On Forward Checking for Non binary Constraint Satisfaction *

Christian Bessière[†]
LIRMM-CNRS, 161 rue Ada, 34392 Montpellier, France
bessiere@lirmm.fr

Pedro Meseguer
IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
pedro@iiia.csic.es

Eugene C. Freuder
Cork Constraint Computation Centre,
University College Cork, Cork, Ireland
e.freuder@4c.ucc.ie

Javier Larrosa
Dep. LSI, UPC, Jordi Girona Salgado, 1-3,
08034 Barcelona, Spain
larrosa@lsi.upc.es

Abstract

Solving non binary constraint satisfaction problems, a crucial challenge today, can be tackled in two different ways: translating the non binary problem into an equivalent binary one, or extending binary search algorithms to solve directly the original problem. The latter option raises some issues when we want to extend definitions written for the binary case. This paper focuses on the well-known forward checking algorithm, and shows that it can be generalized to several non binary versions, all fitting its binary definition. The classical non binary version, proposed by Van Hentenryck, is only one of these generalizations.

1 Introduction

In the last two decades, most of the research done in constraint satisfaction assumed that constraint problems can be exclusively formulated in terms of binary constraints. While many academic problems (n-queens, zebra, etc.) fit this condition, many real problems include non binary constraints. It is well known the equivalence between binary and non binary formulations [15]. Theoretically, this equivalence solves the issue of algorithms for non binary problems. In practice, however, it presents serious drawbacks concerning spatial and temporal requirements, which often make it inapplicable. The translation process generates new variables, which may

*This work was supported by an Integrated Action financed by the Generalitat de Catalunya and by the Spanish CICYT project TAP99-1086-C03 (P. Meseguer and J. Larrosa), by an "action incitative CNRS/NSF" under Grant no. 0690 (C. Bessière), and by the National Science Foundation under Grant No. IRI-9504316 (E.C. Freuder). E.C. Freuder contributed to this work while at the University of New Hampshire Computer Science Department; he is currently supported by a Science Foundation Ireland Principal Investigator Award.

†Member of the Coconut group (http://www.lirmm.fr/~bessiere/COCONUT/).

have very large domains, causing extra memory requirements for algorithms. In some cases, solving the binary formulation can be very inefficient [1]. In any case, this forced binarization generates unnatural formulations, which cause extra difficulties for constraint solver interfaces with human users.

An alternative approach consists in extending binary algorithms to non binary versions, able to solve non binary problems in their original formulation. This approach eliminates the translation process and its drawbacks, but it raises other issues, among which how a binary algorithm is generalized is a central one. For some algorithms, such as *chronological backtracking* (BT) [7] or *maintaining arc consistency* (MAC) [16], this extension presents no conceptual difficulty: their binary definitions allow only one possible non binary generalization. For other algorithms, such as *forward checking* (FC) [8], several generalizations are possible.

In this paper, we study how the popular FC algorithm can be extended to consider non binary constraints. We present different generalizations, all collapsing to the standard version in the binary case. Our intention is mainly conceptual, trying to draw a clear picture of the different options for non binary FC. We also provide some experimental results to initially assess the relative performance of the studied algorithms.

This paper is organized as follows. In Section 2, we present basic concepts used in the rest of the paper. In Section 3, we show the different ways in which binary FC can be generalized into non binary versions. In Section 4, we provide properties and analysis of these generalizations, relating them to the algorithm FC+ [1], an algorithm specially designed to deal with the hidden binary formulation of non binary problems. In Section 5, we provide experimental results of the proposed algorithms. Finally, Section 6 contains some conclusions and directions for further research.

2 Preliminaries

A finite *constraint network* $\mathcal{CN}$ is defined as a set of n *variables* $\mathcal{X} = \{x_1, \ldots, x_n\}$, a current *domain* $D(x_i)$ of possible values for each variable x_i, and a set $\mathcal{C}$ of *constraints* among variables. A constraint c_j on the ordered set of variables $var(c_j) = (x_{j_1}, \ldots, x_{j_{r(j)}})$ specifies the relation $rel(c_j)$ of the *allowed* combinations of values for the variables in $var(c_j)$. $rel(c_j)$ is a subset of $D_0(x_{j_1}) \times \cdots \times D_0(x_{j_{r(j)}})$, where $D_0(x_i)$ is the initial domain of x_i. (The definition of a constraint does not depend on the current domains.) An element of $D_0(x_{j_1}) \times \cdots \times D_0(x_{j_{r(j)}})$ is called a *tuple on* $var(c_j)$. An element of $D(x_{j_1}) \times \cdots \times D(x_{j_{r(j)}})$ is called a *valid* tuple on $var(c_j)$. We introduce the notions of *initial* and *current* domains to explicitly differentiate the initial network, $\mathcal{CN}_0$, from a network $\mathcal{CN}$, obtained at a given node of a tree search after some operations (instantiations and/or filtering). The tuple I_P on the ordered set of *past* variables P represents the sequence of instantiations performed to reach a given node. The set $\mathcal{X} \setminus P$ of the *future* variables is denoted by F. The tuple I_P on P is said to be *consistent* iff for all $c \in \mathcal{C}$ such that $var(c) \subseteq P$, I_P satisfies c.

A value a for variable x is *consistent with* a constraint c iff $x \notin var(c)$, or there exists a valid tuple in $rel(c)$ with value a for x. A variable x is *consistent with* a constraint c iff $D(x)$ is not empty and all its values are consistent with c. A constraint c is *arc consistent* iff for all $x \in var(c)$, x is consistent with c. A set of constraints C is *arc consistent* iff all its constraints are arc consistent [11, 12].

Let $C = \{c_1, \ldots, c_k\}$ be a set of constraints. We will denote by $AC(C)$ the procedure which enforces arc consistency on the set C.[1] Given an arbitrary ordering of constraints $c_1, \ldots, c_k$, we say that AC is applied on each constraint *in one pass* (denoted by $AC(\{c_1\}), \ldots, AC(\{c_k\})$)

[1] Abusing notation, we will also denote by $AC(C)$ the set of values removed by the procedure $AC(C)$.

when AC is executed once on each individual constraint following the constraint ordering. Let σ be a tuple on the set of variables S. The *projection* of σ on a subset S' of S, denoted by $\sigma[S']$, is the restriction of σ to the variables of S'. The projection $c[S']$ of the constraint c on the subset S' of $var(c)$ is a constraint defined by $var(c[S']) = S'$, and $rel(c[S']) = \{t[S']/t \in rel(c)\}$. The *join* of σ and a relation $rel(c)$ on $var(c)$, denoted by $\sigma \bowtie rel(c)$, is the set $\{t/t$ is a tuple on $S \cup var(c)$, and $t[var(c)] \in rel(c)$, and $t[S] = \sigma\}$.

3 From Binary to Non binary FC

FC (from now on, bFC) was defined in [8] for binary constraint networks. They described bFC as an algorithm pursuing this condition at each node,

> *there is no future unit having any of its labels inconsistent with any past unit-label pairs*

where unit stands for variable, and label for value. Values in future domains are removed to achieve this condition, and if a future domain becomes empty, bFC backtracks. This condition is equivalent to require that the set $C^b_{p,f}$, consisting of constraints connecting one past and one future variable, is arc consistent. To do this, it is enough performing arc consistency on the set $C^b_{c,f}$ of constraints involving the current and a future variable, each time a new current variable is assigned (Proposition 2, Section 4.1). In addition, arc consistency on this set can be achieved by computing arc consistency on each constraint in one single pass (Corollary 1, Section 4.1). With this strategy, after assigning the current variable we have,

$$AC(C^b_{p,f}) = AC(C^b_{c,f}) = AC(\{c_1\}), \ldots, AC(\{c_q\}) \quad (\alpha)$$

where $c_i \in C^b_{c,f}$ and $|C^b_{c,f}| = q$. So, bFC works as follows,

bFC: After assigning the current variable, apply arc consistency on each constraint of $C^b_{c,f}$ in one pass. If success (i.e., no empty domain detected), continue with a new variable, otherwise backtrack.

How can the FC strategy be extended for non binary constraints? It seems reasonable to achieve arc consistency (the same level of consistency as bFC) on a set of constraints involving past and future variables. In the binary case, there is only one option for such a set: constraints connecting *one* past variable (the current variable) and *one* future variable. In the non binary case, there are different alternatives. We analyze the following ones,[2]

1. Constraints involving *at least one* past variable and *at least one* future variable;
2. Constraints or constraint projections involving *at least one* past variable and *exactly one* future variable;
3. Constraints involving *at least one* past variable and *exactly one* future variable.

[2] It is worth noting that our analysis is complete. The remaining alternative, constraint or constraint projections involving *at least one* past variable and *at least one* future variable is redundant with option 1, since arc consistency on a constraint implies arc consistency on all its projections.

Considering option (1), we define the set $C^n_{p,f}$ of the constraints involving *at least* one past variable and *at least* one future variable, and the set $C^n_{c,f}$ consisting of constraints involving the current variable and *at least* one future variable (no restriction on the number of past variables). The big difference with the binary case is that, in these sets, we have to deal with partially instantiated constraints, with more than one uninstantiated variable. In this situation, the equivalences of (α) no longer hold for the non binary case. After assigning the current variable we have:

$$AC(C^n_{p,f}) \neq AC(C^n_{c,f}) \neq AC(\{c_1\}), \ldots, AC(\{c_q\}) \quad (\beta)$$

where $c_i \in C^n_{c,f}$ and $|C^n_{c,f}| = q$. Then, we have different alternatives, depending on the set of constraints considered ($C^n_{p,f}$ or $C^n_{c,f}$) and whether arc consistency is achieved on the whole set, or applied on each constraint one by one. They are the following,

nFC5: After assigning the current variable, make the set $C^n_{p,f}$ arc consistent. If success, continue with a new variable, otherwise backtrack.

nFC4: After assigning the current variable, apply arc consistency on each constraint of $C^n_{p,f}$ in one pass. If success, continue with a new variable, otherwise backtrack.

nFC3: After assigning the current variable, make the set $C^n_{c,f}$ arc consistent. If success, continue with a new variable, otherwise backtrack.

nFC2: After assigning the current variable, apply arc consistency on each constraint of $C^n_{c,f}$ in one pass. If success, continue with a new variable, otherwise backtrack.

Regarding options (2) and (3), we define the set $C^n_{p,1}$ of the constraints involving at least one past variable and exactly one future variable, and the set $C^n_{c,1}$ of the constraints involving the current variable and exactly one future variable (no restriction on the number of past variables). Analogously, we define the set $CP^n_{p,1}$ of the constraint projections[3] involving at least one past variable and exactly one future variable, and the set $CP^n_{c,1}$ of the constraint projections involving the current variable and exactly one future variable (no restriction on the number of past variables). Both cases are concerned with the following generalization of (α) (proved in Section 4.1), stating that after assigning the current variable we have,

$$AC(C^n_{p,1}) = AC(C^n_{c,1}) = AC(\{c_1\}), \ldots, AC(\{c_q\}) \quad (\gamma)$$

where $c_i \in C^n_{c,1}$ and $|C^n_{c,1}| = q$. As a result, only one alternative exists for each of the options (2) and (3), and they are respectively the following,

nFC1: ([10]) After assigning the current variable, apply arc consistency on each constraint of $C^n_{c,1} \cup CP^n_{c,1}$ in one pass. If success, continue with a new variable, otherwise backtrack.

nFC0: ([19]) After assigning the current variable, apply arc consistency on each constraint of $C^n_{c,1}$ in one pass. If success, continue with a new variable, otherwise backtrack.

To illustrate the differences between the six presented algorithms, a simple example is presented in Figure 1. It is composed of 6 variables $\{x, y, z, u, v, w\}$, sharing the same domain $\{a, b, c\}$, and subject to three ternary constraints, $c_1(x, y, z)$, $c_2(u, v, w)$ and $c_3(x, y, w)$. After the assignment (x, a), none of the constraints have two instantiated variables. Therefore, nFC0 does no filtering. nFC1 applies arc consistency on the constraint projections of c_1 and c_3 on the subsets $\{x, y\}$, $\{x, z\}$ and $\{x, w\}$, removing c from $D(y)$ and b from $D(w)$. nFC2 applies arc

[3] A constraint projection is computed from the constraint definition which involves initial domains.

$\mathcal{X} = \{x, y, z, u, v, w\}$, every domain is $\{a, b, c\}$

c_1			c_2			c_3		
x	y	z	u	v	w	x	y	w
a	a	a	a	a	a	a	a	a
a	b	c	a	b	b	a	b	c
a	c	b	c	c	c			

Assign	Alg.	Action
(x, a)	nFC0	none
	nFC1	$AC(\{c_1[x,y]\}), AC(\{c_1[x,z]\}), AC(\{c_3[x,y]\}), AC(\{c_3[x,w]\})$
	nFC2	$AC(\{c_1\}), AC(\{c_3\})$
	nFC3	$AC(\{c_1, c_3\})$
	nFC4	$AC(\{c_1\}), AC(\{c_3\})$
	nFC5	$AC(\{c_1, c_3\})$
(u, a)	nFC0	none
	nFC1	$AC(\{c_2[u,v]\}), AC(\{c_2[u,w]\})$
	nFC2	$AC(\{c_2\})$
	nFC3	$AC(\{c_2\})$
	nFC4	$AC(\{c_1\}), AC(\{c_2\}), AC(\{c_3\})$
	nFC5	$AC(\{c_1, c_2, c_3\})$

(x, a)	nFC0	nFC1	nFC2	nFC3	nFC4	nFC5
$D(x)$	a	a	a	a	a	a
$D(y)$	a, b, c	a, b	a, b	a, b	a, b	a, b
$D(z)$	a, b, c	a, b, c	a, b, c	a, c	a, b, c	a, c
$D(u)$	a, b, c	a, b, c	a, b, c	a, b, c	a, b, c	a, b, c
$D(v)$	a, b, c	a, b, c	a, b, c	a, b, c	a, b, c	a, b, c
$D(w)$	a, b, c	a, c	a, c	a, c	a, c	a, c

(u, a)	nFC0	nFC1	nFC2	nFC3	nFC4	nFC5
$D(x)$	a	a	a	a	a	a
$D(y)$	a, b, c	a, b	a, b	a, b	a	a
$D(z)$	a, b, c	a, b, c	a, b, c	a, c	a, c	a
$D(u)$	a	a	a	a	a	a
$D(v)$	a, b, c	a, b	a	a	a	a
$D(w)$	a, b, c	a	a	a	a	a

Figure 1: A simple problem and the filtering caused by the six algorithms, after the assignments (x, a) and (u, a).

consistency on c_1 and later on c_3, pruning the same values as nFC1. Notice that if we consider these constraints in a different order, the filtering will be different. nFC3 achieves arc consistency on the subset $\{c_1, c_3\}$, which causes the filtering of nFC2 plus the removal of b from $D(z)$. Given that x is the first instantiated variable, nFC4 applies arc consistency on the same constraints as nFC2, and it causes the same filtering. For the same reason, nFC5 performs the same filtering as nFC3.

After the assignment (u, a), none of the constraints have two instantiated variables. So, nFC0 does no filtering. nFC1 applies arc consistency on the constraint projections of c_1 on the subsets $\{u, v\}$ and $\{u, w\}$, removing c from $D(v)$ and c from $D(w)$. nFC2 applies arc consistency on c_2, and it removes b and c from $D(v)$ and c from $D(w)$. nFC3 achieves arc consistency on the subset $\{c_2\}$, thus causing the same filtering as nFC2 (differences in $D(z)$ come from the previous assignment). nFC4 applies arc consistency on the constraints c_1, c_2 and c_3, removing b from $D(y)$ and $D(z)$, b and c from $D(v)$ and c from $D(w)$. nFC5 achieves arc consistency on the whole constraint set. It removes b from $D(y)$, c from $D(z)$, b and c from $D(v)$ and c from $D(w)$.

4 Formal Results on nFC

4.1 Properties

In the next results, we prove the equivalences of (γ) used in Section 3.

Proposition 1 *Let c be a constraint such that all its variables but one are instantiated. If c is made arc consistent, it remains arc consistent after achieving arc consistency on any other problem constraint.*

Proof. Let x_j be the only uninstantiated variable of c, and let c_h be another constraint involving x_j. If c_h is made arc consistent after c, this may cause further filtering in $D(x_j)$ but c will remain arc consistent since all remaining values in $D(x_j)$ are already consistent with c. □

Corollary 1 *Let C be a set of constraints such that all their variables but one are instantiated. Achieving arc consistency on C is equivalent to make each of its constraints arc consistent in one pass.*

Proposition 2 *Let P be the ordered set of past variables. Let $C_{p,1}$ be the set of constraints involving at least one past variable and exactly one future variable. If each time a variable of P was assigned, the set $C_{c,1}$ of constraints involving that variable and one future variable was made arc consistent, then the set $C_{p,1}$ is arc consistent.*

Proof. Let us assume that $C_{c,1}$ has been made arc consistent after assigning each variable in P. If $C_{p,1}$ is not arc consistent, this means that there is at least one of its constraints c_h which is not arc consistent. Let x_k be the last assigned variable in $var(c_h)$. c_h has been made arc consistent after x_k assignment. And because of Proposition 1 c_h remained arc consistent. This is in contradiction with the assumption. Therefore, $C_{p,1}$ is arc consistent. □

Regarding the correctness of the proposed algorithms, we have to show that they are sound (they find only solutions), complete (find all solutions) and terminate. All algorithms follow a depth-first strategy with chronological backtracking, so it is clear that all terminate. Then, we have to show soundness and completeness.

Proposition 3 *Any nFCi (i:{0,...,5}) is correct.*

Proof. *Soundness.* We prove that, after achieving the corresponding arc consistency condition, the tuple I_P of past variables reached by any algorithm is consistent. When this tuple includes all variables, we have a solution. The sets of constraints to be made arc consistent by the proposed algorithms all include the set $C_{p,1}$ of nFC0. By Proposition 1, we know that once those constraints are made arc consistent, they remain arc consistent after processing any other constraint. So, proving this result for nFC0 makes it valid for any nFCi algorithm (i:{0,...,5}). If I_P of nFC0 is inconsistent then at least one constraint c involving only variables in P is inconsistent. Let x_i and x_j be the two last assigned variables in $var(c)$, in this order. After assigning x_i, c was in $C_{p,1}$ which was made arc consistent (Proposition 2). Assigning x_j a value inconsistent with c is in contradiction with the assumption that $C_{p,1}$ was arc consistent. So, I_P is consistent.

Completeness. We show completeness for nFC5, proofs for other algorithms are similar. Given a variable ordering, it is clear that nFC5 visits all successors of nodes compatible with such ordering where the set $C^n_{p,f}$ can be made arc consistent. Let us suppose that there is a node solution, I_P, where all variables are past. If x_n is the last variable to be instantiated, the parent node $I_{P\setminus\{x_n\}}$ is a node where $C^n_{p,f}$ can be made arc consistent. By induction, nFC5 visits the node solution I_P. □

At a given node θ, we define the *filtering* caused by an algorithm nFCi, $\Phi(nFCi, \theta)$, as the set of pairs (x, a) where a is a value removed from the future domain $D(x)$ by the corresponding arc consistency condition.

Proposition 4 *At any node θ, these relations hold,*

1. $\Phi(nFC0, \theta) \subseteq \Phi(nFC1, \theta) \subseteq \Phi(nFC2, \theta)$,
2. $\Phi(nFC2, \theta) \subseteq \Phi(nFC3, \theta) \subseteq \Phi(nFC5, \theta)$,
3. $\Phi(nFC2, \theta) \subseteq \Phi(nFC4, \theta) \subseteq \Phi(nFC5, \theta)$.

Proof. Regarding nFC0 and nFC1, the relation is a direct consequence of $C^n_{c,1} \subseteq C^n_{c,1} \cup CP^n_{c,1}$. Regarding nFC1 and nFC2, constraint projections are semantically included in $C^n_{c,f}$. Regarding nFC2 and nFC3, applying arc consistency on each constraint of $C^n_{c,f}$ in one pass is part of the process of achieving arc consistency on the set $C^n_{c,f}$. Regarding nFC3 and nFC5, $C^n_{c,f} \subseteq C^n_{p,f}$. Regarding nFC2 and nFC4, $C^n_{c,f} \subseteq C^n_{p,f}$. Regarding nFC4 and nFC5, applying arc consistency on each constraint of $C^n_{p,f}$ in one pass is part of the process of achieving arc consistency on the set $C^n_{p,f}$. $\square$

Regarding nFC3 and nFC4, their filtering is incomparable as can be seen in the example of Figure 1. (After assigning (x, a), nFC3 filtering is stronger than nFC4 filtering; the opposite occurs after assigning (u, a).) A direct consequence of Proposition 4 involves the set of nodes visited by each algorithm. Defining $nodes(nFCi)$ as the set of nodes visited by nFCi until finding a solution,

Corollary 2 *Given a constraint network with a fixed variable and value ordering, the following relations hold,*

1. $nodes(nFC2) \subseteq nodes(nFC1) \subseteq nodes(nFC0)$,
2. $nodes(nFC5) \subseteq nodes(nFC3) \subseteq nodes(nFC2)$,
3. $nodes(nFC5) \subseteq nodes(nFC4) \subseteq nodes(nFC2)$.

4.2 Complexity Analysis

In this subsection, we give upper bounds to the number of constraint checks the different nFC algorithms perform at one node. First, let us give an upper bound to the number of checks needed to make a future variable x_j consistent with a given constraint c_h. For each value b in $D(x_j)$, we have to find a subtuple σ in $\Pi_{x \in var(c_h) \setminus \{x_j\}} D(x)$ such that σ extended to (x_j, b) is allowed by c_h. So, the number of checks needed to make x_j consistent with c_h is in $O(m \cdot |V|)$, where $V = \Pi_{x \in var(c_h) \setminus \{x_j\}} D(x)$, and m denotes the maximal size of a domain.

In nFC0, a constraint c_h is made arc consistent at a given node iff $var(c_h)$ contains only one future variable x_j. Thus, enforcing arc consistency on c_h is in $O(m)$ since $|V| = 1$. (Domains of past variables are singletons.) Therefore, the number of checks performed by nFC0 at one node is in $O(|C^n_{c,1}| \cdot m)$. For the same reason the number of checks performed by nFC1 at one node is in $O(|C^n_{c,1} \cup CP^n_{c,1}| \cdot m)$, assuming that the constraint projections have been built in a preprocessing phase.

Let r_h be the arity of a constraint c_h. In nFC2 and nFC4, arc consistency is performed on c_h when it has at most $r_h - 1$ future variables. Hence, $|V|$ is bounded above by $m^{r_h - 2}$ for a given constraint c_h, and a given future variable x_j in $var(c_h)$. Thus, making x_j consistent with c_h is bounded above by $m \cdot m^{r_h - 2}$, and enforcing arc consistency on c_h is in $O((r_h - 1) \cdot m^{r_h - 1})$ since there are at most $r_h - 1$ variables to make arc consistent with c_h. If a is the maximal arity of the constraints in $\mathcal{C}$, at any node in the tree, the number of checks performed is in $O(|C^n_{c,f}| \cdot (a-1) \cdot m^{a-1})$ for nFC2, and $O(|C^n_{p,f}| \cdot (a-1) \cdot m^{a-1})$ for nFC4. We can point out that this is an upper bound exponential in the arity of the constraints.

At a given node in the search, nFC3 (resp. nFC5) deals with the same set of constraints as nFC2 (resp. nFC4). The difference comes from the propagations nFC3 (resp. nFC5) performs in order to reach an arc consistent state on $C^n_{c,f}$ (resp. $C^n_{p,f}$), whereas nFC2 (resp. nFC4) performs one pass arc consistency on them. Thus, if we suppose that arc consistency is achieved by an optimal algorithm, such as GAC4 [12] or GAC2001 [2], the upper bound in the number of constraint checks performed by nFC3 (resp. nFC5) at a given node is the same as nFC2 (resp. nFC4) bound. (With an AC3-like algorithm [11], nFC3 and nFC5 have a greater upper bound.)

4.3 Limited nFC

There is a gap in complexity between the number of checks performed at a node by nFC0, and the number of checks performed by nFC2, nFC3, nFC4, or nFC5. Indeed, for a given constraint c_h, it depends on m for the former, and on m^{r_h-1} for the others. We can imagine cases in which the arity of c_h is so large that the effort needed by nFC2, nFC3, nFC4, or nFC5, to make c_h arc consistent has a dramatic effect on their efficiency. The following definition proposes a class of algorithms with bounded effort at each node of the search tree.

Let k be a positive integer. k-nFCi (i:{2,. . . ,5}) is the search algorithm which performs the same type of processing as nFCi at each node, but only on the constraints processed by nFCi that involve at most k future variables.

The number of checks performed by k-nFCi at a given node of the search tree is in $O(e_{ik} \cdot k \cdot m^k)$, where e_{ik} is the maximum number of constraints with at most than k uninstantiated variables, and processed by nFCi, at a given node.

Proposition 5 *At any node θ, we have $\Phi(k\text{-}nFCi,\theta) \subseteq \Phi(nFCi,\theta)$ (i:{2,. . . ,5}).*

Proof. The relation is a direct consequence of the fact that at each node, k-nFCi propagates a subset of the constraints propagated by nFCi (by definition). □

Corollary 3 *Given a constraint network with a fixed variable and value ordering, we have, $nodes(nFCi) \subseteq nodes(k\text{-}nFCi)$.*

Depending on the value of k, k-nFCi can degenerate to already known algorithms:

Proposition 6 *For any i in {2,. . . ,5} the following relations hold,*
0-nFCi is equal to BT,
1-nFCi is equal to nFC0,
(a-1)-nFCi is equal to nFCi if a is the maximum arity over all problem constraints.

This proposition points out why the classical Van Hentenryck's nFC, namely nFC0, is frequently subject to thrashing. It is the version that waits the most in the instantiation process before applying arc consistency on a given constraint. It applies it so late that it cannot detect early incompatibilities between the instantiation and the constraint (when only a part of the variables of the constraint are instantiated).

4.4 FC+ and nFC1

The hidden variable representation is a general method for converting a non binary constraint network into an equivalent binary one [4, 15]. In this representation, the problem has two sets of variables: the set of the *ordinary* variables, those of the original non binary problem, with their original domain of values, plus a set of *hidden* variables, or h-variables. There is a h-variable h_c for each constraint c of the original network, with $rel(c)$ as initial domain (i.e.,

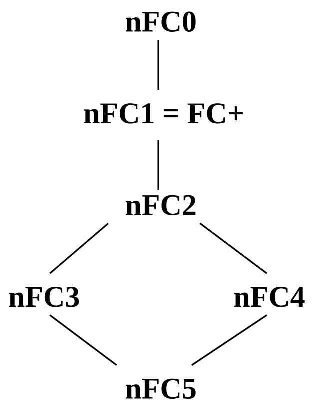

Figure 2: The hierarchy of the algorithms with respect to the number of visited nodes. Two algorithms are connected by an edge if the set of nodes visited by the lower is a subset of the set of nodes visited by the upper.

the tuples allowed by c become the values in $D_0(h_c)$). A h-variable h_c is involved in a binary constraint with each of the ordinary variables x in $var(c)$. Such a constraint allows the set of pairs $\{(v,t)/v \in D_0(x), t \in D_0(h_c), t[x] = v\}$.

FC+ is an algorithm designed to run on the hidden representation [1]. It operates like bFC except that when the domain of a h-variable is pruned, FC+ removes from adjacent ordinary variables those values whose support has been lost. Besides, FC+ never instantiates h-variables. When all its neighboring (ordinary) variables are instantiated, the domain of a h-variable is already reduced to one value. Its assignment is, in a way, implicit. Therefore, there is a direct correspondence between the search space of FC+ and any nFC. The following proposition relates FC+ to the nFC algorithms.

Proposition 7 *Given a constraint network CN with a fixed variable and value ordering, we have $nodes(FC+) = nodes(nFC1)$ if FC+ runs on the hidden representation of CN.*

Proof. Given that FC+ assigns the same variables as nFC1, it is enough to prove that for any ordinary future variable x_j, a value $b \in D(x_j)$ is pruned by FC+ iff it is pruned by nFC1.

Let $x_j \in F$, such that b has been pruned from $D(x_j)$ by nFC1. From the algorithmic description of nFC1, this means that there exists a constraint c with $x_j \in var(c)$ whose projection on $(P \cap var(c)) \cup \{x_j\}$ does not support b. That is, $b \notin I_P \bowtie (rel(c)[(P \cap var(c)) \cup \{x_j\}]))[x_j] = (I_P \bowtie rel(c))[(P \cap var(c)) \cup \{x_j\}][x_j] = (I_P \bowtie rel(c))[x_j] = D(h_c)[x_j]$, if h_c is the variable representing c in the hidden representation. Therefore, b has no support in $D(h_c)$, and FC+ also prunes b.

Analogously, if b is pruned from $D(x_j)$ by FC+, this means that b has lost its support in some hidden variable h_c whose current domain is $D(h_c) = (I_P \bowtie rel(c))$. Hence, the projection of c on $(P \cap var(c)) \cup \{x_j\}$, which is equal to $(I_P \bowtie rel(c))[(P \cap var(c)) \cup \{x_j\}]$, does not support b. So, nFC1 also prunes b. $\square$

Inspired by the hierarchies of algorithms presented in [9], Figure 2 presents the hierarchy resulting from Corollary 2 and Proposition 7.

5 Experimental Results

We have performed some experiments to preliminary assess the relative performance of the proposed algorithms, and to confirm the expectations drawn from the complexity analysis. In our experiments, we have used both random problems and problems from the CSPLib (http://www.csplib.org). On the one hand, random problems permit to relate some characteristics of the algorithms to some varying parameters, such as arity, connectivity or tightness. They also allow to catch the threshold between satisfiability and inconsistency, where hard problems occur. On the other hand, the Schur's lemma (a combinatorial mathematics problem), and the car sequencing problem (a scheduling problem), both from the CSPLib, show the behaviour of the algorithms on more structured problems, with specific constraint semantics.

5.1 Random problems

For random problems, we have extended the well known four-parameter binary model [17, 6] to fixed arity non binary problems as follows. A fixed arity random problem is defined by five parameters $\langle a, n, m, p_1, p_2 \rangle$, where a is the arity of all the constraints in the network, n is the number of variables, m is the cardinality of their domains, p_1 is the problem connectivity as the ratio between existing constraints and the number of possible sets of a variables (the problem has exactly $p_1 \cdot \binom{n}{a}$ constraints), and p_2 is the constraint tightness as the proportion of forbidden value tuples between a constrained variables (the number of forbidden value tuples is exactly $p_2 \cdot m^a$). The constrained variables and their nogoods are randomly selected following a uniform distribution. (We kept only connected problems.) We present results on ternary and quaternary problems because 3-ary are the simplest non binary problems, but 4-ary are the simplest ones on which we have non trivial limited versions of the nFC algorithms.[4]

We performed experiments on the following classes of problems:

(a) $\langle 3, 10, 10, 100/120 = 0.83, p_2 \rangle$,
(b) $\langle 3, 30, 6, 75/4060 = 0.018, p_2 \rangle$,
(c) $\langle 3, 75, 5, 120/67525 = 0.0018, p_2 \rangle$,
(d) $\langle 4, 14, 8, 100/1001 = 10^{-1}, p_2 \rangle$,
(e) $\langle 4, 26, 6, 47/14950 = 10^{-2.5}, p_2 \rangle$,
(f) $\langle 4, 63, 4, 59/595665 = 10^{-4}, p_2 \rangle$.

Regarding connectivity, (a) and (d) are dense classes, while (b) and (e) are relatively sparse, and (c) and (f) very sparse classes. The cross-over point, $\widehat{p_2}$, where 50% of the instances are satisfiable, appears in (a) and (d) at low tightness, in (b) and (d) at medium tightness, and in (c) and (f) at high tightness. (b) and (e) classes were chosen to characterize a situation where nFC0 and nFC2-nFC5 have very close performances.

We solved 50 instances for each set of parameters, using nFC0, nFC1, FC+, nFC2, nFC3, nFC4, and nFC5,[5] with the heuristic *minimum* $\frac{domain\ size}{degree}$ for variable selection [3], and lexicographic value selection. In this paper, we only report results for the values of the tightness where the ratio of satisfiable instances is the closest to 50%. Table 1 contains results on 3-ary problems (classes (a), (b), (c)), and Table 2 contains results on 4-ary problems (classes (d), (e), (f)).

In both Tables 1 and 2, the lines "#nodes" show the mean number of visited nodes to solve each problem class. With no surprise, it is in agreement with Corollary 2, which establishes that nFC0 is the algorithm visiting the most nodes while nFC5 is the one that visits the least nodes. Because of Proposition 7, nFC1 and FC+ visit the same nodes. The new information is about the relation between nFC3 and nFC4, algorithms unordered by Corollary 2. On the six problem classes, nFC4 visits less nodes than nFC3, which means that nFC4 performs more pruning than nFC3.

[4]On 3-ary problems, 2-nFCi is equivalent to nFCi (see Proposition 6).

[5]In nFC2 to nFC5, the algorithm used to apply one pass or "full" arc consistency on a set of constraints is based on GAC2001, an optimal arc consistency algorithm [2].

(a)	$<a=3, n=10, m=10, p_1=0.83, \widehat{p_2}=208/1000>$ (20/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	6,303	6,303	6,303	5,763	5,353	3,547	2,911
#ccks	0.30M	0.84M	55.59M	0.72M	0.75M	0.83M	0.84M
cpu time	0.29	0.57	45.98	0.54	0.91	1.17	1.21
(b)	$<a=3, n=30, m=6, p_1=0.018, \widehat{p_2}=109/216>$ (19/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	61,650	43,455	43,455	31,826	30,003	8,505	6,447
#ccks	0.96M	1.67M	19.98M	1.52M	1.46M	0.82M	0.73M
cpu time	1.39	1.45	13.94	1.39	1.73	1.37	1.29
(c)	$<a=3, n=75, m=5, p_1=0.0018, \widehat{p_2}=76/125>$ (7/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	9,740,904	747,587	747,587	314,330	297,382	56,163	51,116
#ccks	102.69M	19.30M	140.45M	11.64M	11.07M	3.38M	3.16M
cpu time	222.70	23.73	106.44	13.41	15.91	6.99	6.59

Table 1: Results on three classes of 3-ary random problems at the cross-over point. #ccks is in millions and cpu time in seconds. (Mean of 50 instances per class.)

(d)	$<a=4, n=14, m=8, p_1=10^{-1}, \widehat{p_2}=1060/4096>$ (18/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	203,582	203,440	—	178,765	160,600	106,292	86,427
#ccks	7.55M	43.96M	—	28.30M	29.41M	37.24M	38.06M
cpu time	15.32	90.25	>1000	50.02	68.14	101.00	107.00
(e)	$<a=4, n=26, m=6, p_1=10^{-2.5}, \widehat{p_2}=815/1296>$ (18/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	1,214,116	461,999	461,999	309,301	285,301	97,990	65,668
#ccks	14.92M	29.69M	418.67M	22.44M	21.74M	19.12M	17.53M
cpu time	33.38	57.19	277.84	30.32	34.97	36.34	34.41
(f)	$<a=4, n=63, m=4, p_1=10^{-4}, \widehat{p_2}=194/256>$ (20/50 sat)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	—	189,647	189,647	112,900	109,859	23,396	17,767
#ccks	—	7.25M	30.82M	5.80M	5.72M	2.88M	2.60M
cpu time	>1000	15.94	21.67	8.09	9.47	5.39	5.24

Table 2: Results on three classes of 4-ary random problems at the cross-over point. #ccks is in millions and cpu time in seconds. (Mean of 50 instances per class.)

The lines "#ccks" and "cpu time" show the average computational effort[6] (as mean number of constraint checks and mean CPU time) required. We observe that, for problems with loose constraints (cross-over point at low tightness —classes (a) and (d)—) the winner is nFC0, the algorithm that performs the simplest look ahead. For these classes of problems, sophisticated forms of look ahead do not pay-off: the proposed algorithms nFC1 to nFC5 are 1.9 to 7 times slower than nFC0. And the difference is greater on 4-ary problems where the useless look ahead is even more expensive than on 3-ary problems. FC+ on the hidden representation is orders of magnitude slower. For problems with the cross-over point at medium tightness —classes (b) and (e)—, no single algorithm clearly outperforms the others. nFC0, nFC2, nFC3, nFC4, and nFC5

[6]This effort includes the preprocessing phase for nFC1 and the conversion into the hidden representation for FC+.

(a)	$<a=4, n=14, m=8, p_1=10^{-1}, \widehat{p_2}=1060/4096>$ (18/50 sat)							
	2-nFC2	*L/P*	2-nFC3	*L/P*	2-nFC4	*L/P*	2-nFC5	*L/P*
#nodes	179,485	*1.00*	163,410	*1.02*	109,298	*1.03*	88,264	*1.02*
#ccks	22.46M	*0.79*	22.67M	*0.77*	22.84M	*0.61*	22.27M	*0.59*
cpu time	38.61	*0.77*	52.52	*0.77*	61.25	*0.61*	61.54	*0.58*
(c)	$<a=4, n=63, m=4, p_1=10^{-4}, \widehat{p_2}=194/256>$ (20/50 sat)							
	2-nFC2	*L/P*	2-nFC3	*L/P*	2-nFC4	*L/P*	2-nFC5	*L/P*
#nodes	172,825	*1.53*	171,316	*1.56*	137,919	*5.89*	136,648	*7.69*
#ccks	4.49M	*0.77*	4.46M	*0.78*	3.81M	*1.32*	3.78M	*1.46*
cpu time	7.78	*0.96*	9.26	*0.98*	8.49	*1.58*	8.83	*1.69*

Table 3: Results of the 2-nFCi limited versions on the two extreme classes of 4-ary random problems at the cross-over point. #ccks is in millions and cpu time in seconds. (Mean of 50 instances per class.)

are very close.[7] nFC1, close to the others on the 3-ary class (b), is twice slower on the 4-ary class (e). The bad behaviour of FC+ is confirmed. For problems with the cross-over point located at high tightness —classes (c) and (f)—, the proposed algorithms nFC1 to nFC5 clearly outperform nFC0. Even FC+ performs much better than nFC0. The winner is nFC5, the algorithm which performs the greatest effort per node, and causes the highest filtering. It is orders of magnitude faster than nFC0.

On these six classes, it seems that nFC2 is the more stable algorithm. If we average 3-ary and 4-ary classes, it is second (behind nFC0) on loose constraints, third (behind nFC4 and nFC5) on tight constraints, and the winner on medium constraints. nFC1, which is among the best choices on 3-ary constraints is not as good on the 4-ary problems. The main reason is probably that projecting a 3-ary constraint on all the subsets of variables creates only three binary constraints while projecting a 4-ary constraint creates four 3-ary constraints and six binary ones. Even if it is a way to avoid the complexity of arc consistency, which is growing with the arity on stronger versions (nFC2 to nFC5 —see Section 4.2), these projections will become loose when the projected constraint is not very tight, generating almost no pruning. Consequently, the behaviour of nFC1 is expected to decay with growing arity. Considering FC+, it has the worst performance for loose and medium constraints, and it is the second worst (after nFC0) for tight constraints. Any of the proposed algorithms outperforms FC+ in the six problem classes.[8]

We can also point out some other noteworthy phenomena that are not visible in the tables reported here. First, on the problem classes presented there, nFC0 is the only algorithm that encountered exceptionally hard problems, located in the satisfiable region of the $\langle 3, 75, 5, 0.0018, p_2\rangle$ and $\langle 4, 63, 4, 10^{-4}, p_2\rangle$ classes. Second, when the heuristic *minimum domain size* for variable selection is used instead of *minimum* $\frac{\textit{domain size}}{\textit{degree}}$, nFC0 becomes more frequently subject to thrashing, even on problem sizes remaining very easy for the algorithms nFC1 to nFC5.

Limited versions of nFC2–nFC5

Table 3 presents the results for the limited versions of the nFC2–nFC5 algorithms on the 4-ary classes. On 4-ary constraints, the only non trivial limited version is with $k = 2$. We present only the two extreme cases (loose and tight constraints). Column L/P gives the ratio limited version / plain version. According to Corollary 3, the number of nodes visited by any 2-nFCi

[7] As already mentioned, we chose to report these classes because of this particularity.

[8] These results do not contradict some already published works showing a good behaviour of FC+ on cross-word puzzles. On cross-words, indeed, constraints are very tight, and they are given in extension.

is always greater than in the plain version nFCi. Not surprisingly, the difference is greater on tight constraints. This means that more propagations are missed by the limited version on tight constraints. Regarding the computational effort (number of constraint checks and cpu time), it is significantly less for limited versions on loose constraints since the number of nodes are close to plain versions and the number of possible constraint checks on a constraint is bounded above by m^2 instead of m^3 for plain versions. For tight constraints, things are less straightforward. In 2-nFC4 (resp. 2-nFC5), the great increase in number of nodes was too high to be outweighed by the constraint checks saved at each node. Hence, nFC4 (resp. nFC5) outperforms 2-nFC4 (resp. 2-nFC5) both in constraint checks and cpu time. For 2-nFC2 and 2-nFC3, however, the increase in number of nodes was small compared to plain versions. Thus, the constraint checks saved at each node permit to outperform slightly the plain versions in cpu time.

5.2 Schur's lemma

We also performed experiments on several combinatorial mathematics problems of the CSPLib. In this section, we present results we obtained on the Schur's lemma, and on a modified version of this problem. The Schur's lemma consists in putting n balls labelled from 1 to n into 3 boxes, such that three balls labelled x, y, and z are not put in the same box if $x + y = z$. We encoded this problem as a CSP in which balls are variables, boxes are their values, and a constraint $notequal(x, y, z)$ is put on a triple of balls x, y, z when $x + y = z$. This constraint forbids them to take all the same value/box. As in random problems, we used GAC2001 to apply arc consistency [2]. The greatest number of balls that can be put in 3 boxes is 23. The only non trivial instance is the proof of optimality, i.e., proving that the problem with 24 balls and 3 boxes does not have solution. Results for this problem are reported at the top of Table 4. Because the constraint *notequal* is very loose, we have a pattern which is close to what we obtained on classes (a) and (d) of random problems: weaker look ahead produces better performance.

To see what happens when tightness increases, we changed slightly the definition of the problem, replacing the *notequal* constraint by a $alldiff$ when three balls labelled x, y, z, verify $x + y = z$. $Alldiff$ is loose, but not as loose as the *notequal*. With this new specification, the problem with 3 boxes becomes trivial (4 is the optimal number of balls). We increased the number of boxes until non trivial cases occur. The optimal number of balls for 9 boxes is 11. In Table 4 (bottom), we report the proof of optimality, i.e., proving that the problem with 12 balls and 9 boxes does not have solution. The effect of having constraints slightly tighter than in the previous case appears clearly. The look ahead performed by nFC0 is too weak. The one performed by nFC4/nFC5 is still too much. The good compromise in this case is nFC1/nFC2/nFC3, which visit the same nodes. nFC2 is the winner in cpu time since it has the simplest behaviour among these three algorithms. We can expect that on problems with even tighter constraints, nFC4/nFC5 would have been the winners.

We have to bear in mind that all these results were obtained while the constraints *notequal* and $alldiff$ were made arc consistent with a generic algorithm, although there exist specific algorithms using their semantics. As we will see in the Subsection 5.3, the use of specific algorithms to make arc consistent the constraints with a specific semantics can affect the results.

5.3 Car sequencing

To illustrate the behaviour of the nFCs on a real problem, we choose the car sequencing problem, a scheduling problem from the CSPLib. In this problem, a number of cars are to be produced. They are not identical because different options can be required as variants on the basic model. The problem consists in scheduling the cars on a assembly line so that the options can be installed in different stations along the line. A station is designed to handle at most a certain proportion

	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
	Schur's lemma ($x+y=z \to notequal(x,y,z)$, 24 balls, 3 boxes)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	9,840	9,840	9,840	9,816	9,816	4,860	4,404
#ccks	0.19M	0.53M	2.77M	0.30M	0.30M	0.35M	0.33M
cpu time	0.33	0.90	1.87	0.54	0.69	0.99	0.95
	Modified Schur ($x+y=z \to alldiff(x,y,z)$, 12 balls, 9 boxes)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	1,546,362	986,409	986,409	986,409	986,409	623,529	623,529
#ccks	24.33M	15.53M	293.08M	8.41M	8.41M	12.41M	12.41M
cpu time	36.31	25.78	215.68	17.47	21.78	32.69	39.14

Table 4: Results on the Schur's lemma and its modified version. #ccks is in millions and cpu time in seconds.

of the cars passing along the assembly line. For instance, if a particular station, installing option k, can only cope with at most one third of the cars passing along the line (i.e., capacity 1/3), the sequence of cars must be built so that at most 1 car in any 3 consecutive cars requires that option. In the data files tested, it is assumed that there are five options with capacities 1/2, 2/3, 1/3, 2/5, and 1/5 respectively.

We encoded this problem as a CSP in which slots in the sequence are variables, cars to be built are their values. For each option k installed in a station of capacity p_k/q_k, a constraint c of arity q_k allowing only p_k cars with option k among the variables in $var(c)$ is posted on any q_k consecutive slots. A clique of inequalities ensures that a car is not assigned twice. Finally, we added a redundant global constraint ensuring that there remain enough free slots uninstantiated to assign the remaining cars with a given option. (For instance, if at a given node there only remain the 17 last slots uninstantiated, and if 5 of the remaining cars need the option installed in the station of capacity 1/5, we know that we cannot reach a solution from this node —only 4 such cars can be placed.) Since we wanted to see the behaviour of the nFCs on "close to reality" conditions, we developed specific algorithms for the different kinds of constraints needed in our simple encoding. The binary inequalities are propagated (i.e., made arc consistent) with the technique described in [13]. The constraint "at most p among q consecutive cars" is propagated with an algorithm pruning the domains of the involved variables in one turn, avoiding the search for support in the Cartesian product of the domains. The redundant global constraint is also propagated by a specific algorithm. Finally, following Smith's recommendation [18], we used the lexicographic variable ordering in all the algorithms.

The data files reported in the CSPLib only contain very difficult instances. Those for which a solution/inconsistency is known have been solved with sophisticated algorithms tuned to deal with the features of these problems (global sequencing constraints [14], symmetries, etc.). Hence, we decreased the sizes of these problems to obtain smaller running times.

Results for two instances are reported in Table 5. (All instances tested gave similar results.) The lines "#ccks" count the number of times a call to the propagation algorithm of a constraint is performed. (The classical notion of "constraint check" no longer exists in specific algorithms.) We observe that propagating a constraint as soon as one of its variables is instantiated greatly pays off since nFC0 is much slower than the others. It finished its search in less than 1 hour only on instances on which the other algorithms needed less than 1 second. (Limited versions 2-nFC2 to 2-nFC5 were almost as bad as nFC0 in cpu time.) As opposed to nFC0, performance of nFC1 to nFC5 is in the same order of magnitude, nFC2 being the winner, followed by nFC1. This can be explained in part by the fact that all the non binary constraints[9] involve consecutive

[9] namely, the capacity constraints, and the redundant global constraint.

	Number of cars: 43 (satisfiable)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	—	4,256,089	—	4,256,089	4,097,904	4,256,089	4,097,904
#ccks	—	28.54M	—	30.64M	32.99M	30.64M	32.99M
cpu time	>1h	77.02	—	67.97	92.92	100.37	101.84
	Number of cars: 38 (inconsistent)						
	nFC0	nFC1	FC+	nFC2	nFC3	nFC4	nFC5
#nodes	—	6,595,664	—	6,595,664	6,484,440	6,595,664	6,484,440
#ccks	—	34.50M	—	39.28M	42.69M	39.28M	42.69M
cpu time	>1h	112.70	—	107.33	141.79	150.64	152.84

Table 5: Results on the car sequencing problem. #ccks is in millions and cpu time in seconds.

variables. Together with the lexicographic variable ordering, this implies that $C^n_{p,f}$ and $C^n_{c,f}$ always contain the same non binary constraints. (The only binary constraints are those of the clique of inequalities on which all nFCs collapse —see Section 3.) As a result, nFC2 (resp. nFC3) and nFC4 (resp. nFC5) explore the same search tree. The cpu time difference between nFC2 and nFC4, or between nFC3 and nFC5, is due to the data structures handled by nFC4 and nFC5. A last thing to notice is that "full" arc consistency on $C^n_{c,f}$ (nFC3) or $C^n_{p,f}$ (nFC5) does not pay off w.r.t. one pass arc consistency of nFC2 and nFC4. Regarding nFC1, despite the big number of constraints generated, it remains competitive. This is a very particular case where all the projections are constraints with the same semantics as the projected one, so that the specific propagation algorithm can be used. FC+ could not be run because of the huge space needed for the domains of the hidden variables.

Finally, in order to more deeply assess the effect of using specific algorithms to propagate the constraints, we solved a (satisfiable) car sequencing problem with 16 cars where we replaced the specific propagation algorithms by the generic GAC2001. nFC0 becomes the winner in 18.09 sec., whilst nFC2 and nFC5 need 51.60 sec. and 68.89 sec. respectively.[10] This is consistent with the results obtained on random problems. In our encoding, indeed, some arities are large, and the tightest constraint (the 5-ary capacity constraint "1/5") is relatively loose: its tightness is never greater than 0.4 on satisfiable instances.

5.4 Discussion

In this Section, we briefly synthesize the lessons that can be drawn from the experiments we performed. First, we have to keep in mind that stronger look ahead pays off only when it is outweighed by domain pruning, i.e., by a reduction in the number of nodes. The first consequence of this evidence is straightforward on random problems, for which we saw that problems with loose constraints were better solved by nFC0, while on problems with tight constraints, nFC5 was the winner. On medium tightnesses, all the nFCs exhibited a similar performance. On the Schur's lemma and its modified version, we found the same behaviour: nFC0 wins on the very loose Schur's lemma while the best performance shifts to nFC2 with the slightly tighter modified version.

In addition to the tightness of the constraints, the arity is another parameter that affects performance. Indeed, as pointed out in Section 4.2, the cost of applying arc consistency on a constraint grows with the arity. This is illustrated by classes (b) and (e) in the random experiments. These classes represent the case where nFC0 and the other nFCs have the closest

[10] Using the specific propagation algorithms, this instance is solved in 0.00 sec. by nFC1-nFC5, and in 0.03 by nFC0.

performance at the threshold. For 3-ary problems it is for $\widehat{p_2} = 109/216 \approx 0.5$ while for 4-ary problems it is for $\widehat{p_2} = 815/1296 \approx 0.63$. We can expect that the greater the arity will be, the greater the tightness will have to be to find problems where nFC0 is the worst choice.

While the picture seems to be clear when arc consistency is enforced by a generic algorithm, this is much less definite on problems where specific propagation algorithms are used. Indeed, if we look at the results on the car sequencing problem, we did not expect such a big advantage for nFC1-nFC5 compared to nFC0. This problem has loose constraints with large arities, which seemed to be more in favour of nFC0. The key point is that we implemented specific algorithms for which the cost of propagation is much lower than for a generic algorithm. Thus, algorithms with stronger look ahead benefit from more domain reductions at reduced cost. Therefore, on problems with constraints for which specific propagation algorithms are available, it is not sufficient to know the arity and the tightness of the constraints to predict the right level of look ahead. It is indeed related to the trade-off between benefit and cost of constraint propagation, which completely depends on the given constraint and its propagation algorithm.

Concerning specific constraint propagation algorithms, we can notice that nFC1 has no chance to be competitive if the semantics of the constraint is lost on its projections, preventing the use of the specific algorithm. The capacity constraint p/q of the car sequencing was a favorable case since its projection on $q-k$ variables preserves the semantics: it is the capacity constraint $p/(q-k)$. (When $p \geq q-k$, the constraint is still a capacity constraint, but equal to the universal constraint.) On the contrary, a constraint such as $x+y+z=t$, which has a simple propagation algorithm based on arithmetic properties, loses its semantics when projected on subsets of the variables.

Finally, can we decide between the one pass behaviour of nFC2-nFC4 and the "full" arc consistency behaviour of nFC3-nFC5? The answer is not obvious. Though nFC3 is slower than its one pass equivalent –nFC2– on all our experiments, nFC4 and nFC5 are much harder to separate.

6 Summary and Conclusion

We presented several possible generalizations of the FC algorithm to non binary constraint networks. We studied their properties, and analyzed their complexities. We also compared these non binary algorithms to the binary FC+ algorithm, which runs on the hidden conversion of non binary networks. We provided empirical results on the relative performances of these algorithms. Their performances greatly depend on the tightness and arity of the constraints. This fits the already known trade-off between the benefits of early pruning caused by constraint propagation, and the effort it requires. But the use of the semantics of the constraints can also affect performance. When a specific algorithm is used to deal with a specific constraint, the trade-off between benefit and cost of constraint propagation has shown a slide to the advantage of versions with higher look ahead.[11] An ultimate goal could be to exhibit a criterion under which to decide when a constraint should be processed by the nFC0 principle, and when it should be propagated with a more pruningful mechanism. Such a criterion might be learned on some instances of problems, such as in [5], where variable ordering heuristics are learned by experience. The result would be a mixed algorithm, taking the best of each technique.

[11] Existing constraint solvers often perform some form of look ahead (arc consistency or weaker) on all the constraints (i.e., a kind of MAC algorithm).

References

[1] F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings AAAI'98*, pages 311–318, Madison WI, 1998.

[2] C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI'01*, pages 309–315, Seattle WA, 2001.

[3] C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings CP'96*, pages 61–75, Cambridge MA, 1996.

[4] R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings AAAI'90*, pages 556–562, Boston MA, 1990.

[5] S.L. Epstein and E.C. Freuder. Collaborative learning for constraint solving. In *Proceedings CP'01*, pages 46–60, Paphos, Cyprus, 2001.

[6] D. Frost, C. Bessière, R. Dechter, and J.C. Régin. Random uniform csp generators. URL: http://www.ics.uci.edu/~dfrost/csp/generator.html, 1996.

[7] S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, October 1965.

[8] R.M. Haralick and G.L. Elliott. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[9] G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[10] J. Larrosa and P. Meseguer. Adding constraint projections in n-ary csp. In J.C. Régin and W. Nuijtens, editors, *Proceedings of the ECAI'98 workshop on non-binary constraints*, pages 41–48, Brighton, UK, 1998.

[11] A.K. Mackworth. On reading sketch maps. In *Proceedings IJCAI'77*, pages 598–606, Cambridge MA, 1977.

[12] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI'88*, pages 651–656, Munchen, FRG, 1988.

[13] R. Mohr and G. Masini. Running efficiently arc consistency. In G. Ferraté et al., editor, *Syntactic and Structural Pattern Recognition*, pages 217–231. Springer-Verlag, Berlin, 1988.

[14] J.C. Régin and J.F. Puget. A filtering algorithm for global sequencing constraints. In *Proceedings CP'97*, pages 32–46, Linz, Austria, 1997.

[15] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings ECAI'90*, pages 550–556, Stockholm, Sweden, 1990.

[16] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings PPCP'94*, Seattle WA, 1994.

[17] B. Smith. Phase transition and the mushy region in constraint satisfaction problems. In *Proceedings ECAI'94*, pages 100–104, Amsterdam, The Netherlands, 1994.

[18] B. Smith. Succeed-first or fail-first: a case study in variable and value ordering. In *Proceedings ILOG Solver and ILOG Scheduler International Users' Conference*, Paris, France, 1996.

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