Improving Forward Checking with Delayed Evaluation

Camilo Rueda and Frank D. Valencia
Grupo AVISPA
Universidad Javeriana de Cali
Research Team AVISPA
Pontificia Universidad Javeriana Cali
Calle 18 Num. 118-250 Via a Pance
Cali, Colombia
Phone:(57) 2-5552175
Fax:(570 2-5552823
[crueda,fvalenci]@atlas.univalle.edu.co

Abstract

Several algorithms for solving constraint satisfaction problems (CSP) have been proposed in recent years. One of the most successful approaches involves using arc consistency as a pre-processing step followed by a form of backtrack search called forward checking (FC). Efficient algorithms for arc consistency based on the notion of first support have been proposed recently. We consider here this notion to be a natural consequence of delayed evaluation of constraint checks and apply it to FC. We present a specific algorithm, first-found-forward-checking (F3C), which takes advantage of the delayed evaluation of constraints. We prove that F3C can be no worse than FC, and show substantial reductions of constraint checks in practice. F3C has been effectively used to solve complex CSP’s in a music composition system.

Keywords: constraints, constraint reasoning techniques, arc consistency, backtrack search, lazy evaluation.

1. Introduction

The use of simple forms of constraint reasoning has proved to be very effective in most practical applications. These techniques amount to a pre-processing step using arc consistency schemes (Mackworth[77]) to reduce variable domains followed by a form of backtrack search called forward checking (Haralick[80]). The standard formalism in constraint-based reasoning is a constraint graph defined by a set of variables (the nodes), a domain of values for each variable, and a set of constraints between the variables (the arcs). Efficient arc consistency algorithms (Mohr[86]) rest on the idea of computing for each value of a variable domain a set of supporting values. Supporting values are witnesses that constraints between variables hold. In Bessiere[94a,b] improvements in the time and space complexity of previous arc consistency algorithms are shown to be obtained by computing the set of supports incrementally. With a judicious choice of data structures it can be efficiently guaranteed that a particular element of the set of supports is generated only when it is really needed. The principle involved in this strategy, called delayed evaluation, makes part of the standard trade of functional programming (Abelson[85]). We use it to improve the standard forward checking (FC) algorithm.

FC attempts to reduce backtrack search by eliminating from the domain of some variables those values that are inconsistent with those already chosen for the other variables. It turns out that this process can also be seen as a set of supports computation in such a way that the idea of incrementality applies. In fact, delayed evaluation of data structures can be profitably employed in several aspects of constraint reasoning for improving the time and/or space complexity of search algorithms.

Kondrak[94] proposes a characterization allowing to define theoretically an ordering of FC and various others backtracking algorithms according to their efficiency. We propose in this paper F3C (first found forward checking), a modification of FC with the incremental computation of supports and show that it ranks better than FC in that characterization.

§ This work has been partially supported by grant 12-51-14-041-95 from Colciencias-BID
The paper is organized as follows. Section 2 contains some preliminary definitions and properties on constraints networks. Section 3 describes F3C, section 4 proves the improvement over FC and gives experimental results, section 5 discusses the use of F3C and delayed evaluation in a system for computer aided composition, and section 6 contains conclusions.

2. Background

A constraint network (CN) is a triple $\langle V, D, C \rangle$ where $V = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $D$ is a set of values (the domain) and $C$ is a set of constraints. We assume variables are identified by indexes in a set $X$ where index $i \in X$ identifies variable $x_i \in V$. A given CN $\langle X, D, C \rangle$ is assumed throughout the paper. A domain assignment is a function $\delta: X \rightarrow P(D)$. $\delta(i)$ denotes the domain of values for variable $x_i$. Given assignments $\delta_1, \delta_2$, we say $\delta_1 \subseteq \delta_2$ when for all $i \in X$, $\delta_1(i) \subseteq \delta_2(i)$. Constraints are subsets of the products of their domains (that is, $R(i_1, i_2, \ldots, i_k) \subseteq \delta(i_1) \times \ldots \times \delta(i_k)$). We say that values $v_{i_1} \in \delta(i_1), \ldots, v_{i_k} \in \delta(i_k)$, satisfy constraint $R(i_1, i_2, \ldots, i_k)$ when $(v_{i_1}, \ldots, v_{i_k}) \in R(i_1, i_2, \ldots, i_k)$. Given a domain assignment $\delta$, a value assignment set (VAS) is a function $S: I \subseteq X \rightarrow D$ such that $S(i) \in \delta(i)$. A VAS represents the instantiation of variables with values in their corresponding domains. A VAS $S$ is consistent if for each constraint involving variables instantiated by $S$ the corresponding assignments satisfy the constraint. A solution of a CN is a consistent VAS instantiating all variables.

A constraint network is binary when all its constraints are binary relations. A binary network $\langle V, D, C \rangle$ is usually represented as a directed graph $CG = \langle N, E \rangle$ with $N = V$ and $(x_i, x_j) \in E$ if $R(i, j) \in C$. Arc consistency techniques are mostly used in binary constraint networks. The notion of support is fundamental in these techniques (Mohr[86]). Given a domain assignment $\delta$, for each constraint $R(i, j)$ we say that value $z \in \delta(j)$ is a support for value $v \in \delta(i)$ if $(v, z) \in R(i, j)$. A value $v_j$ is viable if for every $j \in X$ such that $R(i, j) \in C$, $v_j$ has support in $\delta(j)$. $\delta(i)$ is viable when all of its elements are viable. A domain $D$ in a network is arc consistent when for all $i \in X$, $\delta(i)$ is viable.

Extending the notion of support to a VAS will allow us to neatly characterize backtrack search algorithms. We consider only binary networks to simplify notation, but the definitions can easily be extended to n-ary constraints. 

Having a constraint $R(i, j) \in C$ and given a VAS $S$ such that $j \notin \text{dom}(S)$ and a pair $(i, v_i) \in S$, $v_i$ is a support for $S$ if $(v_i, v_j) \in R(i, j)$. A set $A$ is a support for $S$ if each element of $A$ is a support for $S$.

Definition (viability and S-consistency). A VAS $S$ is viable in a domain assignment $\delta$ if for each $i \in \text{dom}(S)$ and for all variables $j \in X$ such that $j \notin \text{dom}(S)$ and $R(i, j) \in C$, $S$ has a support in $\delta(j)$. We assume VAS $\emptyset$ is viable. A domain assignment $\delta$ is S-consistent if for all $j \in X$, $\delta(j)$ is a support for $S$. A domain assignment $\delta$ is maximal S-consistent if it is S-consistent and for all S-consistent $\delta'$, $\delta' \subseteq \delta$. Given a viable VAS $S$ such that $j \notin \text{dom}(S)$ and $v_j \in \delta(j)$ a support for $S$, the VAS $S' = S \cup \{(j, v_j)\}$ is called an extension of $S$ in $\delta$.

Several backtrack search algorithms, including FC, work by extending consistent viable VAS's. Extending a VAS opens up at least three choice possibilities (we assume a given $\delta$):

1. An element must be chosen from the set of candidate variables indexes $I(S) = \{j \notin \text{dom}(S) : j \in X\}$.
2. A support in the domain of the chosen variable must be selected from the set $\sigma_j(S) = \{v \in \delta(j) : v$ is a support for $S\}$.
3. When no extension is possible (i.e. $S$ is not viable), a previously extended VAS $S' \subset S$ must be reconsidered.

Different backtrack search algorithms can be identified by the way in which they handle these sets. Chronological backtracking (BT, Bitner[75]) assumes predefined static orderings both on variables and on domains. The candidates chosen in $I(S)$ and in $\sigma_j(S)$ are the first elements according to their respective orderings. Similarly, when $S$ is
not viable, a backtrack VAS \( S' = S - \{(j, v_j)\} \) is selected, where \( x_j \) is the last variable in the ordering among those instantiated by \( S \). Forward Checking performs essentially the same choices as BT does. The difference lies in that FC computes for a given VAS \( S \) all sets \( \sigma_j(S) \), for every \( j \in I(S) \). This allows to find out earlier whether \( S \) is viable or not and, additionally, reduces the sets of values to inspect for supports in subsequent extensions of \( S \).

\[
\text{FCforward}(S)
\]

\[
\text{if } |S| = n \text{ then output}(S) / * S \text{ is a solution } */
\]

\[
\text{FCback}(S)
\]

\[
\text{else repeat for } j \in 1(S) \text{ such that } R_{(k,j)} \in C
\]

\[
\sigma_j(S) \leftarrow \text{computeSupports}(S, \sigma_j(S'))
\]

\[
\text{until } \sigma_j(S) = \emptyset
\]

\[
\text{if } \exists j \in 1(S) \text{ such that } \sigma_j(S) = \emptyset \text{ then FCback}(S)
\]

\[
j \leftarrow \text{first}(1(S))
\]

\[
v_j \leftarrow \text{first}(\sigma_j(S))
\]

\[
\sigma_j(S) \leftarrow \sigma_j(S) - \{v_j\}
\]

\[
\text{FCforward}(S \cup \{(j, v_j)\})
\]

\[
\text{FCback}(S)
\]

\[
\text{if } S = \emptyset \text{ then return()}
\]

\[
\text{else } (j, v_j) \leftarrow \text{last}(S)
\]

\[
S' \leftarrow S - \{(j, v_j)\}
\]

\[
\text{if } \sigma_j(S') = \emptyset \text{ then FCback}(S')
\]

\[
\text{else } v' \leftarrow \text{first}(\sigma_j(S'))
\]

\[
\sigma_j(S') \leftarrow \sigma_j(S') - \{v'_j\}
\]

\[
\text{FCforward}(S' \cup \{(j, v_j)\})
\]

\[
\text{Figure 1: FC algorithm}
\]

FC (see Figure 1) computes a sequence of viable VAS's and corresponding sequence of domain assignments maintaining the following property:

Property (FC invariant).

Let \( S_0, S_1, \ldots, S_k \) be the sequence of VAS's computed by FC when extending \( S_0 \) to \( S_k \) and let \( \delta_0, \delta_1, \ldots, \delta_k \) be domain assignments such that for all \( j \in X \), \( \delta_i(j) = \sigma_j(S_i) \). For all \( i = 1, \ldots, k - 1 \) the following holds:

1. \( S_i \) is viable. \( S_k \) is consistent.
2. \( \delta_i \) is maximal \( S_i \)-consistent.
3. \( \delta_{i-1} \subseteq \delta_i \)

FC thus computes consistent viable VAS's by maintaining maximal \( S \)-consistent domain assignments. This extra reduction of domains is both simple and very effective (Nadel[85]). In an efficient implementation of FC, keeping all sets \( \sigma_j(S) \) for every \( S \) can be avoided by a judicious choice of data structures but we do not consider these details here. In FC, \text{computeSupports} insures maximal \( S \)-consistency of domain assignments (we assume \( \sigma(\emptyset) = \delta_0 \)). This is a costly operation. It takes \( O(d) \) time for \( d \) the size of a variable domain. \text{computeSupports} is invoked for each constraint involving the last variable instantiated in the current VAS. This gives an \( O(ed) \) computation, where \( e = |C| \). It turns out that the necessary information provided by this computation (i.e. the viability of a VAS) can be obtained more efficiently with delayed exploration of the above mentioned sets. This optimization is in the spirit of that proposed by Bessiere[94a] for arc consistency computations.

3. First Found Forward Checking (F3C).

The new algorithm is based on the notion of delayed-evaluation structure (DES) which is essentially a stream in the sense of Abelson[85]. A DES can be seen as a sequence having an explicit representation of its first element and an
implicit representation of the other elements (it can thus be infinite). DES's are quite convenient to efficiently represent large data structures. We consider a DES as a data type with two operations: \textit{first:DES \to element}, returns the first element of a DES. This operation is assumed to take constant time. \textit{move:DES \to DES} returns the DES resulting after discarding its first element. We use a DES to represent domain assignments. The key idea is to remark that the three properties of FC (see above) can be maintained without explicitly computing domain assignments. We only keep carefully chosen witnesses of each \( \delta(j) \).

**Definition (Witness set)**

A witness set of a domain assignment \( \delta \) is a function \( \omega^\delta : X \to D \cup \{\bot\} \) (where \( \bot \notin D \)) such that for all \( j \in \text{dom}(\delta) \) either \( \omega^\delta(j) \in \delta(j) \) or \( \omega^\delta(j) = \bot \). We extend the notion of \( S \)-consistency to witness sets. Given a VAS \( S \), a witness set of an \( S \)-consistent domain assignment \( \delta_S \), written \( \omega^\delta_X \), is such that \( \omega_X^\delta(i) = S(i) \), for \( i \in \text{dom}(S) \).

Given an ordering relation \( \leq_X \) on \( X \), a witness set can be expressed as a tuple \( (\omega^\delta(i_1), \ldots, \omega^\delta(i_n)) \) where \( i_{k+1} \leq_X i_k \), \( 1 < k \leq n \). We also assume an ordering relation, \( \leq_D \), on \( D \cup \{\bot\} \) such that for all \( d \in D, \bot \leq_D d \), and we order witness tuples lexicographically in the obvious way. For an \( S \)-consistent domain assignment \( \delta_S \), a useful \( S \)-consistent witness set of \( \delta_S \) is \( \omega^\delta_S(j) = \text{first}(\delta_S(j)) \), where \( \text{first} \) denotes the first element of each set in the domain assignment (we define \( \text{first}(\emptyset) = \bot \)). Given a VAS \( S \), F3C uses \( \omega_S^\delta \) both to assert the viability of \( S \) and to provide a delayed representation of \( \delta_S \). As elements of \( \delta_S \) are expanded they are inserted in a separate structure \( \sigma(S) \). The following properties establish how this is done (when no confusion arises we drop superscripts for witness sets).

**Proposition (F3C invariant)**

Let \( S_0 \subseteq \cdots \subseteq S_k \subseteq S \) be the sequence of consistent VAS's and let \( \omega^\delta_{S_0}, \omega^\delta_{S_1}, \ldots, \omega^\delta_{S_k} \) and \( \sigma(S_0), \ldots, \sigma(S_k) \) be corresponding witness tuples and support sets computed by F3C in extending \( S_0 \) to \( S \). Let \( \delta^\neg_S \subseteq \delta^\neg_{S_1} \subseteq \cdots \subseteq \delta^\neg_{S_k} \) be domain assignments such that \( \delta^\neg_{S_i} \) is maximal \( S_i \)-consistent. The following holds:

1. \( \sigma(S_i) \subseteq \delta^\neg_{S_i} \).
2. \( \omega^\delta_{S_i} \) is a witness set (tuple) of \( \sigma(S_i) \) and \( \omega^\delta_{S_k} \supseteq \omega^\delta_{S_1} \).
3. \( \forall j \in X: \omega^\delta_S(j) = \max\{\sigma^\delta_{S_i}(S_i)\} \geq \bot \), where \( \max[A] \) denotes the maximum element of set \( A \) in the pre-defined ordering (we define \( \max(\emptyset) = \bot \)).
4. If \( d \in \delta^\neg_{S_i}(j) \) is such that \( d \notin \sigma^\delta_{S_i}(S_i) \land d < \omega^\delta_{S_i}(j) \) then either every extension \( S' \) of \( S_i \) such that \( S'(j) = d \) is not viable or else it has been output as a solution.
5. \( S(k+1) = \text{first}(\sigma^\neg_{S_k+1}(S_k)) \) is a support of \( S \).

Property 1 says that F3C maintains witnesses of the viability of computed VAS's. Properties 2 and 3 assert that witness sets act as lexicographically moved markers of the known portion of delayed-evaluated consistent domain assignments. Property 4 guarantees that no viable element less than a witness is missed by an explicit part of a domain assignment. Property 5 asserts that extensions are consistent.

As mentioned above, domain assignments are represented "lazily". \( \delta^\neg_0 \) is the initial DES. Given a VAS \( S \), the \( S \)-consistent domain assignment is represented by an explicit and an implicit part. The explicit part \( \sigma(S) \) contains elements of \( \delta^\neg_0 \) already known to be \( S \)-consistent. The implicit part \( \delta_S \subseteq \delta^\neg_0 \) is a DES of currently untested potential supports for \( S \). Both structures are related by \( \delta_S(j) = \emptyset \cup \text{successor}^\delta_{\omega^\delta_X}(\omega^\delta_X(j)) = \text{first}(\delta_S(j)) \), where \( \text{successor}^\delta_X(x) \) is the element of \( \delta(j) \) following \( x \) in the domain ordering. Procedure \text{nextSupport} (see figure 3) updates these structures when a viable VAS \( S \) is extended (F3CForward) or when non viability has been proved and a different extension of an \( S' \subseteq S \) VAS must be tried (F3CBack), thus checking viability of consistent extensions. The following specification captures this behaviour of \text{nextSupport} more precisely.
Proposition (nextSupport postcondition)

Let $\sigma_j(S), \omega_S(j), \delta_S(j)$ be the input and $\sigma_j'(S), \omega_S'(j), \delta_S'(j)$ the computed structures in an invocation of nextSupport. Let $S_0 \subseteq \ldots S_t \subseteq \ldots S$ be the sequence of extensions leading to $S$. The following properties hold:

1. For all $S_t$, $\omega_S'(j) = \text{last}(\sigma_j(S))$.
2. For all $S_t \subseteq S$, the set $\{v \in \sigma_j'(S_t) \mid \omega_S'(j) \leq v < \omega_S'(j)\}$ is not a support for $S$.
3. If there is no $v > \text{first}(\sigma_j(S))$ in $\delta_S(j)$ such that $v$ is support for $S$ then $\sigma_j'(S) = \emptyset$.
4. Either $\sigma_j(S) \neq \emptyset$ and then $\sigma_j'(S) = \{\text{first}(\sigma_j(S))\}$, or $\sigma_j(S) = \emptyset$ and then we have $\forall v > \text{first}(\sigma_j(S)) < v < \text{first}(\delta_S(j)) \land v$ is support of $S = \emptyset$, $\omega_S'(j)$ is the first support of $S$ in $\delta_S(j)$, and $\sigma_j'(S) = \{\omega_S'(j)\}$.

\begin{verbatim}
F3Cforward(S)
  if |S| = n then output(S) /* S is a solution */
  F3Cback(S)
  F3Cforward(S)

F3Cback(S)
  if S = \emptyset then done()
  else (j, v_j) \leftarrow last(S)
  S' \leftarrow S - [(j, v_j)]
  $\omega_S'(j), \sigma_j(S') \leftarrow \text{nextSupport}(S', j)$
  if $\omega_S'(j) = \bot$ then F3Cback(S')
  else F3Cforward(S' \cup [(j, \text{first}(\sigma_j(S')))])

Figure 2. F3C algorithm

nextSupport(j, S): returns $\omega_j(S), \sigma_j$

\begin{verbatim}
v \leftarrow \text{moveToNext}((\sigma_j(S)))
found \leftarrow false
while v \neq \bot \land \neg found do
  isSupport \leftarrow true
  for (i, x) \in S, i = 0 to |S| - 1 while isSupport do
    if v > $\omega_S(i)$ then
      if (x, v) \in R(i, j)
        then $\sigma_j(S_i) \leftarrow \sigma_j(S_i) \cup \{v\}$
        $\omega_S(i) \leftarrow v$
      else isSupport \leftarrow false
    else skip
  if isSupport then found \leftarrow true else v \leftarrow \text{moveToNext}((\sigma_j(S)))
return (\sigma_j(S), \omega_S(S)))

Figure 3. Procedure nextSupport

moveToNext(\sigma_j(S)): returns v

\begin{verbatim}
v \leftarrow \text{first}(\sigma_j(S))
if v = \bot then v \leftarrow \text{first}(\delta_S(j))
if v \neq \bot then $\delta_S(j) \leftarrow \text{move}(\delta_S(j))$
else $\sigma_j(S) \leftarrow \sigma_j(S) - \{v\}$
return v

Figure 4. Procedure moveToNext.
\end{verbatim}

101
Procedure moveToNext (see figure 4) performs a move operation on the DES representing the appropriate domain assignment, when this is needed.

We discuss next some properties supporting the correctness of F3C. We only sketch proofs. We say a VAS $S$ is considered in step 1 of $F3Cforward$ or step 4 of $F3Cback$ (see Figure 2). A VAS $S' \subset S$ is said visited. By the properties mentioned before, a visited VAS is viable and a considered VAS is consistent.

**Lemma 1.**
Let $S_0 \subset \ldots \subset S_k$ be a sequence of VAS's such that $S_k$ is the currently considered VAS. If $\sigma_{k+1}(S_k) \neq \emptyset$ then $S_k$ is viable, has been visited, and the extension $S_k \cup \{(k+1, \text{first}(\sigma_{k+1}(S_k)))\}$ has been considered.

**Proof:** Only an invocation of nextSupport$(k+1, S_k)$ can add values to $\sigma_{k+1}(S_k)$. This cannot have been in the current invocation of $F3Cforward$ or $F3Cback$ since $S_k$ is currently considered, so there must has been a previous consideration of $S_k$. Only $F3Cback$ reconsiders VAS's, so step 4 of $F3Cback$ is being performed. But then $S_k = S - \{(k+1, \text{first}(\sigma_{k+1}(S_k)))\}$, $S$ is consistent and $S_k$ is viable.

**Lemma 2.**
If $S$ is visited by F3C then all consistent direct extensions are considered.

**Proof:** Let $S_0 \subset \ldots \subset S_{k-1} \subset S$ be the extensions computed by F3C such that $S_{k-1}$ is viable (visited), $S$ is consistent and not visited by F3C. There are three reasons for not considering $S$:
1. $v = S(k)$ is not a support for $S_{k-1}$. But then $S$ cannot be consistent.
2. Extension $(k, v) \in S$ is a support but $v < \text{first}(\sigma_k(S_{k-1}))$. Since $v$ is a support, the only reason $v \in \sigma_k(S_{k-1})$ is that nextSupport$(k, S_{k-1})$ moved passed it. But then $\sigma_k(S_{k-1}) \neq \emptyset$ and by lemma 1 extension $S_{k-1} \cup \{(k, v)\} = S$ would have been considered before, contradicting the assumption.
3. Extension $(k, v) \in S$ is a support but $v > \omega_{S_{k-1}}(k)$. F3C can only terminate in a recursive invocation of $F3Cback$ since $F3Cforward$ never changes a VAS before invoking $F3Cback$ and the empty VAS is viable by definition. Since $S_{k-1}$ is visited, a call $F3Cback(S_{k-1})$ from $F3Cback$ must be performed. But this can only happen when nextSupport$(k, S_{k-1})$ returns $\omega_{S_{k-1}}(k) = \bot$, which means that $\delta_{S_{k-1}}(k)$ is the empty DES. But then, by the property linking a DES and its explicit part, nextSupport must have moved passed every support for $S_{k-1}$ in $\delta_{S_{k-1}}(k)$. By lemma 1, $S$ would have been considered, which is a contradiction.

**Corollary.**
F3C visits all (and only) viable VAS's.

**Proof:** follows from the lemmas above.

**Theorem 1.**
F3C finds all (and only) solutions.

**Proof:** (\(\Rightarrow\)) Let $S$ be a solution. Then $S$ is a consistent extension of an $S'$, which must be viable. By lemma 1, $S$ is considered. Since $|S| = n$, it is output as a solution by $F3Cforward$. (\(\Leftarrow\)) Let $S$ be a VAS output by $F3Cforward$. Let $S_0 \subset \ldots \subset S_{k-1} \subset S$ be the extensions by $F3Cforward$. Then $|S| = n$, by the corollary $S_{k-1}$ is viable and by lemma 1 $S$ is consistent. Thus $S$ is a solution.

**4. Efficiency of F3C.**

The characterization proposed in Kondrak[94] considers VAS's as nodes in a tree. This tree represents the search space of a backtrack algorithm. Using as metrics the number of tree nodes (i.e. VAS's) visited and the number of constraints checked on each visit, Kondrak[94] is able to define a precise hierarchy for the efficiency of several backtrack search algorithms: Chronological Backtrack (BT), back jumping (BJ), back marking (BM), conflict-directed back jumping (CBJ), forward checking (FC), really full look-ahead (RFL) and some hybrids (e.g. Backmarking + Backjumping, BMJ). Given algorithms $P$ and $Q$, $P$ appears below $Q$ (i.e. it is better) in the hierarchy if $Q$ visits all VAS's that $P$ visits and $P$ performs less constraint checks than $Q$ in each visited VAS. F3C can be placed below FC in this hierarchy. In Kondrak[94] it is shown that FC visits a node (considers a VAS) iff its father (the VAS it is an extension of) is consistent with all the variables (is viable). The next theorem follows immediately.
Theorem 2.
FC considers all VAS's that F3C considers.

Theorem 3.
FC performs on each considered VAS at least as much constraint checks as F3C does.

proof: Let $S_0 \subseteq \ldots \subseteq S_{k-1} \subseteq S$ be the sequence of extensions computed by F3C, with $S_{k-1}$ viable and $S$ being considered. Let $(v,w) \in R_{(i,j)}$, $i \leq k < j \leq n$ be a constraint checked in an invocation of nextSupport$(j,S)$. Constraint $R_{(i,j)}$ is checked only once in a given invocation since $w$ is eliminated from $\sigma_j(S_i)$ if $i = k$. Then $w \in \delta_S(j)$, a potential support of $v$, is being tested. Since FC tests all potential supports in (the explicit set) $\delta_S(j)$, it must have checked $R_{(i,j)}$. If $i < k$ a potential support of a previously considered VAS $S_i$ is tested. We must also have $w > \omega_{S_i}(j)$, so $(v,w)$ cannot have been checked before in the current sequence of extensions leading to $S$. Also, since nextSupport considers constraints in the underlying variable ordering, $w$ must be a support for all $S_j \subset S_i$. This means $w$ would be a candidate for FC's $\sigma_j(S_i)$ if FC considered $S_i$. By theorem 2, this must be the case. But then FC must have also checked constraint $R_{(i,j)}$, which proves the theorem. We may further notice that FC would have always checked $(v,w)$, while in F3C this only happens if $w > \omega_{S_i}(j)$. Thus F3C might perform less constraint checks than FC.

Figure 5 shows F3C in the hierarchy of Kondrak[94]. RFL was added to this hierarchy in Valencia[95]. Dashed lines express our conjecture regarding F3C+CBJ.

In our experiments we have found that the performance of F3C is quite stable, scoring consistently better than the other algorithms in most of our tests. We tested F3C on two benchmark problems and on a set of randomly generated problems. As benchmarks we used the puzzle problem ZEBRA and a combinatorial theory problem (CTP), both described in Vanhent[89]. For the randomly generated problems a set of parameters are defined to control different aspects of the constraint graph (Sabin[94], Bessiere[93]). $0 \leq pc \leq 1$ denotes the probability of having a constraint between a pair of variables, $0 \leq pu \leq 1$ is the probability that some constraint holds between a given pair of values, $d$ is the domain size and $n$ is the number of variables.

For our experiments we used an implementation of F3C with an explicit representation of $\delta_S$ (i.e. no DES's) and a matrix of markers pointing to witnesses of each $\delta_S(j)$. We also avoided creating copies of domains by maintaining explicitly the set of deleted (i.e. non supports) values in each domain and used it to reinstall them as
needed on backtracking. Results are shown in tables 1-3. We analyzed FC and F3C, both using the min-domains heuristic, i.e. the variable having the smallest current domain is the one selected next for extension. Results of these tests are given in tables 4 and 5.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>Constraint Checks</th>
<th>Number of VAS's considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBJ</td>
<td>939</td>
<td>127</td>
</tr>
<tr>
<td>F3C</td>
<td>1037</td>
<td>144</td>
</tr>
<tr>
<td>BMJ</td>
<td>1227</td>
<td>267</td>
</tr>
<tr>
<td>FC</td>
<td>1329</td>
<td>144</td>
</tr>
<tr>
<td>BJ</td>
<td>2066</td>
<td>267</td>
</tr>
<tr>
<td>BT</td>
<td>2530</td>
<td>339</td>
</tr>
</tbody>
</table>

Table 1. Constraint checks and VAS's considered for the problem ZEBRA.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>Constraint Checks</th>
<th>Number of VAS's considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3C</td>
<td>124858</td>
<td>8626</td>
</tr>
<tr>
<td>BMJ</td>
<td>127015</td>
<td>14582</td>
</tr>
<tr>
<td>FC</td>
<td>164557</td>
<td>8626</td>
</tr>
<tr>
<td>CBJ</td>
<td>620824</td>
<td>14487</td>
</tr>
<tr>
<td>BJ</td>
<td>624377</td>
<td>14582</td>
</tr>
<tr>
<td>BT</td>
<td>6668616</td>
<td>15733</td>
</tr>
</tbody>
</table>

Table 2. Constraint checks and VAS's considered, for CTP

<table>
<thead>
<tr>
<th>Pu</th>
<th>FC</th>
<th>F3C</th>
<th>CBJ</th>
<th>BJ</th>
<th>BMJ</th>
<th>BT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>30</td>
<td>28</td>
<td>50</td>
<td>583</td>
<td>583</td>
<td>708</td>
</tr>
<tr>
<td>10</td>
<td>48</td>
<td>36</td>
<td>272</td>
<td>351</td>
<td>264</td>
<td>771</td>
</tr>
<tr>
<td>15</td>
<td>65</td>
<td>46</td>
<td>48</td>
<td>52</td>
<td>47</td>
<td>52</td>
</tr>
<tr>
<td>20</td>
<td>131</td>
<td>97</td>
<td>195</td>
<td>267</td>
<td>188</td>
<td>967</td>
</tr>
<tr>
<td>25</td>
<td>103</td>
<td>67</td>
<td>232</td>
<td>311</td>
<td>242</td>
<td>2420</td>
</tr>
<tr>
<td>30</td>
<td>228</td>
<td>150</td>
<td>482</td>
<td>565</td>
<td>271</td>
<td>1245</td>
</tr>
<tr>
<td>35</td>
<td>284</td>
<td>188</td>
<td>293</td>
<td>1401</td>
<td>359</td>
<td>4608</td>
</tr>
<tr>
<td>40</td>
<td>991</td>
<td>604</td>
<td>3788</td>
<td>10548</td>
<td>3590</td>
<td>24960</td>
</tr>
<tr>
<td>45</td>
<td>860</td>
<td>515</td>
<td>3808</td>
<td>8230</td>
<td>3653</td>
<td>11757</td>
</tr>
<tr>
<td>50</td>
<td>2093</td>
<td>1418</td>
<td>4722</td>
<td>13477</td>
<td>4195</td>
<td>27158</td>
</tr>
<tr>
<td>55</td>
<td>3211</td>
<td>2169</td>
<td>15481</td>
<td>32635</td>
<td>5972</td>
<td>176578</td>
</tr>
<tr>
<td>60</td>
<td>8461</td>
<td>5667</td>
<td>36540</td>
<td>132939</td>
<td>29623</td>
<td>544724</td>
</tr>
<tr>
<td>65</td>
<td>15067</td>
<td>9573</td>
<td>38846</td>
<td>133502</td>
<td>26479</td>
<td>345203</td>
</tr>
<tr>
<td>70</td>
<td>503</td>
<td>357</td>
<td>620</td>
<td>2396</td>
<td>577</td>
<td>10542</td>
</tr>
<tr>
<td>75</td>
<td>808</td>
<td>716</td>
<td>557</td>
<td>6277</td>
<td>712</td>
<td>15108</td>
</tr>
<tr>
<td>80</td>
<td>190</td>
<td>106</td>
<td>436</td>
<td>436</td>
<td>209</td>
<td>1071</td>
</tr>
<tr>
<td>Total</td>
<td>33073</td>
<td>21737</td>
<td>106370</td>
<td>343970</td>
<td>76964</td>
<td>1167872</td>
</tr>
</tbody>
</table>

Table 3. Random problems with \( pc = 30\% \), \( n = 20 \), \( d = 5 \) and \( 0.05 \leq pu \leq 0.8 \).

<table>
<thead>
<tr>
<th>Pu</th>
<th>FC(Mindom)</th>
<th>F3C(Mindom)</th>
<th>FC/F3C</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3849</td>
<td>2050</td>
<td>0.53</td>
</tr>
<tr>
<td>10</td>
<td>6068.7</td>
<td>3386</td>
<td>0.56</td>
</tr>
<tr>
<td>15</td>
<td>10696.13</td>
<td>4912.5</td>
<td>0.46</td>
</tr>
<tr>
<td>20</td>
<td>15804.4</td>
<td>7498.6</td>
<td>0.47</td>
</tr>
<tr>
<td>25</td>
<td>35814.7</td>
<td>17536.2</td>
<td>0.49</td>
</tr>
<tr>
<td>30</td>
<td>44856.9</td>
<td>25409.8</td>
<td>0.57</td>
</tr>
<tr>
<td>35</td>
<td>103653</td>
<td>67919.1</td>
<td>0.66</td>
</tr>
<tr>
<td>40</td>
<td>194373.5</td>
<td>119939.5</td>
<td>0.62</td>
</tr>
<tr>
<td>n</td>
<td>1478361.3</td>
<td>1147081.6</td>
<td>0.78</td>
</tr>
<tr>
<td>----</td>
<td>-----------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>50</td>
<td>3297041</td>
<td>3235978.3</td>
<td>0.98</td>
</tr>
<tr>
<td>55</td>
<td>3973.1</td>
<td>876.6</td>
<td>0.61</td>
</tr>
<tr>
<td>60</td>
<td>1590</td>
<td>530.6</td>
<td>0.33</td>
</tr>
<tr>
<td>65</td>
<td>1658.8</td>
<td>530</td>
<td>0.21</td>
</tr>
<tr>
<td>70</td>
<td>2015.6</td>
<td>156</td>
<td>0.08</td>
</tr>
<tr>
<td>75</td>
<td>2495.6</td>
<td>121.8</td>
<td>0.05</td>
</tr>
<tr>
<td>80</td>
<td>2940</td>
<td>98</td>
<td>0.03</td>
</tr>
<tr>
<td>85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5224099.93</td>
<td>4638674.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Constraint checks for FC and F3C with min-domain heuristics. Parameters are: $n = 20$, $d = 30$, $pc = 0.5$.

It can be seen in the tables that F3C outperforms FC both with or without the min-domains heuristics. An increase of about 12% in the first case and 25% in the second is observed. If we ignore the instances laying just at the border between under-constraint and over-constraint problems ($pu = 0.5$), the performance increase with min-domains is about 40%.

5. F3C in a musical composition system.

F3C has been used to improve the constraint engine of a visual environment for computer aided musical composition, called Niobé (Assayag[93]). The musical domain is particularly demanding for constraint satisfaction techniques: The number of variables and the domain size tend to be very large (e.g. a variable might denote a chord in a section of a piece), progressions from under-constraint to over-constraint subsequences might be precisely what the composer looks for and constraints are usually considered to have different degrees of importance. Niobé applies different constraint satisfaction techniques. Solution searching used originally FC with an embedded technique for handling soft constraints (Schiex[92]). Delayed evaluation is extensively used in Niobé: Domains are represented implicitly, and all pruning is performed "lazily". In fact, the lazily evaluated AC-5 in Niobé can be straightforwardly adapted to behave as AC-6 (Bessière[94]).

6. Conclusions.

We presented F3C, a new backtrack search algorithm based on the notion of first support defined in Bessière[94] for arc consistency computation. We described the key properties of FC and argued how they can be maintained more efficiently by keeping only a witness from each domain. We showed how this concept can be formally adapted to FC by representing domains as streams and considering delayed constraint filtering. This characterization allowed us to place F3C in the hierarchy proposed in Kondrak[94] and thus to prove the improvement of F3C over FC. In our experiments the improvement over FC was clearly demonstrated in terms of a smaller number of constraint checks, both with or without the min-domains heuristics. Comparison of F3C (without min-domains) with several other well known backtrack search algorithms shows that F3C consistently ranks first in most cases. We described a system for music composition in which the idea of delayed evaluation is profitably used in several aspects of the underlying constraint reasoning engine.

7. References