Joint NCC & IIS Bull., Comp. Science, 16 (2001), 29–44
© 2001 NCC Publisher

A general schema for constraint propagation

R. Gennari

Various algorithms for achieving different levels of local consistency (i.e., constraint propagation algorithms), even diverse ones for the same kind of local consistency, are present in the literature and built into existing systems. Due to their variety and diversity, a natural quest is to search for a common framework. In this article, we approach constraint propagation from a general perspective, by enlarging on previous algorithm schemata and augmenting their expressive power: in that, further constraint propagation algorithms are instances of our schema. This is due to new relations that we establish among function sets and use to instantiate our algorithm schema; these relations result from abstracting common properties of the surveyed constraint propagation algorithms. Hence, our general approach is expressive enough to bring out the common properties of most of the constraint propagation algorithms, as well as their distinctions.

1. Introduction

Constraint programming consists of formulating and solving constraint satisfaction problems. One of the most important techniques developed in that area is *local consistency*, which is also well known as *constraint propagation*. In general, constraint propagation algorithms aim at pruning the search space, without adding or removing solutions. Lots of these algorithms were devised for achieving different levels of local consistency. Besides, various algorithms were designed and optimized for enforcing the same level of local consistency: some of them were specialized for particular domains, other for exploiting a specific orders over variables etc. Moreover, most of constraint propagation algorithms were built into the existing constraint programming systems.

Due to the variety of constraint propagation algorithms in the literature and their importance in practice, a natural quest is to search for a common "thread" among them. In [2, 3], constraint propagation algorithms were approached from a general perspective and many of them were proved to be instances of a unique schema, namely the Generic Iteration algorithm (`GI`). Lately, we generalized `GI` to a more expressive schema, in which the latter can be instantiated to more local consistency algorithms than `GI` can, cf. [8].

In the general framework elaborated in [2, 3], the author pinpointed the basic properties that are common to most of the functions for enforcing local consistency. Following the spirit of our previous work [8], we enlarge on that approach in this paper: in fact, we do not only take into consideration *properties* of functions as in [2, 3]; besides, we study and emphasize the role of *relations* among functions for enforcing constraint propagation. From our analysis, we work out a new generalization of the GI schema in [1, 2] and of ours in [8]: namely, the Generic Iteration algorithm with Functions (GIF).

This article is organized as follows: first, we introduce our new algorithm schema and some of its specializations, cf. Section 2. In Section 3, we define constraint satisfaction problems, some orderings over them and make explicit their connection with the general algorithm schema. Finally, we show that our schema can be instantiated to more constraint propagation algorithms, like PC-4 and KS, than the previous schemata.

2. The GIF algorithm schema and its specializations

In [8], we introduced the *Generic Iteration algorithm with Subsumed Functions* (GISF) as a schema that generalizes the *Generic Iteration algorithm* (GI) of [2, 3]. Moreover, we proved that our schema is more expressive than GI, in that some local consistency algorithms are instances of the former and not of the latter. In this section, we introduce a schema that generalizes GISF, too: namely, the *Generic Iteration algorithm with Functions* (GIF). After introducing GIF, we specialize it: first we recall our previous algorithm schema, namely GISF, and show that GI is an instance of GISF itself; finally, we introduce a new specialization of GIF, the *Generic Iteration algorithm with Included Functions* (GIIF), and prove its correctness.

2.1. The GIF algorithm schema

While the GI algorithm schema selects functions from one set, the GIF algorithm schema can choose functions from two possibly different sets. Therefore, the GIF schema can also be instantiated to local consistency algorithms that are split into two main sub-programs, like AC-4 or PC-4: one performs a "global pruning" and the other — that is not interleaved with the former — achieves the desired level of local consistency by means of some kinds of more "local actions".

```
GENERIC ITERATION ALGORITHM WITH FUNCTIONS (GIF)

1. $d := \bot$;
2. $G := H$;
3. while $G \neq \emptyset$ do
4.     choose $g \in G$;
5.     $G := G - \{g\}$;
6.     $G := G \cup update(G, F, g, d)$;
7.     $d := g(d)$
8. od
```

The *update* operator (6th line of `GIF`) has to satisfy three conditions:

A. if $g(d) \neq d$, then the following functions have to be in $update(G, F, g, d)$: all $f \in F - G$ such that $f(d) = d$ and $f(g(d)) \neq g(d)$;

B. $g(d) = d$ implies $update(G, F, g, d) = \emptyset$;

C. if $g(g(d)) \neq g(d)$, then g is in $update(G, F, g, d)$.

No further restrictions are imposed on the general schema. Instead, in the following, we shall study diverse conditions, namely *properties of functions or relations among them*, under which that unique schema can be applied to enforce various forms of local consistency; like, for instance, path consistency. Besides, we can express and analyze different algorithms for the same level of local consistency; for example `PC-1` (cf. [1]) and `PC-4` (cf. Section 4). In fact, those algorithms can be instantiated to our schema by means of *specific functions*.

Note 1. Suppose that g is *idempotent*; that is, for every $d \in D$, $g(g(d)) = g(d)$. In this case, g does not need to be added to $update(G, F, g, d)$ according to the third condition **C**; cf. [2, 3].

2.2. The `GISF` algorithm schema

The `GISF` algorithm is an instance of the `GIF` schema. In `GISF`, the sets of functions F and H are not arbitrary but related in order to guarantee that a fixed point of all the functions from H is a fixed point of all the functions from F. Precisely, let f and g be two functions on a set D; we say that the *function g subsumes the function f* iff $g(d) = d$ implies $f(d) = d$. Let F and H be two sets of functions defined on the same set D; we say that the *set H subsumes the set F* iff each function of F is subsumed by a function of H; in that case, we write $subs(F, H)$ (cf. [8]). On the overall, the condition $subs(F, H)$ guarantees that d is a fixed point of all the functions from F if it is a fixed point of all the functions from H.

Note 2. In general, it is not trivial to establish whether a function g subsumes a function f. However, suppose that f and g are functions defined on $\langle D, \sqsubseteq\rangle$ and that f is inflationary with respect to $\sqsubseteq$: i.e., for all $d \in D$, $d \sqsubseteq f(d)$ holds. Further, if $f(d) \sqsubseteq g(d)$ for every $d \in D$, then g subsumes f.

The following result, concerning the correctness of `GISF`, was proved in [8].

Theorem 1. (`GISF`) *Let $(D, \sqsubseteq)$ be a partial ordering with bottom, $\bot$; suppose that H and F are two sets of functions on D; if the relation $subs(F, H)$ holds, then the following statements are valid.*

i. Every terminating execution of the `GISF` *algorithm computes in d a common fixed point of the functions in $H \cup F$.*

ii. Suppose that all the functions of $H \cup F$ are monotonic. Then every terminating execution of the `GISF` *algorithm computes in d the least common fixed point of all the functions from H and F.*

iii. Suppose that H and F contain finitely many functions which are all inflationary. Further, assume that the strict partial order on D satisfies the ascending chain condition (ACC): namely, there are not infinite ascending chains of D elements. Then every execution of the `GISF` *algorithm terminates.* □

Observe that $subs(H, H)$ always holds; this means that $subs$ is a reflexive relation. Hence the `GI` algorithm schema is an instance of our `GISF` algorithm itself; in fact, it is enough to set $F = H$ in the latter to obtain the `GI` algorithm of [2, 3]. Besides, the `GISF` schema is strictly more "expressive" than `GI`: there are local consistency (precisely, arc consistency) algorithms in it that are instances of `GISF` but not of `GI`; cf. [8].

2.3. The `GIIF` algorithm schema

As we have noticed, the `GISF` schema is already a generalization of `GI` and is more expressive than the latter. Yet, there are constraint propagation algorithms that are instances of neither `GI` nor `GISF`, like the k-consistency algorithm of Cooper, cf. Section 5. In those algorithms, the relation between H and F is of another sort: basically, H is a subset of F that contains all $f \in F$ for which $f(\bot) \neq \bot$; then the *update* operator picks out from F the functions that are still to be inspected and adds them to G. The Generic Iteration algorithm with Included Functions (`GIIF`) is this new instantiation of the `GIF` algorithm schema. In the following, we prove the partial and total correctness of `GIIF`.

Theorem 2 (`GIIF`). *Let $(D, \sqsubseteq)$ be a partial ordering with bottom $\perp$; suppose that H and F are two sets of functions on D; if H is a subset of F that includes the set $\{f \in F \,:\, f(\perp) \neq \perp\}$, then the following statements hold.*

i. Every terminating execution of the `GIIF` *algorithm computes in d a common fixed point of the functions in F.*

ii. Suppose that all the functions in F are monotonic. Then every terminating execution of the `GIIF` *algorithm computes in d the least common fixed point of all the functions from F.*

iii. Suppose that F has finitely many functions which are all inflationary. Further, assume that the strict partial order on D satisfies the ascending chain condition (ACC): namely, there are not infinite ascending chains of D elements. Then every execution of the `GIIF` *algorithm terminates.*

Proof. We just need to prove the first item, the proof of the other two is like in [8] for the case of `GISF`. Consider the predicate I defined by

$$\forall f\,(f \in F - G \wedge f \in H \rightarrow f(d) = d).$$

The predicate I is established by the assignment $G := H$; in fact, if $f \in F - G$, then $f \notin H$, hence I trivially holds. Now, suppose that I holds before a **while**-loop is entered. After an iteration of the **while**-body, only the inspected function g of F can be added to $F - G$, just in case of $g(g(d)) = g(d)$; hence, for the new computed value of d after the execution of the **while**-body, we have that I still holds. Thereby, I is an invariant of the **while**-loop. Upon the termination of the algorithm, G is empty and $H = F \cap H$, so I implies I', defined by

$$\forall f\,(f \in H \rightarrow f(d) = d);$$

hence the predicate I' holds as well. The predicate I' guarantees that d is a fixed point of all the functions from H. We claim that d is a fixed point of all the functions from F, too, by definition of *update*. Let $\perp =: d_0, \ldots, d_n := d$ be the sequence from D computed by the `GIIF` algorithm, so that, for every $i = 0, \ldots, (n-1)$, $d_{i+1} = g_{i+1}(d_i)$, where $g_{i+1} \in F$, such a sequence exists because we assume that the algorithm terminates. Suppose that there exists $f \in (F - H)$ such that $f(d_n) \neq d_n$; observe that $f \notin H$ implies that $f(d_0) = d_0$. Since the sequence is finite, $f(d_n) \neq d_n$ and $f(d_0) = d_0$, there must be a maximal $i = 0, \ldots, (n-1)$ such that $f(d_i) = d_i$ and $f(d_j) \neq d_j$ for all j such that $i < j \leq n$. Then *update*(G, F, g_{i+1}, D) adds G the function f, because of condition **A**; notice that f cannot be removed from G in any subsequent iteration of the **while**-loop, because of the conditions **C** and $f(d_j) \neq d_j$ for all j for which $i < j \leq n$. Hence G is not empty after processing g_n, which is absurd. $\square$

3. Constraint satisfaction problems and partial orderings

In order to apply the `GIF` algorithm schema over CSP's, we need to define proper orders among CSP's. In the following, we introduce different orderings that vary according to the local consistency algorithms which are surveyed in the article.

3.1. Constraint satisfaction problems

Consider a finite sequence X of different variables, say $x_1, \ldots, x_n$ for $n > 0$, with associated domains $D_1, \ldots, D_n$. A *constraint sequence* s, briefly *c-sequence*, on $n > 0$ is a strictly growing sequence of different integers from $1, \ldots, n$. Let $\boldsymbol{D}$ be the Cartesian product $D_1 \times \cdots \times D_n$ and s be the c-sequence $i_1, \ldots, i_m$ on n. Then we denote by $\boldsymbol{D}(s)$ the Cartesian product $D_{i_1} \times \cdots \times D_{i_n}$. For instance, if $D_1 = \{0\}$, $D_2 = \{2, 6\}$, $D_3 = \{4\}$ and s is the c-sequence $1, 3$, then $\boldsymbol{D}(s)$ is the set $\{(0, 4)\}$. Further, we shall denote by $d(s)$ an element of $\boldsymbol{D}(s)$, for a tuple d of $D_1 \times \cdots \times D_n$: i.e., if s is the c-sequence $i_1, \ldots, i_m$ on n and $d = (d_1, \ldots, d_n)$, then $d(s)$ is the tuple $(d_{i_1}, \ldots, d_{i_m})$. Given two c-sequences on n of equal length $m \leq n$, say $s = i_1, \ldots, i_m$ and $t = j_1, \ldots, j_m$, we write $s <_{sch} t$ if, for all $k = 1, \ldots, l < m$, we have that $i_k = j_k$ and $i_l < j_l$. Moreover, we write $s <_{sch} t$ if s and t are c-sequences on n and the length of s is strictly less than that of t. Hence the relation $<_{sch}$ is a total order on c-sequences on n.

Definition 1. Let X be a sequence of $n > 0$ different variables with domains $D_1, \ldots, D_n$, the set $\boldsymbol{D}$ be the Cartesian product $D_1 \times \cdots \times D_n$ and s be a c-sequence on n; a *constraint on* s is a subset of $\boldsymbol{D}(s)$. Then we write $C(s)$, or C when no confusion can arise. A *constraint satisfaction problem on* X, briefly CSP, is a triple $P := \langle X, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{D}$ is the sequence of domains $D_1, \ldots, D_n$ and $\mathcal{C}$ is a sequence of constraints $C(s_1), \ldots, C(s_n)$, that is ordered according to the order $s_1 <_{sch} \cdots <_{sch} s_n$ on c-sequences.

If s is a c-sequence on n, then $\{s\}$ is the corresponding set of integers occurring in s; for instance, if s is the c-sequence $1, 3, 4$, then $\{s\}$ is the set $\{1, 3, 4\}$. Observe that every set of $i \leq n$ integers uniquely determines the c-sequence s on n to which it corresponds; in fact a c-sequence is a strictly growing sequence of integers, so, for example, the set $\{1, 3, 4\}$ determines the c-sequence $1, 3, 4$. A *c-subsequence* of s is just a c-sequence t on n such that $\{t\}$ is a subset of $\{s\}$. Consider a CSP P on n variables and a c-sequence $s = i_1, \ldots, i_k$ on n. The set $I(s)$ of all *consistent instantiations relative to* s is the set of all $d \in D(s)$ such that $d(t) \in C(t)$, for all $C(t)$ of P on a c-subsequence t of s.

A *solution* to a CSP P on n variables is a tuple of $I(s)$, where s is the c-sequence of all integers $1, \ldots, n$; then $I(s)$ is the *solution set* of P, usually written as $Sol(P)$. A CSP P is *globally consistent* iff $\mathbf{D} = Sol(P)$. Two CSP's on the same sequence of variables X are *equivalent* iff they have the same solution set.

3.2. Partial orderings

So far, we have an algorithm schema, namely `GIF`, that is able to compute the common fixed point of functions defined on a partial ordering with bottom. We aim at applying the `GIF` algorithm to CSP's and instantiating it to some local consistency algorithms that modify constraints. Hence, we need to feed the `GIRF` algorithm with suitable functions that are capable of modifying constraints, as well as to devise a partial order between problems. In the following, we define the orderings that we shall use lately in this article; cf. also [1] and [4] for similar ones.

Definition 2. Consider a CSP P and all its constraints $C_1, \ldots, C_n$. The *completion of* P is the CSP $\bar{P}$ that has the same sequence of variables and domains as P, but the constraints of which are as follows: for each c-sequence s on n, if $C(s) \in \mathcal{C}$, then $C(s)$ is the constraint on s of $\bar{P}$; otherwise $C(s)$ is $D\,[s]$. We say that a CSP P is *complete* iff $P = \bar{P}$.

However, if we work with *binary* CSP's P (CSP's that have only binary constraints), the choice of $\bar{P}$ is not optimal: we may add too many constraints to P. Hence, we refine the above definition as follows.

Definition 3. Consider a CSP P on $n > 0$ variables, a natural number k not greater than n, two CSP's $\bar{P}_k$ and $\bar{P}^s_k$ that have the same sequence of variables and domains as P. Then $\bar{P}_k$ is the *k-completion of* P if the constraints of $\bar{P}_k$ are all the k-ary constraints of $\bar{P}$; the problem P is *k-complete* iff $P = \bar{P}_k$. Whilst $\bar{P}^s_k$ is the *k-strong completion of* P iff the constraints of $\bar{P}^s_k$ are all the i-ary constraints of $\bar{P}$ for every $0 < i \leq k$, the problem P is *k-strong complete* iff $P = \bar{P}^s_k$.

A CSP P and its completions defined above are *equivalent* problems. Furthermore, a CSP P on n variables is n-strong complete iff it is complete.

Definition 4. Consider a CSP P and the Cartesian product $\boldsymbol{C}$ of all the constraints of $\bar{P}$. The *constraint order of* P is the binary relation $\sqsubseteq$ defined as follows: given two subsets B and B' of $\boldsymbol{C}$, $B \sqsubseteq B'$ iff $B \supseteq B'$.
Let $\mathcal{F}(\boldsymbol{C})$ be a family of subsets of $\boldsymbol{C}$ to which $\boldsymbol{C}$ belongs. Then $\langle \mathcal{F}(\boldsymbol{C}), \sqsubseteq, \boldsymbol{C} \rangle$ is a *constraint ordering of* P and its bottom is $\boldsymbol{C}$.

It is immediate to verify that the binary relation introduced above is a partial order, because so is $\supseteq$. Moreover, the Cartesian products $\boldsymbol{C}$ is the bottom of $\sqsubseteq$.

Observe that, given a CSP P, there is always a constraint ordering of P, namely, the one based on the power set $\wp(\boldsymbol{C})$. Yet, we may want to deal only with constraints of a fixed arity, like in path or k-consistency algorithms; hence we restrict the above introduced order as follows.

Definition 5. Consider a CSP P and the Cartesian product $\boldsymbol{C}_k$ of all the constraints of $\bar{P}_k$. The *k-constraint order of* P is the restriction of $\sqsubseteq$ to subsets of $\boldsymbol{C}_k$: we write it as $\sqsubseteq_k$. Let $\mathcal{F}(\boldsymbol{C}_k)$ be a family of subsets of $\boldsymbol{C}_k$ to which $\boldsymbol{C}_k$ belongs. Then $\langle \mathcal{F}(\boldsymbol{C}_k), \sqsubseteq_k, \boldsymbol{C}_k \rangle$ is a *k-constraint ordering of* P with bottom $\boldsymbol{C}_k$.
Consider now the Cartesian product $\boldsymbol{C}^s_k$ of all the constraints of $\bar{P}^s_k$. The *k-strong constraint order of* P is the restriction of $\sqsubseteq$ to subsets of $\boldsymbol{C}^s_k$: we write it as $\sqsubseteq^s_k$. Let $\mathcal{F}(\boldsymbol{C}^s_k)$ be a family of subsets of $\boldsymbol{C}^s_k$ to which $\boldsymbol{C}^s_k$ belongs. Then $\langle \mathcal{F}(\boldsymbol{C}^s_k), \sqsubseteq^s_k, \boldsymbol{C}^s_k \rangle$ is a *k-constraint ordering of* P the bottom of which is $\boldsymbol{C}^s_k$.

Indeed, the restriction of $\wp(\boldsymbol{C})$ to k-ary relations gives rise to a k-constraint ordering of P; considering the restriction to all i-ary relations of $\wp(\boldsymbol{C})$, for each $0 < i \leq k$, we obtain a k-strong constraint ordering of P.

4. The path consistency algorithm `PC-4`

The `PC-4` algorithm was designed in [10]; however, here we refer to its corrected form given in [9]. This algorithm enforces path consistency on binary CSP's. In this section, we prove that `PC-4` is an instance of the `GISF` algorithm schema; hence, in the following, we restrict our attention to binary CSP's.

4.1. Preliminaries

Given a constraint $C(i,j)$ and any its subset $B(i,j)$, the *transposed* $B(i,j)$ is the relation $B(j,i)$ the elements of which are all pairs (d,d') such that $(d',d) \in B(i,j)$. The transposed relations will help us to better describe the `PC-4` algorithm; however, given a CSP P and a constraint $C(i,j)$ of it, $C(j,i)$

is not a constraint of P, according to Definition 1. The `PC-4` algorithm is split into two parts, as explained in the following.
The first part. Suppose we are given a constraint $C(i,j)$ of the input CSP (i.e., $i < j$) and a variable x_k of the problem such that $k \neq i, j$. For each $(a,b) \in C(i,j)$, the algorithm checks whether there exists c in D_k such that $(a,c) \in C(i,k)$ and $(c,b) \in C(k,j)$; if c exists, then $Total := Total + 1$, $S_{iakc} := S_{iakc} \cup \{(j,b)\}$ and $S_{jbkc} := S_{jbkc} \cup \{(i,a)\}$. After the search for supports in D_k is complete, $Counter\,[(i,a,j,b),k]$ is set to $Total$ and so is $Counter\,[(j,b,i,a),k]$. If (a,b) has no supports in D_k (i.e. $Total$ is 0), then the algorithm sets $M\,[i,a,j,b]$ and $M\,[j,b,i,a]$ to 1 in order to record that either (a,b) or (b,a) are to be eliminated. So the set $List$ is initialized with all the tuples (i,a,j,b) for which M is 1.
The second part. The second part consists of a **while**-loop that terminates when $List$ is empty. An element (i,a,j,b) is chosen and deleted from $List$. Then all pairs (k,c), such that $(a,c) \in C(i,k)$ and $(c,b) \in C(k,j)$, are checked. Suppose that (a,c) is no longer supported in $C(i,k)$ or (c,b) is no longer supported in $C(k,j)$ (i.e., $Counter\,[(i,a,k,c),j] = 0$ or $Counter\,[(j,b,k,c),i] = 0$); if (a,c) or (c,a), and (c,b) or (b,c) have not been removed yet (i.e., $M\,[i,a,k,c] = 0$ and $M\,[j,b,k,c] = 0$), then they are deleted and the effects of their removal are propagated by adding (i,a,k,c) and (j,b,k,c) to $List$.

4.2. `PC4` is an instance of `GISF`

While arc consistency algorithms remove elements from domains (cf. [8]), path consistency algorithms delete pairs from binary constraints. Consider the 2-constraint closure of P and the Cartesian product $\boldsymbol{C}_2$ of its constraints; hence, our functions are of the form $f : B \longrightarrow B$, where B is a subset of $\boldsymbol{C}_2$. As in the case of `(G)AC-4` (cf. [8]), we shall instantiate `GISF` with two sets of functions, H and F: the functions of H perform a "global" action, so to speak, and take care of the first part of the algorithm used to create $List$; the functions of F have a more "local" behavour.

For the sake of readability, we introduce a new relation that we shall use for defining our functions: consider a subset B of $\boldsymbol{C}_2$; assuming that $0 < i < j \leq n$, the pair $(a,b) \in B(i,j)$ belongs to $Del(B;i;j;k)$ iff, for all $c \in D_k$,

$$(a,c) \notin B(i,k) \vee (c,b) \notin B(k,j).$$

Notice that the relation $Del(B;i;j;k)$ is a subset of $B(i,j)$.

1. For every constraint $C(i,j)$ of the given P_2, $k = 1, \ldots, n$, $k \neq i, j$, and $(a,b) \in C(i,j)$, we define a function $\pi(i,a,j,b;k)(B) := B'$,

where $B,\ B' \subseteq \boldsymbol{C}_2$ and each $B(r,m)'$ is defined as follows:

$$B(r,m)' := \begin{cases} B(r,m) - \{(a,b)\} & \text{if } r = i,\ m = j \text{ and } (a,b) \in Del(B;i,j;k), \\ B(r,m) & \text{otherwise.} \end{cases}$$

Basically, $\pi(i,a,j,b;k)$ removes (a,b) from $B(i,j)$ iff there is no $c \in D_k$ such that $(a,c) \in B(i,k)$ and $(c,b) \in B(k,j)$.

2. Let us consider two constraints $C(i,k)$ and $C(k,j)$ of $\boldsymbol{C}_2$, or their transposed; further, let us choose $(a,c) \in C(i,k)$ and $(c,b) \in C(k,j)$. Then we define two new different functions, namely $\pi(i,a,j,b;k,c)(B) := B'$ and $\pi(j,b,i,a;k,c)(B) := B''$, in the following way:

$$B(r,m)' := \begin{cases} B(i,k) - \{(a,c)\} & \text{if } r = i,\ m = k,\ i < k \text{ and} \\ & (a,c) \in Del(B;i,k;j), \\ B(k,i) - \{(c,a)\} & \text{if } r = k,\ m = i,\ k < i \text{ and} \\ & (c,a) \in Del(B;k,i;j), \\ B(r,m) & \text{otherwise;} \end{cases}$$

$$B(r,m)'' := \begin{cases} B(k,j) - \{(c,b)\} & \text{if } r = k,\ m = j,\ k < j \text{ and} \\ & (c,b) \in Del(B;k,j;i), \\ B(j,k) - \{(b,c)\} & \text{if } r = j,\ m = k,\ j < k \text{ and} \\ & (b,c) \in Del(B;j,k;i), \\ B(r,m) & \text{otherwise.} \end{cases}$$

Intuitively, the function $\pi(i,a,j,b;k,c)$ removes (a,c) ((c,a) if $k < i$) from $B(i,k)$ (from $B(k,i)$ if $k < i$) iff a or c have lost their unique support b in D_j; whilst $\pi(j,b,i,a;k,c)$ removes the pair (c,b) ((b,c) if $j < k$) from $B(k,j)$ (from $B(j,k)$ if $j < k$) iff c or b have lost their unique support a in D_i.

Notice that, for every function $\pi(i,a,j,b;k) \in H$ and $\pi(i,a,k,c;j,b) \in F$, the following relation holds, if $i < j$:

$$\pi(i,a,j,b;k)(B) \subseteq \pi(i,a,k,c;j,b)(B);$$

otherwise the following is true if $j < i$:

$$\pi(j,a,i,b;k)(B) \subseteq \pi(i,a,k,c;j,b)(B).$$

Thereby $subs(F,H)$ holds. Now, we can define *update* as follows.

- If $\pi(i,a,j,b;k)(B) = B$, then $update(G,F,\pi(i,a,j,b;k),B) = \emptyset$. Otherwise, the set $update(G,F,\pi(i,a,j,b;k),B)$ contains all the following functions:

  1. $\pi(i,a,j,b;k,c)$ such that $(a,c) \in B(i,k)$, $(c,b) \in B(k,j)$ and furthermore, for all $b' \in D_j$, we have that $(c,b') \notin B(k,j)$;
  2. $\pi(j,b,i,a;k,c)$ such that $(a,c) \in B(i,k)$, $(c,b) \in B(k,j)$ and furthermore, for all $a' \in D_i$, we have that $(a',c) \notin B(i,k)$.

- If $\pi(i,a,j,b;k,c)(B) = B$, then $update(G,F,\pi(i,a,j,b;k,c),B) = \emptyset$. Otherwise, the set $update(G,F,\pi(i,a,j,b;k,c),B)$ contains all the following functions:

  1. $\pi(i,a,k,c;l,e)$ such that $(a,e) \in B(i,l)$ and $(e,c) \in B(l,k)$ and furthermore, for all $c' \in D_k$ different from c, we have that $(e,c') \notin B(l,k)$;
  2. $\pi(k,c,i,a;l,e)$ such that $(a,e) \in B(i,l)$ and $(e,c) \in B(l,k)$ and furthermore, for all $a' \in D_i$ different from a, we have that $(a',e) \notin B(i,l)$.

The *update* operator above satisfies **A** and **B**; the last condition, **C**, is trivially fulfilled, because all of the considered functions are inflationary, cf. Note 1.

Corollary 1 (GISF for path consistency). *Consider a CSP $P := \langle X, \mathcal{D}, \mathcal{C}\rangle$ with the associated 2-constraint order and the sets of functions H and F as defined above. If $\mathcal{D}$ is finite, then every execution of the* GISF *algorithm terminates computation of the greatest path consistent problem equivalent with P.*

Proof. Indeed a fixed point of the functions from H is a path consistent problem equivalent to the given one. As $subs(F,H)$ holds, a fixed point of the functions from H is a fixed point of the functions from F, too. Furthermore, if all domains are finite, so is $H \cup F$. Our statement follows now from Theorem 1. □

We are left to prove that the PC-4 algorithm is indeed an instance of GISF.

Theorem 2 (GISF for PC-4). PC-4 *is an instance of the* GISF *algorithm schema.*

Proof. The first part of the PC4 algorithm is reproduced by iterating GISF with the functions from H, none of which is any longer introduced by *update*. For every constraint $C(i,j)$ of P_2 (line 2 of PC-4), $k = 1, \dots, n$ (line 3 of PC-4) and $(a,b) \in C(i,j)$ (line 4 and 5 of PC-4), the first **for**-loop of the PC-4 algorithm checks whether there exists $c \in D_k$ that supports a and b; if there is no such c, then (a,b) is removed from $C(i,j)$ and the effects of its removal are propagated. The function $g(i,a,j,b;k)$ has a similar behaviour: it removes the pair (a,b) iff there is no support in D_k for a and b; then the effects of the removal of (a,b) are propagated by means of the *update* operator.
After inspecting all functions from H, we feed GISF with the functions from F that are added by *update*; so we can reproduce the second part of the PC-4 algorithm, namely, the second **for**-loop, by choosing the functions $\pi(i,a,j,b;k,c)$ and $\pi(j,b,i,a;k,c)$, consecutively. $\square$

5. The KS algorithm

The KS algorithm by Cooper [5] is an optimization of the synthesis algorithm by Freuder [7]. The algorithm by Cooper can enforce either k-consistency or k-strong consistency over a CSP. In this section, we prove that the GIIF schema can be instantiated to the KS algorithm.

5.1. Preliminaries

The concepts of arc and path consistency were generalized in [7] to k consistency. Given a CSP P on n variables and an integer $1 \leq k < n$, consider a c-sequence $s := i_1, \dots, i_k$ on n. Consider a positive integer $j \notin s$ and $1 \leq j \leq n$, and a c-sequence s' on $\{s\} \cup \{j\}$; then a k-consistent instantiation d of $I(s)$ is a *projection* (*over* s) of a $(k+1)$-consistent instantiation of $I(s')$ iff there exists $d' \in I(s')$ such that $d'(s) = d$. A CSP P on n variables is 1-*consistent* iff, for every $i = 1, \dots, n$, the set $I(i)$ is not empty. A CSP P on n variables is $(k+1)$-*consistent* for $0 < k \leq n$ iff any k-consistent instantiation is a projection of a $(k+1)$-consistent instantiation. Furthermore, the CSP P is k-*strong consistent* iff it is i-consistent for each $0 < i \leq k$. In particular, P on n variables is consistent if it is n-strong consistent.

The KS algorithm is split into two main sub-programs: the initialization process takes place in the first step; then the pruning of k-inconsistent values from domains begins the second step. The second sub-program consists of two main actions: the algorithm chooses a tuple d that is already removed from $C(t)$, for t of length i; if $i < k$, the effects of the removal of d are first propagated considering all $(i+1)$-consistent instantiations d'

such that $d'(t) = d$; if $i > 1$, the effects of the removal of d are propagated considering all $(i-1)$-consistent instantiations d' such that $d' = d(s)$ for all c-subsequences s of t of length $(i-1)$.

5.2. GIIF can enforce k and k-strong consistency

Given a c-sequence s on n of length $0 < i \leq k$, let us choose an element $d \in D(s)$. Consider $\boldsymbol{C}_k^s$ and, if $(i+1) < k$, a c-sequence s^+ of length $(i+1)$, a c-subsequence of which is s; furthermore, if $(i-1) > 0$, choose a c-subsequence s^- of s whose length is $(i-1)$. Then we consider any subset B of $\boldsymbol{C}_k^s$. We can now define two subsets, the first of $B(s^+)$ and the second of $B(s^-)$, that will help us to better describe our functions. Suppose that $d \notin B(s)$; then we have that

$$d^+ \in Del^+(B(s^+), d) \quad \text{iff} \quad d = d^+(s),$$
$$d^- \in Del^-(B(s^-), d) \quad \text{iff} \quad d(s^-) = d^- \text{ and } \forall\, a\,(a \in B(s) \Rightarrow a(s^-) \neq d^-).$$

Instead, if $d \in B(s)$, then both $Del^+(B(s^+), d)$ and $Del^+(B(s^-), d)$ are empty.

1. Consider a c-sequence s on n of length $1 \leq i < k$, a tuple $d \in D(s)$ and any subset B of $\boldsymbol{C}_k^s$; then we define the function $\pi^+(s,i,d)(B) := B'$, where $B' := B'_1 \times \ldots \times B'_n$ and, for every t, we have

$$B(t)' := \begin{cases} B(t) - Del^+(B(t), d) & \text{if } s \text{ is a c-subsequence of } t \\ & \quad \text{and the length of } t \text{ is } i+1, \\ B(t) & \text{otherwise.} \end{cases}$$

   Basically, $\pi^+(s,i,d)$ removes all the tuples of length $(i+1)$ a projection of which is d, if d is not in $B(s)$.

2. Consider a c-sequence s on n of length $1 < i \leq k$, a tuple $d \in D(s)$ and any subset B of $\boldsymbol{C}_k^s$; then we define the function $\pi^-(s,i,d)(B) := B'$, where $B' := B'_1 \times \ldots \times B'_n$ and, for every t, we have

$$B(t)' := \begin{cases} B(t) - Del^-(B(t), d) & \text{if } t \text{ is a c-subsequence of } s \\ & \quad \text{and the length of } t \text{ is } i-1, \\ B(t) & \text{otherwise.} \end{cases}$$

   Basically, $\pi^-(s,i,d)$ removes all the tuples of length $(i-1)$ that are projections of d but of no other i-consistent instantiation, if $d \notin B(s)$.

We instantiate G with the subset H of all functions $\pi^+(s,i,d)$ and $\pi^-(s,i,d)$ such that $d \notin C(s)$; observe that the functions modifying $\boldsymbol{C}_k^s$ are among those of H. Then the *update* operator will take care of adding all the functions of F we need for enforcing (strong) k-consistency.

- If $\pi^+(s,i,d)(B) = B$, then $update(G,F,\pi^+(s,i,d),B) = \emptyset$. Otherwise, the set $update(G,F,\pi^+(s,i,d),B)$ contains all functions $\pi^+(s',i,d')$, for each s' a c-subsequence of which is s and the length is $(i+1)$, and $d' \in Del^+(B(s'),d)$.
- If $\pi^-(s,i,d)(B) = B$, then $update(G,F,\pi^-(s,i,d),B) = \emptyset$. Otherwise, the set $update(G,F,\pi^-(s,i,d),B)$ contains all functions $\pi^+(s',i,d')$, for each c-subsequence s' of s the length of which is $(i-1)$, and $d' \in Del^-(B(s'),d)$.

Corollary 1 (`GIIF` for k and k-strong consistency). *Consider a CSP P with a finite variable domain.*

- *If we instantiate* `GIIG` *only with functions of type π^+, then the algorithm terminates, computing the greatest k-consistent problem that is equivalent to the input one.*
- *If we instantiate* `GIIF` *with all the above defined functions, then the algorithm terminates, computing the greatest k-strong consistent problem that is equivalent to the input one.*

Proof. Observe that a fixed point of all the functions π^+ and π^- is a k-strong consistent problem equivalent to the given one; as well, if we just instantiate F and H with the π^+ functions, then we enforce k-consistency. As the variable domains are finite, so is F; furthermore, the relation $H \subseteq F$ holds and H contains all the functions that modify the bottom $\mathbf{C}_k^s$. So we can apply Theorem 2. □

Now we can prove that `KS` is an instance of the `GIIF` algorithm schema.

Theorem 2 (`GIIF` for `KS`). *The* `KS` *algorithm is an instance of* `GIIF`.

Proof. The instantiation of *List* (that contains the elements to be removed) in `KS` is reproduced by the instantiation $G := H$ in `GIIF`. The second subprogram of the `KS` algorithm is split into two parts. The functions of the form π^+ take care of the first part, while the functions of the form π^- are employed in the second. □

Note 1. As remarked in [5], the `KS` algorithm only enforces k-consistency if the second step of the second subprogram is never executed. So does the `GIIF` algorithm if we use only the functions of the form π^+, cf. also Corollary 1.

6. Conclusions

In this paper, we enlarged on previous algorithm schemata for local consistency ([2, 8]), establishing a more expressive schema, namely `GIF`, specializations of which the previous schemata turned out to be. After analyzing various algorithms for achieving different levels of local consistency, we brought out some of the relations among the sets of functions that are sufficient for correctness of `GIF` and necessary for instantiating it to most of the constraint propagation algorithms: according to the relation held and the functions chosen, we obtain a specialization of `GIF` for a specific local consistency algorithm. In this paper, we surveyed the `PC-4` and `KS`, and proved that they are all instances of specializations of `GIF`; in a previous work ([8]), we did it for `AC-4`, `AC-5` and their generalization for hyper arc consistency; cf. also [2, 3] for many other constraint propagation algorithms which `GI`, hence `GIF`, can be instantiated to.

References

[1] K. R. Apt, *The Rough Guide to Constraint Propagation*, Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99), Springer-Verlag Lecture Notes in Computer Science 1713, pp. 1–23.

[2] K. R. Apt, *The Rough Guide to Constraint Propagation*, Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99), Springer-Verlag Lecture Notes in Computer Science 1713, pp. 1–23.

[3] K. R. Apt, *The Role of Commutativity in Constraint Propagation Algorithms*, ACM TOPLAS, 22(6), pp. 1002–1036, 2000.

[4] S. Bistarelli, R. Gennari and F. Rossi, *Constraint Propagation for Soft Constraint Satisfaction Problems: Generalization and Termination Conditions*, Proc. of the 6th International Conference on Principles and Practice of Constraint Programming (CP2000), Springer-Verlag Lecture Notes in Computer Science, 1894, pp. 83–97.

[5] M. Cooper, *An Optimal k-Consistency Algorithm*, Artificial Intelligence, 41, pp. 89–95, 1989.

[6] R. Decheter and Peter van Beek, *Local and Global Relational Consistency*, Theoretical Computer Science, 173, pp. 283–308, 1997.

[7] E. C. Freuder, *Synthesizing constraint expressions*, Communication of ACM, 21, pp. 958–966, 1978.

[8] R. Gennari, *Arc Consistency via Iterations of Subsumed Functions*, Proc. of Computational Logic 2000 (CL2000), Springer-Verlag Lecture Notes in Computer Science, 1861, pp. 358–372.

[9] C.-C. Han and C.-H. Lee, *Comments on Mohr and Henderson's Path Consistency Algorithm*, Artificial Intelligence, 36, pp. 125–130, 1986.

[10] R. Mohr and T. C. Henderson, *Arc and Path Consistency Revisited*, Artificial Intelligence, 28, pp. 225–233, 1986.

[11] U. Montanari, *Networks of constraints: Fundamental properties and applications to picture processing*, Information Science, 7(2), pp.95–132, 1974.