On Forward Checking for Non-binary Constraint Satisfaction [⋆]

Christian Bessière[1], Pedro Meseguer[2], Eugene C. Freuder[3], and Javier Larrosa[4]

[1] LIRMM-CNRS, 161 rue Ada, 34392 Montpellier, France
bessiere@lirmm.fr,
[2] IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
pedro@iiia.csic.es,
[3] University of New Hampshire, Durham, NH 03824, USA
ecf@cs.unh.edu,
[4] 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 for the next years, 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 version, proposed by Van Hentenryck, is only one of these generalizations.

1 Introduction

Up to now, most of the research done in constraint satisfaction assumes 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 [8]. 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 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 formulations.

[⋆] P. Meseguer and J. Larrosa were supported by an Integrated Action financed by the Generalitat de Catalunya, and by the Spanish CICYT project TIC96-0721-C02-02, C. Bessière was supported by an "action CNRS/NSF" under Grant no. 0690, and E.C. Freuder by the National Science Foundation under Grant No. IRI-9504316.

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, like backtracking or MAC, this extension presents no conceptual difficulties: their binary definitions allow only one possible non-binary generalization. For other algorithms, like forward checking (FC), 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 performance of the proposed algorithms with respect to other non-binary FC algorithms previously presented.

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]. In Section 5, we provide experimental results of the proposed algorithms on random ternary problems. 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* $\mathcal{X} = \{x_1, \dots, x_n\}$, 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 $var(c_j) = (x_{j_1}, \dots, x_{j_{r(j)}})$ 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 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 [6, 7].

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 $AC(\{c_1\}), \ldots, AC(\{c_k\}))$ when AC is executed once on each individual constraint following the constraint ordering. We will denote by $\Pi_{x \in S} D(x)$ the Cartesian product of the domains of the variables in S. 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 [4] 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 $C^b_{p,f}$, composed 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)

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

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,

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.

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 $C^n_{c,f}$ composed of constraints involving the current variable and *at least* one future variable. 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, that is,

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

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. Analogously, we define the set $CP^n_{p,1}$ of the constraint projections[2] 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. Both cases are concerned with the following generalization of (α) (proved in Section 4.1), stating that after assigning the current variable we have,

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

[2] 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

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

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 the following,

nFC1: ([5]) 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: ([10]) 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 proposed 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 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 $\tilde{c}$ be another constraint sharing an uninstantiated variable x_j with c. If $\tilde{c}$ 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. $\square$

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, 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 which is not arc consistent. Let x_k be the last assigned variable in $var(c)$. Because of Proposition 1 this is in contradiction with the assumption that it was made arc consistent after assigning x_k. 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. Assigning x_j a value inconsistent with c is in contradiction with the assumption that $C_{p,1}$ was made 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 k, we define the *filtering* caused by an algorithm nFCi, $\Phi(nFCi, k)$, 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 k these relations hold,*

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

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 $C^n_{p,f}$. □

Regarding nFC3 and nFC4, their filterings are incomparable as can be seen in 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 this result 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 variable x_j consistent with a given constraint c. For each value b in $D(x_j)$, we have to find a subtuple σ in $\Pi_{x \in var(c) \setminus \{x_j\}} D(x)$ such that σ extended to (x_j, b) is allowed by c. So, the number of checks needed to make x_j consistent with c is in $O(d \cdot |V|)$, where $V = \Pi_{x \in var(c) \setminus \{x_j\}} D(x)$, and d denotes the maximal size of a domain.

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

In nFC2 and nFC4, $|V|$ is bounded above by $d^{|var(c) \cap F|-1}$ for a given constraint c, and a given future variable x_j in $var(c)$. Thus, making x_j consistent with c is bounded above by $d \cdot d^{|var(c) \cap F|-1}$, and enforcing arc consistency on c is in $O(|var(c) \cap F| \cdot d^{|var(c) \cap F|})$ since there are $|var(c) \cap F|$ variables to make arc consistent with c. So, the number of checks performed at one node is in $O(|C^n_{c,f}| \cdot |var(c) \cap F| \cdot d^{|var(c) \cap F|})$ for nFC2, and in $O(|C^n_{p,f}| \cdot |var(c) \cap F| \cdot d^{|var(c) \cap F|})$ for nFC4.

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 [7] or GAC-schema [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 [6], nFC3 and nFC5 have a greater upper bound.)

4.3 FC+ and nFC1

The hidden variable representation is a general method for converting a non-binary constraint network into an equivalent binary one [3, 8]. In this representation, the problem has two sets of variables: the set of the *ordinary* variables, those of the original non-binary network, 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., 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 $\{(a,t)/a \in D_0(x), t \in D_0(h_c), t[x] = a\}$.

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 shows that FC+ is equivalent to nFC1.

Proposition 5 *Given any non-binary constraint network, nFC1 visits exactly the same nodes as FC+ applied to the hidden representation, provided that both algorithms use the same variable and value orderings.*

Proof. Because of the algorithmic description of FC+, we know that h-variables may only have their domain pruned by the bFC look ahead. An arbitrary h-domain, $D(h_c)$, may only be pruned if $P \cap var(c) \neq \emptyset$, and $D(h_c) = \{(I_P \bowtie rel(c))[var(c)]\}$. Domains of ordinary variables may only be pruned by the extra look ahead of FC+. At a given node, value b for a future variable x_j belongs to $D(x_j)$ iff it still has support from all its adjacent h-variables that may have been pruned. That is, $\forall x_j \in F$, $b \in D(x_j)$ iff $\forall c$ s.t. $var(c) \cap P \neq \emptyset$ and $x_j \in var(c)$, $b \in D(h_c)[x_j] = ((I_P \bowtie rel(c))[var(c)])[x_j] = (I_P \bowtie rel(c))[x_j]$. Now, because of its definition, we know that at a given node nFC1 ensures that value b for a future variable x_j belongs to $D(x_j)$ iff it is consistent with the projections $rel(c)[var(c) \cap P \cup \{x_j\}]$ for all the constraints c such that $var(c) \cap P \neq \emptyset$ and $x_j \in var(c)$. That is, $\forall x_j \in F$, $b \in D(x_j)$ iff $\forall c$ s.t. $var(c) \cap P \neq \emptyset$ and $x_j \in var(c)$, $b \in (I_P \bowtie (rel(c)[var(c) \cap P \cup \{x_j\}]))[x_j] = (I_P \bowtie rel(c))[var(c) \cap P \cup \{x_j\}][x_j] = (I_P \bowtie rel(c))[x_j]$, which is exactly what we found for FC+. Since FC+ and nFC1 prune exactly the same values on the ordinary variables at a given node, we have the proof. □

5 Experimental Results

We have performed some experiments to preliminary assess the potential of the proposed algorithms. In our experiments we have used random problems extending the well known four-parameter binary model [9] to ternary problems as follows. A ternary random problem is defined by four parameters $\langle n, m, p_1, p_2\rangle$ where 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 maximum number of possible constraints (the problem has exactly $p_1 n(n-1)(n-2)/6$ constraints), and p_2 is the constraint tightness as the proportion of forbidden value triplets between three constrained variables (the number of forbidden value triplets is exactly $T = p_2 m^3$). The constrained variables and their nogoods are randomly selected following a uniform distribution.

We performed experiments on the following classes of problems:
(a) $\langle 10, 10, 100/120, p_2\rangle$,
(b) $\langle 30, 6, 75/4060, p_2\rangle$,
(c) $\langle 75, 5, 120/67525, p_2\rangle$.
Regarding connectivity, (a) is a dense class while (b) and (c) are sparse classes. The complexity peak location appears in (a) at low tightness, in (b) at medium tightness, and in (c) at high tightness.

We solved 50 instances for each set of parameters, using nFC0, nFC1, FC+, nFC2, nFC3, nFC4, and nFC5,[3] with the heuristic *minimum* $\frac{domain\ size}{degree}$ for variable selection and lexicographic value selection. Figure 2 shows the mean number of visited nodes to solve each problem class. Only the complexity peak region is shown. 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 5, nFC1 and FC+ visit the same nodes. The new information is about the relation between nFC3 and nFC4, algorithms unordered by Corollary 2. Consistently in the three problem classes, nFC4 visits less nodes than nFC3, which implies that nFC4 performs more pruning than nFC3.

Figures 3 and 4 show the average computational effort[4] (as mean number of consistency checks and mean CPU time) required. We observe that, for easy problems (with peak at low tightness) the winner is nFC0, the algorithm that performs the simplest lookahead. For this class of problems, sophisticated forms of lookahead do not pay-off: the proposed algorithms nFC1 to nFC5 are 1.8 to 4.8 times slower than nFC0 at the peak. FC+ on the hidden representation is orders of magnitude slower. For problems with the peak at medium tightness, no single algorithm clearly outperforms the others. nFC0, nFC1, nFC2, and nFC5 are very close. nFC3 and nFC4 are slightly worse. The bad behavior of FC+ is confirmed. For difficult problems with the peak located at high tightness, the proposed

[3] In nFC3 and nFC5, the technique used to achieve arc consistency on a set of constraints is a brute force non optimal GAC3-based algorithm.

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

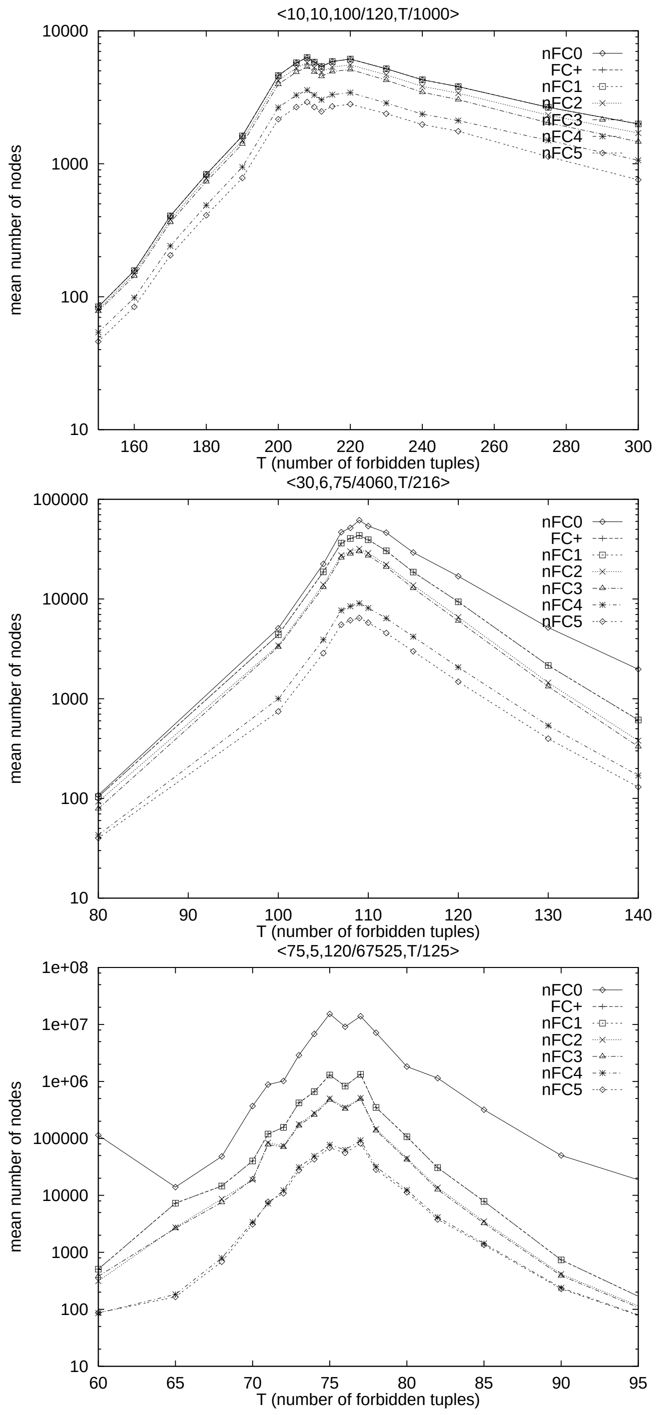


Fig. 2. Average number of visited nodes for three classes of ternary random problems.

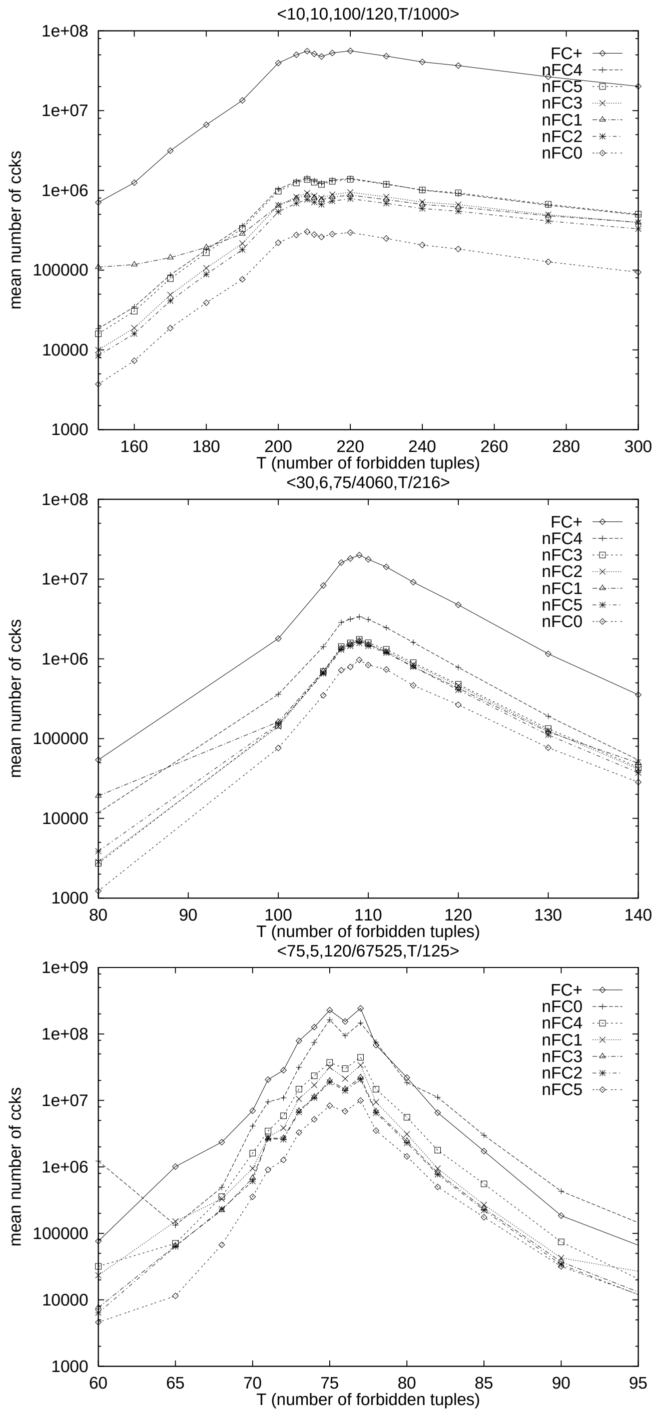
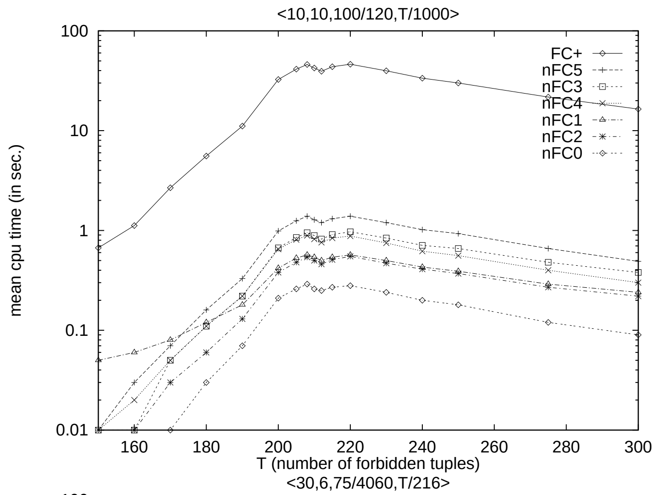
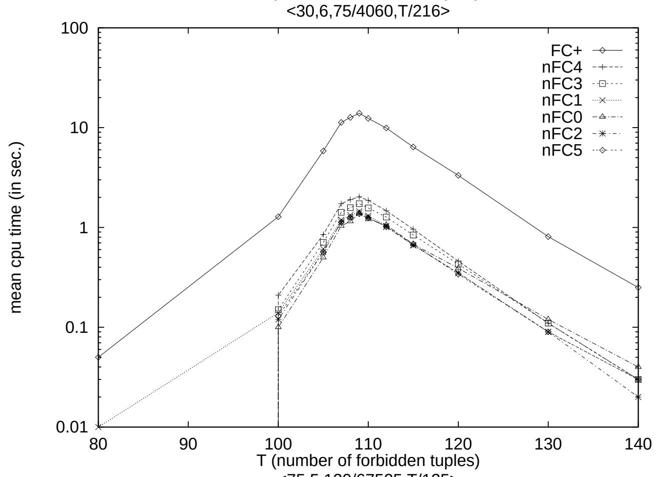


Fig. 3. Average number of checks for three classes of ternary random problems.





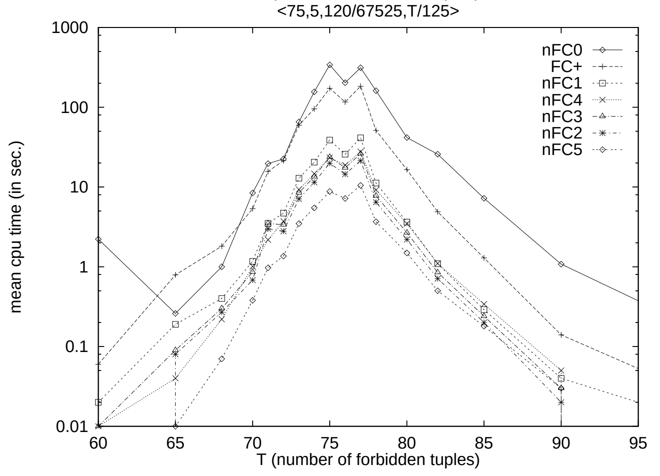


Fig. 4. Average cpu time for three classes of ternary random problems.

algorithms nFC1 to nFC5 clearly outperform nFC0. Even FC+ performs better than nFC0. The winner is nFC5, the algorithm which performs the greatest effort per node, and causes the highest filtering. It is 32 times faster than nFC0 at the peak. The good behaviour of nFC4 in number of visited nodes has no translation in performance. However, nFC2 visiting more nodes than nFC4 is the second best algorithm in performance in the three problem classes. 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 three problem classes.

We can also point out some other noteworthy phenomena that are not visible in the figures reported here. First, on the problem classes presented there, nFC0 is the only algorithm that encountered an exceptionally hard problem, located in the satisfiable region of the $\langle 75, 5, 120/67525, p_2 \rangle$ class. Second, when the heuristic *minimum domain size* for variable selection is used instead of *minimum* $\frac{domain\ size}{degree}$, nFC0 becomes more frequently subject to thrashing, even on problem sizes remaining very easy for the algorithms nFC1 to nFC5.

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 initial empirical results on the performance of these algorithms. From them, we conclude that the proposed algorithms outperform existing approaches on sparse problems with tight constraints. On easier problems, the benefits caused by their early lookahead do not outweigh the propagation effort. This unsurprising conclusion fits the already known trade-off between benefits and costs in constraint satisfaction. Nevertheless, more empirical studies are needed to substantiate which of these algorithms are promising, and on which constraints they perform better. 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. The result would be a mixed algorithm, taking the best of both techniques.

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. Arc consistency for general constraint networks: preliminary results. In *Proceedings IJCAI'97*, pages 398–404, Nagoya, Japan, 1997.
3. R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings AAAI'90*, pages 556–562, Boston MA, 1990.
4. R.M. Haralick and G.L. Elliot. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

5. 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.
6. A.K. Mackworth. On reading sketch maps. In *Proceedings IJCAI'77*, pages 598–606, Cambridge MA, 1977.
7. R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI'88*, pages 651–656, Munchen, FRG, 1988.
8. F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings ECAI'90*, pages 550–556, Stockholm, Sweden, 1990.
9. B. Smith. Phase transition and the mushy region in constraint satisfaction problems. In *Proceedings ECAI'94*, pages 100–104, Amsterdam, The Netherlands, 1994.
10. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, MA, 1989.