathcal{C}$ involves the combination of two different techniques. The first one which is described in the previous part is reduction. The second one is search. The efficiency of constraint programming resides in the good balance by which this two efforts are combined. Usually constraint programming systems propagate until reaching the fixpoint to ensure consistency after each search step.

The closure computed by ($\gamma-$)chaotic iterations has formally no relationship with the set of solutions for a given CSP. Only the assumption that all reduction operators are correct with respect to the set of solutions allows to ensure that their closure is a superset of the solutions. In order to really find solutions, we need to focus on domains subparts of this closure; to ensure completeness, we need to explore them in a systematic manner.

In order to modelize computations which combine the two techniques, computation schemes are introduced. They are composed by computation steps which contain a subset of C (it represents the constraints taken into account at this step) and the associated set of reduction operator R and an other set of constraints S with its associated set of reduction operators L used to focus on particular parts of the domains. In a obvious way, it is a particular case of adding new constraints.

Definition 20 (Computation Step).
A computation step is a 4-uple $(Cons, R, S, L)$ such that:

- *$Cons$ is a subset of $\mathcal{C}$.*
- *S is a set of constraints.*

– *R and L are reduction operators sets associated respectivaly to $Cons$ and S.*

So $Cons$ is the set of constraint which must be solved with local consistency methods and S is the set of constraints which defined the subpart of the search space in which are added because of the search technique.

Reduction and search steps are modelized by transitions between two computation steps. So, there is two kinds of transition.

Definition 21 (Transition).
A transition is a couple $((Cons_1, R_1, S_1, L_1), (Cons_2, R_2, S_2, L_2))$ of computation steps. It is a r-transition if $Cons_2 = Cons_1 \cup \{c\}$ with $c \in C$ (and so $R_2 = R_1 \cup R_c$ with R_c the set of reduction operators associated to the constraint c). It is a s-transition if $S_2 = S_1 \cup \{c'\}$ (and so $L_2 = L_1 \cup R_{c'}$).

So, a r-transition consists in taking into account a new constraint of C and this corresponds to a reduction step. A s-transition modelizes a search step.

Let D the family of domains given by the CSP $\mathcal{C}$. At each computation step $(Cons, R, S, L)$ is associated the closure $C_{R \cup L}(D)$. This family is called the domain of the computation step. In order to define a computation scheme, some particular computation steps have to be defined.

Definition 22 (Terminal Computation Step).
Let $(Cons, R, S, L)$ be a computation step and $d = (d_x)_{x \in V}$ its associated domain.

– *If there exists $x \in V$ such that $d_x = \emptyset$ then the computation step is a* failure terminal *one.*
– *If $\forall x \in V$, d_x is a singleton then the computation step is a* solution terminal *one.*

Since a failure terminal computation step has an associated domain which has an empty component d_x, there is no solution for the CSP $(V, Cons, var, D, ext)$ in the search space restricted by the set of constraints S because no assignement is possible for the variable x. On the opposite, a success terminal solution computation step has an associated domain such that each component is a singleton. So, this singletonic search point is a solution of the CSP $(V, Cons, var, D, ext)$.

Now, here is the definition of a computation scheme. Its goal is to describe a computation composed of search and reduction steps.

Definition 23 (Computation Scheme).
A computation scheme is a finite sequence of computation step such that:

– *every couple of computation step is a transition*
– *if there exists a terminal computation step then it is the last one.*

It is important to note that a computation scheme can be finished by a computation step which is not terminal.

A computation scheme is associated to a CSP (V, C, var, D, ext) if its first computation step is $(\emptyset, \emptyset, \emptyset, \emptyset)$ and has the associated domain D and the last computation step must verify one of the two following conditions: $Cons = C$ or it is a failure terminal one.

In order to obtain the associated domain of a computation step $(Cons, R, S, L)$, that is $C_{R \cup L}(D)$, a stable γ-chaotic iteration of $R \cup L$ starting from the associated domain of the previous computation step is done. The concatenation of all the γ-chaotic iterations is a computation.

Definition 24 (Computation). *The computation associated to a scheme is the finite sequence of reduction operators obtained by the concatenation of all the γ-chaotic iterations done at each step of the scheme.*

Let (C, R, S, L) be the last step of a scheme. Then, the associated computation is a chaotic iteration of $R \cup L$.

Finaly, the correctness of a computation is established. If D is the domain associated to the first step then all the solutions (of the set of constraints $Cons$ of the last step) which are in D restricted by the set of constraint S of the last step are in the domain associated to the last step.

Theorem 25 (Correctness). *Let $(C_i, R_i, S_i, L_i)_{i \in I}$ with $I = [1..n]$ a computation scheme such that D is the domain associated to the first step. Every solution of C_n included in $C_{L_n}(D)$ is in $C_{R_n \cup L_n}(D)$.*

Proof. Straightforward (the reduction operators are correct).

3.2 Search Trees

in order to ensure completeness, we need to combine computation schemes in a systematic manner. For solving this problem, search trees are introduced.

Definition 26 (Search Tree). *A search tree is composed of nodes which are computation steps. There is two kinds of node:*

– *a reduction node is a node $n = (C, R, S, L)$ which has only one child $n' = (C \cup \{c\}, R \cup R_c, S, L)$ with c a constraint, R_c the set of reduction operators associated to the constraint c and such that (n, n') is a c-transition.*

$$\begin{array}{c} (C, R, S, L) \\ | \\ (C \cup \{c\}, R \cup R_c, S, L) \end{array}$$

– *a partition node is a node $n = (C, R, S, L)$ such that:*
  - *each transition with one of its children is a l-transition*
  - *if $n_1, \dots n_p$ are the p children of n, with $n_i = (C, R, S \cup \{c_i\}, L \cup R_{c_i})$ where c_i is a constraint and R_{c_i} its set of reduction operators, then $\bigcup_{i \in [1..p]} c_i$ is the constraint* true.

$$\begin{array}{ccccc} & & (C, R, S, L) & & \\ (C, R, S \cup \{c_1\}, L \cup R_{c_1}) & \dots & (C, R, S \cup \{c_i\}, L \cup R_{c_i}) & \dots & (C, R, S \cup \{c_p\}, L \cup R_{c_p}) \end{array}$$

So, if d is a domain associated to a partition node, then d is partitioned into different subparts of the search space. But, contrary to computation scheme, no subpart of d is lost and so all the solutions are conserved.

Moreover, each branch of a search tree is a computation scheme and verify the same properties: a node which is a terminal computation step is a leaf of the tree. So, there is three kinds of leaf: success solution, failure and the last one correspond to non terminal computation step.

As it was the case for computation schemes, a search tree is associated to the CSP $\mathcal{C} = (V, C, var, D, ext)$ if $(\emptyset, \emptyset, \emptyset, \emptyset)$ is the root node of the search tree and D is its associated domain and if all the non failure leaves $(Cons, R, S, L)$ verify $Cons = C$.

Now, search trees completness is established.

Theorem 27 (Completness). *Let $\mathcal{T}$ be a search tree associated to a CSP $\mathcal{C}$. For any solution of $\mathcal{C}$, there exists a leaf of $\mathcal{T}$ which contains it.*

Proof. Let n be a node (C, R, S, L) and $\{s_1, \ldots, s_q\}$ the solutions of C in $C_{R\cup L}(D)$. If n is a reduction node, since reduction operators are correct, then the domain associated to its child contains $\{s_1, \ldots, s_q\}$. If n is a partition node, let $d_1, \ldots, d_p$ be the associated domains of the p children of n. Since $\bigcup_{i\in[1..p]} d_p = C_{R\cup L}(D)$, then $\forall i \in [1..q],\ \exists j \in [1..p]\ s_i \in d_j$.

3.3 Labeling

As only finit domains are taken into account, search steps are done most of the time by labeling. It consists in partitioning the domain of only one variable of the CSP. So, only constraints of the general form $x\ in\ E$ (E fixed) are used to proceed to a search step. This kind of constraints has only one associated reduction operator. This operators are called labeling operators.

Definition 28 (Labeling Operator). *Let x a variable and $E \subseteq D_x$. The* labeling operator *$f_{x=E}$ associated to the constraint* x in E *is defined by:*

$$\forall d \in \mathcal{SS},\ f_{x=E}(d)_y = d_y \text{ if } y \neq x \text{ and } f_{x=E}(d)_x = d_x \cap E.$$

Labeling operators are monotonic, contracting and idempotent. But, contrary to reduction operators, labeling operators only focus on a part of domains of a CSP which may not contain all, or even any, solution. So, labeling operators are "incorrect" reduction operators. In this context, incorrect means that when a labeling operator is applied, there is no way to know if all solutions, or even a part, are conserved.

Because only one labeling operator is associated to each constraint of S (the set of constraints used for searching), computation steps can be simplified and represented by a 3-uple (C, R, L). Thanks to this simplification, the definition of a partition node can be simplified too.

Definition 29 (Labeling Partition Node). *Let D be the domain associated to the root of a search tree. A node (C, R, L) is a labeling partition one if there is a variable x such that its children have the form $n_1 = (C, R, L \cup \{f_{\{x=E_1\}}\}), \ldots, n_p = (C, R, L \cup \{f_{\{x=E_p\}}\})$ with $\bigcup_{i\in[1..p]} E_i = C_L(D)$.*

Since $d = C_{R\cup L}(D) \subseteq C_L(D)$, some of the p sons of n may be failure terminal leaf. A natural optimization is $\bigcup_{i\in[1..p]} E_i = d$.

As labeling operators are a particular kind of reduction operators, the results established for reduction operators can be applied to labeling operators, and, in particular, lemma 11. An interesting application of this lemma can be done with $\mathcal{F}_1 = R$ a set of reduction operators and $\mathcal{F}_2 = L$ a set of labeling operators. In this particular case, lemma 11-4 can be improved. For this, a property on labeling operators has to be established.

Property 30. Let $f_{x=E}$ be a labeling operator and f a composition of reduction operators in R then:

$$f \circ f_{x=E} = f_{x=E} \circ f \circ f_{x=E}.$$

Proof. Let $d \in \mathcal{SS}$. $f \circ f_{x=E}(d) = f(d') = d''$ with $d'_y = d_y$ for $y \neq x$ and $d'_x = d_x \cap E$. But $d''' = f_{x=E}(d'')$ because $d'' \subseteq d'$ and hence $f \circ f_{x=E}(d) = f_{x=E} \circ f \circ f_{x=E}(d)$.

Thanks to this property, the lemma 11-4 gives the following result:

Lemma 31. *Let $d \in \mathcal{SS}$, L be a set of labeling operators and R be a set of reduction operators. Let $\mathcal{L}$ be the function $d \longrightarrow \bigcap_{l\in L} l(d)$ (pointwise intersection is used). Then:*

$$C_R(C_L(d)) = C_{R \cup L}(d) = C_R(\mathcal{L}(d)).$$

This lemma shows that the closure of a search point by a set of labeling operators L and a set of reduction operators R is equal to the closure by R of the closure by L of this search point. So, the closure by L computation can be done first and then L is not necessary any more. Moreover, the closure by L can be obtained by an unique application of all labeling operators of L on the search point.

As a corollary of the confluence theorem 15, two search trees associated to the same CSP which defined the same labeling for their branches own the same non-failure leaves.

More generally, if all the reduction operators are correct, all the solutions of a CSP are included in the domain of the non-failure leaves. No solution is lost because at each partion node, a partition of the current domain is done and so no part of the search space is lost. Morever, if all the branches with non-failure node are complete then the consistency defined by the set of reduction operators is reached. But, this approximation of the solutions depends on the labeling. The more the labeling cuts the search space into "small pieces", the more precise the approximation of the solutions are. So, in order to have exactly the solutions of a CSP, a *complet labeling* have to be done on all the variables of the CSP.

Definition 32. *Let x be a variable of a CSP. A* complete labeling *is done on x if for all the branches with non-failure leaf of a search tree, the domain d_x is reduced to a singleton in this leaves.*

If all the branches with non-failure leaf of a search tree are complete and that a complete labeling is done on all the variables of the CSP, then the success leaves of the tree have domains which are exactly the solutions of the CSP. Note that this is due to the fact that any solveur is complete for singletons.

Respectively, all the solutions of a CSP correspond to leaves of complete branches in which a complete labeling has been done on all the variables.

References

1. K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.
2. K. R. Apt. The role of commutativity in constraint propagation algorithms. *ACM Transactions on Programming Languages and Systems*, To appear.
3. Frédéric Benhamou. Heterogeneous constraint solving. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Algebraic and Logic Programming, 5th International Conference, ALP'96*, volume 1139 of *lncs*, pages 62–76, Aachen, Germany, sep 1996. Springer.
4. Daniel Chazan and Willard Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2:199–222, 1969.
5. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, pages 238–252, 1977.
6. Daniel Diaz and Philippe Codognet. The GNU Prolog system and its implementation. In *ACM Symposium on Applied Computing*, volume 2, pages 728–732, Villa Olmo, Como, Italy, 2000.
7. Laurent Granvilliers and Eric Monfroy. Constraint propagation: between abstract models and ad-hoc strategies. In Rina Dechter, editor, *Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 505–509, Singapore, September 18-21 2000. Springer. short paper.
8. François Laburthe and the OCRE project. Choco: implementing a CP kernel. In *TRICS, Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Technical report TRA9/00, Singapore, sep 2000.

9. Ugo Montanari and Francesca Rossi. Constraint relaxation may be perfect. *Artificial Intelligence*, 48:143–170, 1991.
10. Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
11. M. H. van Emden. Value constraints in the CLP scheme. *Constraints*, 2:163–183, 1997.
12. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.
13. D.L. Waltz. *Understanding line drawings in scenes with shadows*, pages 19–91. McGraw Hill, 1975.