UM-CRL-TR-15-85

THE UNIVERSITY OF MICHIGAN
COMPUTING RESEARCH LABORATORY

THE CONSISTENT LABELING PROBLBEM, PART 4: THE GENERALIZED FORWARD CHECKING AND WORD-WISE FORWARD CHECKING ALGORITHMS

BERNARD NADEL

CRL-TR-15-85

1079 East Engineering Bldg.
Ann Arbor, MI 48109

THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY*

THE CONSISTENT LABELING PROBLBEM, PART 4: THE GENERALIZED FORWARD CHECKING AND WORD-WISE FORWARD CHECKING ALGORITHMS

BERNARD NADEL

CRL-TR-15-85

December 1985

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

*Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

The CONSISTENT LABELING PROBLEM, Part 4:

The GENERALIZED FORWARD CHECKING and

WORD-WISE FORWARD CHECKING ALGORITHMS[1]

Prof. Bernard A. Nadel[2]

Dept. Electrical Engineering and Computer Science

University of Michigan

Ann Arbor, MI 48109

December 1985

[1] This work was in part supported by the Computer Science Dept. of Rutgers University, New Brunswick, N.J., and in part by the Dept. Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor, MI. This technical report therefore appears as part of the Technical Report series of both departments — DCS-TR-167 at Rutgers U. and CRL-TR-15-85 at U. Michigan.

[2] Previously: Bernard A. Nudel.

The CONSISTENT LABELING PROBLEM, Part 4:

The GENERALIZED FORWARD CHECKING and

WORD-WISE FORWARD CHECKING ALGORITHMS[3]

ABSTRACT

This paper is the fourth in a series on the Consistent Labeling Problem (CLP) — also known as the Constraint Satisfaction Problem — of central interest in Artificial Intelligence and Operations Research. The reader is refered to [7], [8] and [9], the first three papers in the series, for the necessary background and notation. Also relevant are [4], [10]-[12] whose work is generalized and extended in this series of papers.

The present paper presents the Forward Checking algorithm and its variant, the word-wise Forward Checking algorithm in a form which (i) is capable of solving arbitrary instances of the fully general CLP class described in [7] and [8] and which (ii) explicitly allows arbitrary instantiation ordering and constraint-check ordering during the problem-solving process. The relevant notation is introduced in terms of which our subsequent complexity analyses will be able to incorporate these ordering effects, thus providing theoretical insight into how to choose in advance good search orderings. Examples are given emphasizing the dependence of problem-solving complexity on the particular instantiation order and constraint-check order used.

1. Generalized Forward Checking (gFC)

Paralleling the treatment of the generalized Backtracking algorithm (gBT) in [9], this section develops the generalized Forward Checking algorithm (gFC) based on an alternative recursive function for the solution set T of a CLP instance. This algorithm is a version of the Forward Checking algorithm of [4] which also appears anonymously in [6]. However, the version treated here is generalized so as to explicitly allow arbitrary instantiation ordering and constraint-check ordering and to be able to handle any problem of the general class CLP defined in [8] — including instances which, amongst other things, may have constraints corresponding to an arbitrary family of arities and an arbitrary family of argument sets. It should be noted that the Forward Checking algorithm on which gFC is based, was found second-best out of seven CLP algorithms studies empirically in [4]. The algorithm found best was a "word-wise" version of Forward Checking, described in generalized form in section 2 below.

1.1. Developement of the Algorithm

Backtracking is notoriously prone to **thrashing** — the repetitive instantiation of a variable to a value that will fail to satisfy the constraints again and again for the same reason [2], [4], [5]. For example, in figure 3.1 the fifth node at level 3 fails for the same reason as the first node at that level. Backtracking has no ability to learn from its mistakes. Several algorithms have been developed in an attempt to incorporate some kind of memory into the Backtrack search process — see for example the algorithms of [2] and [4]. Perhaps the simplest and yet most effective of these is the Forward Checking algorithm of [6] and [4]. Whereas Backtracking checks the current instantiation at a node against past instantiations, Forward Checking checks the potential future instantiations against those made to-date at the node. If a violation of a relevant problem constraint is found then the offending future-variable value is filtered out of the corresponding domain, never to be checked or instantiated again in the search below that node. To make this explicit we first introduce

[3] In this series of papers the numbering of equations, figures, tables and footnotes begins anew from (1) at the start of each paper. From within the paper that they appear in, such items are referenced simply by their number, say (n). From outside their paper of appearance, an item in paper p of the series is referenced as $(p.n)$.

$$d_f^{\bar{X}_k} = \{ \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\bar{X}_k \,||\, \bar{f}) \in T_j \;\; \forall\, j \in \Psi_{X_k \cup \{f\}} \} \subseteq d_f \tag{1}$$

the **domain of future variable $f \in F_k$ filtered with respect to the instantiations $\bar{X}_k$**. In words, this is the subset of values $\bar{f}$ from the original domain d_f that may be used to extend $\bar{X}_k$ to a consistent labeling $\bar{X}_k \,||\, \bar{f}$. Thus (1) might also be called the **set of consistent f-extensions of $\bar{X}_k$**. Of course if $\bar{X}_k$ is itself inconsistent then $d_f^{\bar{X}_k}$ must be empty, since no value in the original d_f can be used to extend an inconsistent labeling to a consistent one. The list of all future variable domains filtered with respect to $\bar{X}_k$ is denoted

$$\mathbf{d}^{\bar{X}_k} = (d_{x_{k+1}}^{\bar{X}_k} \;\; d_{x_{k+2}}^{\bar{X}_k} \;\; . \; . \;\; d_{x_n}^{\bar{X}_k}) \tag{2}$$

Anticipating the 4-queens example in figure 1 below, the value of this list at say the rightmost node shown at level 2, where $\bar{X}_k = \bar{X}_2 = (2\ 4)$, is

$$\mathbf{d}^{\bar{X}_k} = \mathbf{d}^{(2\ 4)} = (d_{x_3}^{(2\ 4)} \;\; d_{x_4}^{(2\ 4)}) = (\ \{1\} \;\; \{1\ 3\} \ ) \tag{3}$$

In terms of filtered domains, the set of all consistent labelings of Z that are extensions of a given labeling $\bar{X}_k$ of X_k can be expressed by the following recursive function

$$F2(\bar{X}_k \;\; k) = \begin{cases} \emptyset \text{ if } \bar{X}_k \text{ is inconsistent} \\[2ex] \{\ \bar{X}_k\ \} \text{ if } \bar{X}_k \text{ is consistent and } k = n \\[2ex] \emptyset \text{ if } \bar{X}_k \text{ is consistent, } k < n \text{ and } \emptyset \in \mathbf{d}^{\bar{X}_k} \\[2ex] \displaystyle\biguplus_{\bar{x}_{k+1} \in d_{x_{k+1}}^{\bar{X}_k}} F2(\bar{X}_k \,||\, \bar{x}_{k+1} \;\; k{+}1) \text{ otherwise} \end{cases} \tag{4}$$

This is very much like the recursive function $F1$ of (3.7) on which gBT is based, except for the following two differences.

(i) The union of consistent labelings that are extensions of $\bar{X}_{k+1} = \bar{X}_k \,||\, \bar{x}_{k+1}$ is only over values $\bar{x}_{k+1}$ in $d_{x_{k+1}}^{\bar{X}_k} \subseteq d_{x_{k+1}}$ rather than over all values in the original domain $d_{x_{k+1}}$. These are the only values that need be considered for x_{k+1} since by definition they are the only ones that allow $\bar{X}_{k+1}$ itself to be consistent.

(ii) If for any future variable $f \in F_k$ the domain filtered with respect to $\bar{X}_k$ is empty then no consistent completion is possible and the empty set is returned.

Paralleling (3.8) the solution set T for an instance can be expressed as

$$T = F2(\emptyset \;\; 0) \tag{5}$$

But since $F2$ extends the labeling $\bar{X}_k$ only with values guaranteed to lead to a consistent labeling of X_{k+1}, it follows by induction that when computing T in this way, no call $F2(\bar{X}_k \;\; k)$ is ever made for which $\bar{X}_k$ is inconsistent. Thus for computing T, checking for consistency of $\bar{X}_k$ may be avoided on the right side of (4) giving the simpler version

$$F\,2(\overline{X}_k\ \ k) = \begin{cases} \{\ \overline{X}_k\ \}\ \text{if } k = n \\ \emptyset\ \text{ if } k < n \text{ and } \emptyset \in \mathbf{d}^{\overline{X}_k} \\ \displaystyle\biguplus_{\overline{x}_{k+1} \in d^{\overline{X}_k}_{x_{k+1}}} F\,2(\overline{X}_k\ ||\ \overline{x}_{k+1}\ \ k{+}1)\ \text{ otherwise} \end{cases} \tag{6}$$

The right side of (6) makes use of $\mathbf{d}^{\overline{X}_k}$. Anticipating its usefulness at the next level of recursion we include it as an explicit argument of $F\,2$ — passing along $\mathbf{d}^{\overline{X}_k}$ on the right side of (6) and receiving its previous level counterpart $\mathbf{d}^{\overline{X}_{k-1}}$ on the left. We thus obtain the modified version of $F\,2$

$$F\,2(\overline{X}_k\ \ \mathbf{d}^{\overline{X}_{k-1}}\ \ k) = \begin{cases} \{\ \overline{X}_k\ \}\ \text{if } k = n \\ \emptyset\ \text{ if } k < n \text{ and } \emptyset \in \mathbf{d}^{\overline{X}_k} \\ \displaystyle\biguplus_{\overline{x}_{k+1} \in d^{\overline{X}_k}_{x_{k+1}}} F\,2(\overline{X}_k\ ||\ \overline{x}_{k+1}\ \ \mathbf{d}^{\overline{X}_k}\ \ k{+}1)\ \text{ otherwise} \end{cases} \tag{7}$$

In computing the solution set T, for the initial call where $k = 0$ and $\overline{X}_k\ = \overline{X}_0 = \emptyset$ it is convenient to use

$$\mathbf{d}^{\overline{X}_{k-1}} = \mathbf{d}^{\overline{X}_{-1}} = (\emptyset\ \ d_{x_1}\ \ d_{x_2}\ \ .\ \ .\ \ d_{x_n})$$

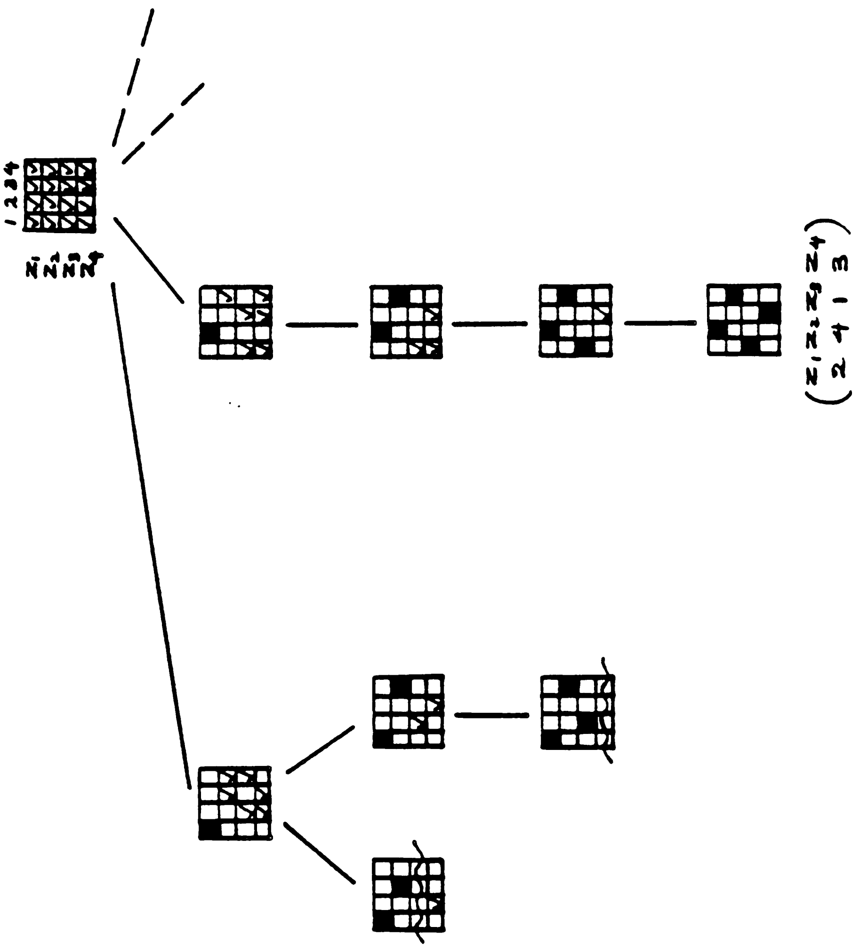
which is the list $\mathbf{d}$ of the original, unfiltered domains d_{x_i} for the variables, prefixed with a dummy element $\emptyset$. This prefixing of $\emptyset$ will allow a uniform implimentation of function $F\,2$ as discussed below.

Paralleling figure 3.1, figure 1 here shows half of the symmetrical search tree when using $F\,2$ to solve 4-queens formulated as in [7]. As in figure 3.1, we are again using for the instantiation order the standard name order $X = (x_1\ \ x_2\ \ x_3\ \ x_4) = (z_1\ \ z_2\ \ z_3\ \ z_4)$. A node is drawn in the search tree for each recursive call to $F\,2$. The node corresponding to call F2($\overline{X}_k\ \ \mathbf{d}^{\overline{X}_{k-1}}\ k$) is called the node at $\overline{X}_k$, or simply node $\overline{X}_k$. Each node $\overline{X}_k$ of the tree shows the corresponding value of $\overline{X}_k$ using the same convention as described for figure 3.1. However, an important addition is that for each future variable $f \in F_k$ (or unplaced queen in the present case) the corresponding row of a node $\overline{X}_k$ is not left empty but is used to show the set $d_f^{\overline{X}_k}$ of still viable values for f given the current instantiations $\overline{X}_k$.[4] In the present case these give the still viable positions for each unplaced queen when the first k queens have been placed on the board as given by $\overline{X}_k$. Values not yet filtered from the original domain d_f are indicated with a check. Filtered values have the corresponding square left blank. For filtered domains that become empty, $d_f^{\overline{X}_k}\ = \emptyset$, a wavy line is shown through the corresponding row. This is called a **domain wipe-out.**

Note that in figure 1 (but not in later such search trees) we indicate in the above manner the (fully) filtered domain $d_f^{\overline{X}_k}$ for *each* future variable $f \in F_k$ at a node $\overline{X}_k$. In practice however, at a node where some future variable has a domain wipe-out, this wipe-out would normally be detected before the computation of all the filtered domains $d_f^{\overline{X}_k}$ from the inputs $d_f^{\overline{X}_{k-1}}$ was fully completed — with filtering of the inputs $d_f^{\overline{X}_{k-1}}$ possibly not having even been started for some future variables. In such a case all further filtering at that node may be avoided, and $d_f^{\overline{X}_k}$ need not necessarily be fully determined for all future variables. (This is one of the reasons for the dependence, discused below, of problem-solving complexity on constraint-check order used.)

[4] Note that the filtered domains input to a node $\overline{X}_k$ are actually $d_f^{\overline{X}_{k-1}}$. These are not shown in the rows of node $\overline{X}_k$ but rather in the rows of the parent node of $\overline{X}_k$. In other words, the rows of node $\overline{X}_k$ show $d_f^{\overline{X}_k}$ which are the input domains *after* filtering at that node rather than before filtering.

Fig. 1: Half of the symmetrical search tree generated by Forward Checking in solving the 4-queens problem under the standard CLP formulation. (Compare with figure 3.1.)



There is a significant difference in the number of nodes generated by gBT and gFC in figures 3.1 and 1. However, this does not provide a meaningful comparison between the two algorithms, since while gFC generates less nodes than gBT does, gFC in general does far more constraint checks at a node.[5] We now consider the details of this constraint-checking process for gFC.

As implied above, the filtered domains of $\mathbf{d}^{\bar{X}_k}$ used on the right side of (7) may be obtained from those of $\mathbf{d}^{\bar{X}_{k-1}}$ input on the left side. Vector $\mathbf{d}^{\bar{X}_k}$ consists of the filtered domains $d_f^{\bar{X}_k}$ for each future variable $f \in F_k$ at node X_k. For each of these variables (and for x_k in addition) the filtered domain $d_f^{\bar{X}_{k-1}}$ is available from $\mathbf{d}^{\bar{X}_{k-1}}$. The question is then how to obtain $d_f^{\bar{X}_k}$ from $d_f^{\bar{X}_{k-1}}$ for a variable $f \in F_k$. By definition, we have from (1) that

$$d_f^{\bar{X}_k} = \{ \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\bar{X}_k \,||\, \bar{f}) \in T_j \quad \forall\, j \in \Psi_{X_k \cup \{f\}} \} \tag{8}$$

$$d_f^{\bar{X}_{k-1}} = \{ \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\bar{X}_{k-1} \,||\, \bar{f}) \in T_j \quad \forall\, j \in \Psi_{X_{k-1} \cup \{f\}} \} \tag{9}$$

From these we see that $d_f^{\bar{X}_k} \subseteq d_f^{\bar{X}_{k-1}}$ so that only values $\bar{f}$ from the latter filtered domain need be considered in computing the former domain. Futhermore, for these values much of the condition required in (8) has already been satisfied by the condition in (9). A little thought shows that for values in $d_f^{\bar{X}_{k-1}}$, the condition of (8) can be ensured by checking consistency of $\bar{X}_k \,||\, \bar{f}$ only with respect to the reduced set of constraints $\Psi_{X_k \cup \{f\}} - \Psi_{X_{k-1} \cup \{f\}}$ — the difference of the two index sets used in (8) and (9) respectively. In addition, at any node $\bar{X}_k$ where such a computation is being performed, we have mentioned above that $\bar{X}_k$ itself is known to be consistent and hence by definition satisfies all constraints in Ψ_{X_k}. Thus these constraints need also not be rechecked in obtaining $d_f^{\bar{X}_k}$. Finally then, (8) can be expressed in terms of (9) as

$$d_f^{\bar{X}_k} = \{ \bar{f} \mid \bar{f} \in d_f^{\bar{X}_{k-1}} \text{ and } \bar{Z}_j(\bar{X}_k \,||\, \bar{f}) \in T_j \quad \forall\, j \in \phi_{kf} \} \tag{10}$$

where $$\phi_{kf} = \Psi_{X_k \cup \{f\}} - \Psi_{X_{k-1} \cup \{f\}} - \Psi_{X_k} \tag{11}$$

$$= \{ j \mid j \in J_1^c \text{ and } \{x_k\ f\} \subseteq Z_j \subseteq X_k \cup \{f\} \} \tag{12}$$

Thus the only constraints that need be checked in filtering the domain of variable $f \in F_k$ at a level k node $\bar{X}_k$ are those whose argument set contains both variables f and x_k and which otherwise contains only variables of X_k, the variables instantiated to date. These, in other words, are the constraints for which the current variable x_k is an argument and its instantiation leaves only *one* more argument variable to instantiate. This is analogous to the situation for gBT where the checkable constraints at level k where those for which x_k was an argument variable and its instantiation left *no* more variables to instantiate. The constraints with indices in ϕ_{kf} we call the constraints **forward-checkable against variable f at level k.** Table 1 shows for our running example clp_0, the sets ϕ_{kf} for each level k and each future variable $f \in F_k$ at that level. As with the sets ψ_k of checkable constraints for gBT, the sets ϕ_{kf} of forward-checkable constraints are a function of the instantiation order used. And as in table 3.3, results are shown for each of the $3! = 6$ instantiation orders possible for the 3 variables of clp_0.

Note that as in this example, for any CLP instance each of the c constraints is forward-checkable against exactly one future variable at exactly one level from 0 to n. The ϕ_{kf} thus constitute a partition of the set of integers 1 to c, so that

[5] Moreover, a meaningful comparison must consider the behaviour over different families of search orderings, since a good family of search orderings for one algorithm may be bad for the other algorithm.

Table 1: Sets ϕ_{kz_i} and ϕ_k at each level k for each of the 6 possible instantiation orders X for instance clp_0.

X	ϕ_{0z_1}	ϕ_{0z_2}	ϕ_{0z_3}	ϕ_{1z_2}	ϕ_{1z_3}	ϕ_{2z_3}	ϕ_0	ϕ_1	ϕ_2
$(z_1\ z_2\ z_3)$	$\emptyset$	$\emptyset$	$\emptyset$	{1}	{2 3}	{4}	$\emptyset$	{1 2 3}	{4}
$(z_1\ z_3\ z_2)$	$\emptyset$	$\emptyset$	$\emptyset$	{2 3}	{1}	{4}	$\emptyset$	{1 2 3}	{4}
$(z_2\ z_1\ z_3)$	$\emptyset$	$\emptyset$	$\emptyset$	{1}	$\emptyset$	{2 3 4}	$\emptyset$	{1}	{2 3 4}
$(z_2\ z_3\ z_1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{1}	{2 3 4}	$\emptyset$	{1}	{2 3 4}
$(z_3\ z_1\ z_2)$	$\emptyset$	$\emptyset$	$\emptyset$	{2 3}	$\emptyset$	{1 4}	$\emptyset$	{2 3}	{1 4}
$(z_3\ z_2\ z_1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{2 3}	{1 4}	$\emptyset$	{2 3}	{1 4}

$$\dot{\bigcup_{k=0}^{n}} \ \dot{\bigcup_{f \in F_k}} \phi_{kf} = \{\ 1\ 2\ ..\ c\ \} \qquad \text{and} \qquad \sum_{k=0}^{n} \sum_{f \in F_k} |\ \phi_{kf}\ | = c \tag{13}$$

There are actually never any constraints to forward-check at level 0 since we are assuming that all constraints involve at least two argument variables. Also, there are never any constraints to forward-check at level n — all constraints have already been forward-checked at prior levels, and any labeling $\overline{X}_k$ generated at level $k = n$ is known to be consisitent without need for further checking, as shown in (6) and (7). We thus have $\phi_{0f} = \phi_{nf} = \emptyset$ for all future variables f at levels 0 and n respectively (there are in fact no future variables at level n). At levels 1 through n–1, sets ϕ_{kf} will in general contain more than one constraint. It therefore becomes relevant to consider the order in which constraints are selected for forward-checking from each set ϕ_{kf} as well as the order in which the sets themselves are chosen. However even this, still assumes we are using a specialized scheme of constraint checking, which we call **f-exhaustive**, where all constraints of a given set ϕ_{kf} are forward-checked before forward-checking those corresponding to a different future variable. In other words an f-exhaustive ordering ϕ_k is some concatenation

$$\phi_k = \phi_{kf_{k1}} \ || \ \phi_{kf_{k2}} \ || \ .. \ || \ \phi_{kf_{k,n-k}} \tag{14}$$

of (possibly empty) orderings ϕ_{kf} of constraints forward-checkable at level k against the various future variables $f \in F_k$. These future variables may however be chosen in any order, and as in (14) we use f_{kt} to denote the t-th such variable chosen in forming the f-exhaustive ordering ϕ_k. Examples of f-exhaustive orderings appear for $k = 3$ in table 4 below. Only f-exhaustive orderings were treated in our earlier paper [12]. The analytic complexity results to appear in later papers of this series give results for the fully general ordering scheme which we now introduce.

Forming the list $\mathbf{d}^{\overline{X}_k}$ of filtered domains at a level k node $\overline{X}_k$ requires the computation of $d_f^{\overline{X}_k}$ for *each* future variable $f \in F_k$ and in total this requires the checking of all constraints in the union

$$\phi_k = \dot{\bigcup_{f \in F_k}} \phi_{kf} = \{\ j \ | \ j \in J_1^c \ \text{and} \ \exists\, f \in F_k \ \text{s.t.}\ \{x_k\ f\} \subseteq Z_j \subseteq X_k \bigcup \{f\}\ \} \tag{15}$$

This union we call the set of constraints **forward-checkable at level k** (against *some* future variable). Besides the sets ϕ_{kf}, table 1 also shows the corresponding unions ϕ_k. Note that paralleling the situation for the sets ϕ_{kf}, we have that $\phi_0 = \phi_n = \emptyset$. And also by inheritance from the ϕ_{kf} from which they are formed, the sets ϕ_k always constitute a partition of the full set of constraint indices, so that paralleling (13) and (3.12) we have

$$\dot{\bigcup_{k=0}^{n}} \phi_k = \{1\ 2\ ..\ c\} \qquad \text{and} \qquad \sum_{k=0}^{n} |\ \phi_k\ | = c \tag{16}$$

The members of a set ϕ_k may very well be checked in any order, even a non f-exhaustive order that interleaves constraints from different sets ϕ_{kf} for a given k. As with varying the instantiation order X, we will see that variations in the constraint-check orders at the different levels can also have a significant effect on problem-solving complexity. To model this we therefore consider set ϕ_k of (15) to have some arbitrary order imposed on it

$$\phi_k = (\phi_k^1 \;\; \phi_k^2 \;\; . \; . \;\; \phi_k^{|\phi_k|}) \tag{17}$$

the i-th component ϕ_k^i being the index of the constraint that is the i-th to be forward-checked at a level k node. Such an ordering we call a **constraint-check ordering at level** k. It is the analog of the constraint-check order for gBT given in (3.13). There are of course $| \phi_k |!$ such orderings possible for a given set ϕ_k, and one is to be chosen by the user at each level k of the search tree. For example, for any n-queens problem there are $(n-k)!$ possible constraint-check orderings at level k, since at that level there are $n-k$ constraints to check — one between the current (or k-th) queen and each of the $n-k$ other queens that are yet to be placed on the board. Each such *future* queen must have its remaining viable positions tested for consistency with that just chosen for the current queen (see figure 1), and the order of pairing the current queen with future queens for this position-compatibility test is arbitrary.

In our later experiments, we will usually be using the **natural constraint-check ordering** ϕ_k at each level, which orders constraints C_j according to increasing index j. (Of course, the ϕ_k themselves, as sets, depend first of all on the instantiation order X, as seen in the example of table 1.) The **vector of constraint-check orderings** for gFC at the various levels we denote by

$$\phi = (\phi_0 \;\; \phi_1 \;\; \phi_2 \;\; . \; . \;\; \phi_{n-1} \;\; \phi_n) = (\emptyset \;\; \phi_1 \;\; \phi_2 \;\; . \; . \;\; \phi_{n-1} \;\; \emptyset) \tag{18}$$

Note that, as for gBT, although we allow an arbitrary instantiation order X and arbitrary constraint-check orders ϕ_k at the various levels, we do still require for gFC that the same variable x_k be instantiated at all nodes of level k (so that the same instantiation order X applies to all paths through the search tree) and that the constraint-check order ϕ_k be the same at all nodes of level k. Actually, an analysis of this globally-fixed search-ordering case can also be used to provide guidance for the locally-determined case — as seen in section 9.2 of [11].

Knowing that a constraint is from ϕ_{kf} makes it clear that the constraint is to be used in filtering the domain of future variable $f \in F_k$ at level k. However it is less clear what is the future variable filtered by constraint ϕ_k^i since all sets ϕ_{kf} for $f \in F_k$ are merged and the result permuted in some way in forming the constraint-check order ϕ_k. However from (12) we see that the future variable corresponding to constraint ϕ_k^i, which we denote as f_k^i or as $f_{\phi_k^i}$, can be recovered as

$$f_k^i = f_{\phi_k^i} = Z_{\phi_k^i} - X_k \tag{19}$$

the set difference between the argument set of the constraint $C_{\phi_k^i}$ and the set of variables X_k.[6] The constraint-check order of (17) then induces the sequence

$$G_k = (f_k^1 \;\; f_k^2 \;\; . \; . \;\; f_k^{|\phi_k|}) \tag{20}$$

of future variables $f \in F_k$ and it is in this order that the latter are chosen to have their domains filtered. We call G_k the induced **domain-filtering order** for future variables at level k. For example

$$G_k = (x_{k+3} \;\; x_{k+1} \;\; x_{k+1} \;\; x_{k+1} \;\; x_{k+3} \;\; x_{k+5}) \tag{21}$$

would be a valid domain-filtering order corresponding to a CLP instance where six constraints are forward-checked at level k. Note that not all future variables $f \in F_k$ are necessarily represented in G_k whereas some may appear more than once, but not necessarily consecutively. In terms of domain filtering, this means that not all future variables necessarily have their domains filtered at the nodes of a given level, while some may be selected for domain filtering multiple times possibly with other variables having their domains filtered in between. This will all depend on the sets ϕ_k induced at the various levels and on the orderings chosen for them. Table 2 shows some of the possible constraint-check orders ϕ_k (permutations of the sets ϕ_k in table 1) together with their induced domain-filtering orders G_k at each level for our running example clp_0 of [7].

[6] The right-hand side of (19) is really a singleton set containing variable f_k^i rather than variable f_k^i itself.

Table 2: Some possible constraint-check orders and their induced domain-filtering orders for instance clp_0 at each level for three different instantiation orders X.

X	ϕ_0 / G_0	ϕ_1 / G_1	ϕ_2 / G_2
$(z_1\, z_3\, z_2)$	∅ / ∅ ∅ / ∅ ∅ / ∅	(1 2 3) / $(z_2\, z_3\, z_3)$ (3 1 2) / $(z_3\, z_2\, z_3)$ (2 1 3) / $(z_3\, z_2\, z_3)$	(4) / (z_2) (4) / (z_2) (4) / (z_2)
$(z_2\, z_3\, z_1)$	∅ / ∅ ∅ / ∅	(1) / (z_1) (1) / (z_1)	(2 4 3) / $(z_1\, z_1\, z_1)$ (3 4 2) / $(z_1\, z_1\, z_1)$
$(z_3\, z_2\, z_1)$	∅ / ∅	(3 2) / $(z_1\, z_1)$	(4 1) / $(z_1\, z_1)$

The above considerations lead us finally to the **generalized Forward-Checking** algorithm or **gFC** presented in figure 2 using a Pascal-like language. As for gBT in [9], it is *generalized* in that it allows arbitrary instantiation order X, arbitrary constraint-check order ϕ_k at each level, and is able to solve arbitrary instances of the very general class CLP defined in [8]. Note that function Filter uses a copy **d** of $\mathbf{d}^{\bar{X}_{k-1}}$. This is simply to avoid using $\mathbf{d}^{\bar{X}_{k-1}}$ as the name of a variable whose value (once filtering is underway) does not agree with the definitions in (1) and (2). In practice this copying need not be done and the filtering can be performed directly with the components of the input vector $\mathbf{d}^{\bar{X}_{k-1}}$.

At line 20, algorithm gFC checks the consistency of value $\bar{f}$ with respect to constraint $C_{\phi_k^{i}}$, given the prior instantiations $\bar{X}_k$. As for gBT it should be remembered that the test $\bar{Z}_j \notin T_j$ is in general just a convenient shorthand for a test performed by a subroutine that represents constraint C_j intensively, rather than indicating a test of membership in an extensive representation of set T_j. The call Update(**d** d^f) at line 24 replaces the previous filtered domain for variable f in **d** with its (generally) new version d^f formed in the FOR loop at lines 17 to 22. Note that d^f is being used to denote the current (generally filtered) set of viable values for variable f, as opposed to d_f which denotes the initial domain of f. At line 26, tail(**d**) is the list **d** with its first element removed. This first element is the filtered domain for variable x_k and is not needed at the next (or lower) levels of the the search tree since x_k has already been instantiated at level k. To allow a uniform implimentation in the case $k = 0$, it is convenient in the initial call gFC($\bar{X}_0$ $\mathbf{d}^{\bar{X}_{-1}}$ 0) to use

$$\mathbf{d}^{\bar{X}_{-1}} = (\emptyset \;\; d_{z_1} \;\; d_{z_2} \; . \; . \; d_{z_n})$$

where we prefix a dummy entry ∅ before the list of initial unfiltered domains for the variables. In this way the initial call to Filter at $k = 0$ will cause tail(**d**) to return $(d_{z_1}\, d_{z_2} \, . \, . \, d_{z_n})$ which is the required value of $\mathbf{d}^{\bar{X}_0}$. Note that it is implicit in gFC that sets d^f are represented directly as lists of their component elements. This will not be so for algorithm gwFC of section 3.5, where a bit vector representation is used for sets d^f.

1.2. Some Examples Using gFC

Paralleling section 2.2 of [9] for gBT, this section shows several examples of using the gFC algorithm of figure 2. Again our standard instance clp_0 of [7] is used as the test case, and is solved using a variety of instantiation orders X and constraint-check orders ϕ_k. And again, the choices of ordering are seen to have a significant effect on the problem-solving complexity of gFC. For more realistic CLP instances much greater variation is possible. For pure and simple binary CLP instances, our earlier paper [11] provides theory-based heuristics for choosing these gFC search orderings and empirically studies the savings that may result from their use.

Figures 3 and 4 represent the tree of recursive calls to gFC when solving clp_0 with the instantiation orderings $X = (z_1\, z_3\, z_2)$ and $X = (z_2\, z_3\, z_1)$ respectively. Essentially the same conventions as described

Fig. 2: Algorithm gFC and its subroutine Filter.

```
PROCEDURE gFC( $\overline{X}_k$ $\mathbf{d}^{\overline{X}_{k-1}}$ $k$ );              : Enter node $\overline{X}_k$
IF $k = n$ THEN print( $\overline{X}_k$ ) ELSE
      BEGIN
      $\mathbf{d}^{\overline{X}_k} \leftarrow$ Filter($\overline{X}_k$ $\mathbf{d}^{\overline{X}_{k-1}}$ $k$)         : Dummy call at $k = 0$
      IF $\mathbf{d}^{\overline{X}_k} \neq (\emptyset)$ THEN
           FOR ALL $\overline{x}_{k+1} \in d_{x_{k+1}}^{\overline{X}_k}$ DO          : $d_{x_{k+1}}^{\overline{X}_k}$ from $\mathbf{d}^{\overline{X}_k}$
               gFC( $\overline{X}_k \,||\, \overline{x}_{k+1}$ $\mathbf{d}^{\overline{X}_k}$ $k$+1);   : Generting node $\overline{X}_{k+1} = \overline{X}_k \,||\, \overline{x}_{k+1}$
      END;
END;

FUNCTION Filter( $\overline{X}_k$ $\mathbf{d}^{\overline{X}_{k-1}}$ $k$ );            : Filters domains in tail of $\mathbf{d}^{\overline{X}_{k-1}}$
$\mathbf{d} \leftarrow \mathbf{d}^{\overline{X}_{k-1}}$;
FOR $i = 1$ TO $|\ \phi_k\ |$ DO                       : $|\ \phi_k\ | = 0$ for $k = 0$ (and $n$)
   BEGIN
   $j \leftarrow \phi_k^i$;                                  : Index of constraint to check
   $f \leftarrow Z_j - X_k$;                            : Determining $f_k^i \in F_k$ whose domain will be filtered
   $d^f \leftarrow$ current domain for $f$ in $\mathbf{d}$;
   FOR ALL $\overline{f} \in d^f$ DO                         : Filter domain $d^f$
      BEGIN
      $\overline{Z}_j \leftarrow \overline{Z}_j( \overline{X}_k \,||\, \overline{f} )$;                 : Projection onto arguments of $C_{\phi_k^i}$
      IF $\overline{Z}_j \notin T_j$                            : Checking constraint $C_{\phi_k^i}$
           THEN $d^f \leftarrow d^f - ( \overline{f} )$;             : Filtering $\overline{f}$ from $d^f$
      END;
   IF $d^f = \emptyset$ THEN RETURN ($\emptyset$)                : A domain wipe-out has occured
           ELSE $\mathbf{d} \leftarrow$ Update( $\mathbf{d}$ $d^f$ );      : Update domain of $f$ in $\mathbf{d}$
   END;
RETURN tail ($\mathbf{d}$);                             : Filter ($\overline{X}_k$ $\mathbf{d}^{\overline{X}_{k-1}}$ $k$) $= \mathbf{d}^{\overline{X}_k}$
END;
```

Initial call:

gFC($\emptyset$ ($\emptyset$ d_{x_1} d_{x_2} . . d_{x_n}) 0)

Global instance parameters:

n, $\mathbf{T} = (T_1\ T_2\ .\ .\ T_c)$, $\mathbf{Z} = (Z_1\ Z_2\ .\ .\ Z_c)$

Global algorithm parameters:

X, $(\phi_1\ \phi_2\ .\ .\ \phi_{n-1})$

for figure 1 are used here. The two families of sets ϕ_{kf} and ϕ_k induced by each instantiation order used here can be seen in the corresponding row of table 1. As for gBT, the node structure of a search tree is determined only by the instantiation order X and not by the choice of constraint-check orderings imposed on the induced sets ϕ_k of forward-checkable constraints at each level. The number of nodes generated at each level k when solving instance *clp* using gFC is denoted $N(\text{gFC } k \ clp)$, and these are shown in the table associated with each figure. Also shown in these tables, at the bottom of the $N(\text{gFC } k \ clp)$ column, are the sums

$$N(\text{gFC } clp) = \sum_{k=1}^{n} N(\text{gFC } k \ clp) \tag{22}$$

giving the total number of nodes generated (excluding the root node). Note that the total number of nodes generated is 5 or 10 depending on the instantiation order used. Again, as for gBT, we see a factor of two difference. But note that in absolute terms, gFC is generating considerably fewer nodes in these examples than did gBT in figures 3.3 and 3.4 where the same instance was being solved. It is easy to show that gFC never generates more nodes than gBT on a given instance with a given instantiation ordering.[7] But this is not the whole story, since gFC in general performs more checks per node.

As mentioned, an instantiation order completely determines the set ϕ_k of constraints forward-checkable at each level k of the search, but the constraint-check *order* at a given level may be any permutation of this set — and these orders may have a considerable influence on the number of constraint-checks performed. Both figures 3 and 4 show numbers of constraint-checks performed using two different families of constraint-check orderings — permutations (appearing in table 2) of the corresponding sets ϕ_k shown in table 1. Each such family of orderings heads a seperate column in the table associated with the figure. In each such column appear the corresponding numbers $C(\text{gFC } k \ clp)$ of constraint-checks performed at each level k of the search tree. These $C(\text{gFC } k \ clp)$ values are of course the sum of the number of constraint-checks performed at each node at level k, so that

$$C(\text{gFC } k \ clp) = \sum_{\overline{X}_k \in D_{X_k}} C(\text{gFC } k \ \overline{X}_k \ clp) \tag{23}$$

where $C(\text{gFC } k \ \overline{X}_k \ clp)$ is the number of constraint-checks performed by gFC at node $\overline{X}_k$ in solving instance *clp*. As for equation (3.16), this sum is over all **node sites** $\overline{X}_k \in D_{X_k}$ at level k, rather than only over nodes actually generated, with the understanding that $C(\text{gFC } k \ \overline{X}_k \ clp) = 0$ at a node site where no node is generated. The number of such node sites at level k for a gFC search tree is again as given in (3.17). For the case of the first (leftmost) family of constraint-check orders ϕ_k appearing in the associated table, figures 3 and 4 label the arc leading to a node by the number of checks $C(\text{gFC } k \ \overline{X}_k \ clp)$ performed at that node. This latter quantity is itself just the sum of the numbers of constraint-checks performed at that node for each of the corresponding forward-checkable constraints. Thus

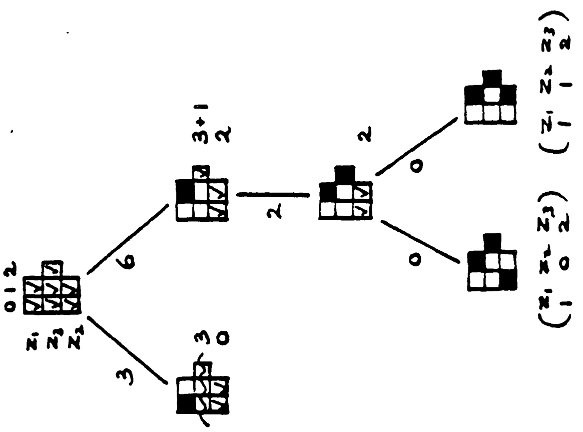
$$C(\text{gFC } k \ \overline{X}_k \ clp) = \sum_{i=1}^{|\phi_k|} C(\text{gFC } k \ i \ \overline{X}_k \ clp) \tag{24}$$

where $C(\text{gFC } k \ i \ \overline{X}_k \ clp)$ is the number of checks of the i-th forward-checkable constraint $C_{\phi_k^i}$ performed by gFC at node-site $\overline{X}_k$ when solving instance *clp*. As before, $C(\text{gFC } k \ i \ \overline{X}_k \ clp) = 0$ at a node-site $\overline{X}_k$ at which no node is actually generated, or at which forward-checking of constraint $C_{\phi_k^i}$ is not reached due to an earlier domain wipe-out at that node. These constraint-specific numbers of checks $C(\text{gFC } k \ i \ \overline{X}_k \ clp)$ that make up the number of checks at a node, can be conveniently indicated on gFC search tree diagrams such as figures 3 and 4, by showing their values alongside the node row associated with the corresponding future variable f_k^i being filtered by constraint $C_{\phi_k^i}$. This is done in figures 3 and 4 for the first family of constraint-check orders tabulated in each figure.

In these figures, if more than one forward-checked constraint filters the same future variable (see table 2), the corresponding numbers of checks for these constraints are shown beside that variable's row as a sum whose summands correspond to the various constraints checked — with the order of summands, left to right, being that of the chronological order of the checking of their corresponding constraints. Thus for example, in figure 3, at the right son $\overline{X}_1 = (1)$ of the root node, using constraint-check order

[7] Since a given node $\overline{X}_k$ is generated by gBT iff $\overline{X}_{k-1}$ is consistent but it is generated by gFC iff the stronger condition is satisfied that $\overline{X}_k$ is consistent and $d_f^{\overline{X}_{k-1}} \neq \emptyset$ for each $f \in F_k$.

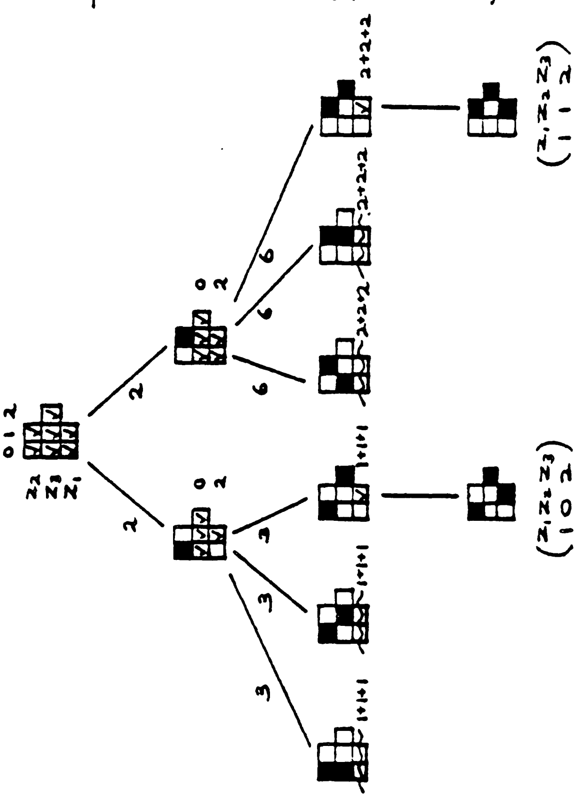
Fig. 3: Solving clp_0 by gFC using a best instantiation-order, $X = (z_1\ z_3\ z_2)$, with a best (and a worst) set of constraint-check orders $\{\phi_1\ \phi_2\}$ for that X. (Compare with figures 4, 3.3 and 3.4.)



k	N(gFC k clp0)		C(gFC k clp0)	
		ϕ_1 ϕ_2	(312) (4)	(213) (4)
1	2		9	16
2	1		2	2
3	2		0	0
	5		11	18

N(gFC clp0) C(gFC clp0)

Fig. 4: Solving clp_0 by gFC using a worst instantiation-order, $X = (z_2\ z_3\ z_1)$, with a worst (and a best) set of constraint-check orders $\{\phi_1\ \phi_2\}$ for that X. (Compare with figures 3, 3.3 and 3.4.)



0 1 2
z_2
z_3
z_1
2
2
0
2
0
2
3
3
3
6
6
6
1+1+1
1+1+1
1+1+1
2+2+2
2+2+2
2+2+2
$\binom{z_1\ z_2\ z_3}{1\ 0\ 2}$
$\binom{z_1\ z_2\ z_3}{1\ 1\ 2}$

k	$N(gFC\ k\ clp_0)$		$C(gFC\ k\ clp_0)$	
		ϕ_1 ϕ_2	(1) (243)	(1) (342)
1	2		4	4
2	6		27	13
3	2		0	0
	10		31	17

$N(gFC\ clp_0)$

$C(gFC\ clp_0)$

$\phi_1 = (3\ 1\ 2)$, C_3 *then* C_2 (the first and third of the forward checkable constraints respectively) are forward-checked to filter the domain of the same future variable $f_1^1 = f_1^3 = z_3$ (as shown in table 2) with constraint C_1 being used between them to filter variable $f_1^2 = z_2$. The corresponding numbers of checks are $C(\text{gFC}\ k\ 1\ \overline{X}_k\ clp) = 3$ and $C(\text{gFC}\ k\ 3\ \overline{X}_k\ clp) = 1$ and these values appear in the sum $3 + 1$ beside the z_3 row of node $\overline{X}_1 = (1)$.

These last-mentioned constraint-specific numbers of checks at a node add, as in (24), to give the total number of checks at a node (and these totals label the arc leading to the node concerned). The numbers of checks performed at the nodes of a level add, as in (23), to give the number of checks performed at that level. And finally, the number of checks at a level add, as below, to give $C(\text{gFC}\ clp)$ the overall total number of constraint-checks performed by gFC when solving clp

$$C(\text{gFC}\ clp) = \sum_{k=1}^{n-1} C(\text{gFC}\ k\ clp) \tag{25}$$

Note that the $k = 0$ and $k = n$ summands are excluded here since gFC never has any constraints to forward check at those levels. The total numbers of checks performed are shown at the bottom of the corresponding $C(\text{gFC}\ k\ clp)$ columns in the tables. We see that as for gBT, changing the family of constraint-check orders may have a considerable effect on the number of constraint-checks performed. This is possible for two reasons. Firstly, the constraint-check order determines at what stage any domain wipe-outs occuring at a node are discovered. As soon as such a wipe-out is detected, the termination condition $\emptyset \in \mathbf{d}^{\overline{X}_k}$ is satisfied and further processing at that node is unecessary.[8] Note that for nodes $\overline{X}_k$ where not only no wipe-out occurs, but where no inconsistency at all is detected (so that no values are filtered), the i-th constraint of ψ_k is checked $| d_{f_k^i}^{\overline{X}_{k-1}} |$ times. In this case, irrespective of the order used for checking the constraints in ϕ_k, we have that

$$C(\text{gFC}\ k\ \overline{X}_k\ clp) = \sum_{i=1}^{|\phi_k|} | d_{f_k^i}^{\overline{X}_{k-1}} |$$

A domain wipe-out is not the only reason that the ϕ_k order can effect the number of constraint-checks at a node. The second reason[9] is because the constraint-check order at a level determines at what stage a given value $\overline{f}$ is elliminated (if it is elliminated) from a given input domain $d_f^{\overline{X}_{k-1}}$. Once it is removed, later constraints in $\phi_{kf} \subseteq \phi_k$ need not check it again. The pattern of these removals at a node $\overline{X}_k$ is determined by the order ϕ_k and over a whole search tree the cummulative effect of using different families of constraint-check orderings may result in large variations in the number of constraint-checks performed in solving a given instance. In figure 4 for example, changing from the family of orderings $\phi_1 = (1)$, $\phi_2 = (2\ 4\ 3)$ to the family $\phi_1 = (1)$, $\psi_3 = (3\ 4\ 2)$ reduces the total number of checks from 31 to 17 — even though the instantiation order is of course the same, $X = (z_2\ z_3\ z_1)$. Using the instantiation order $X = (z_1\ z_3\ z_2)$ and constraint-order family $\phi_1 = (3\ 1\ 2)$, $\phi_2 = (4)$ of figure 3 further reduces the total number of checks performed to 11. We will see in a later paper of this series that this figure, 11 (attained also for gBT in figure 3.3), is in fact for both algorithms gBT and gFC, the minimum number of constraint-checks possible in solving clp_0. Thus as long as the appropriate search orderings are chosen, we see that there exist CLP instances for which gBT is just as efficient (in terms of constraint checks) as gFC in solving a given instance. In fact there are instances for which gBT has a *lower* minimum over orderings than does gFC.

As done for gBT in section 2.2 of [9], we also introduce here two indicator functions in terms of which $N(\text{gFC}\ k\ clp)$ and $C(\text{gFC}\ k\ i\ \overline{X}_k\ clp)$ can be be further expanded. The first is

[8] Figures 3 and 4 show (for the case of the first family of constraint-check orders in each figure) filtered domains and the corresponding number of constraint-checks performed only up until the discovery of a domain wipe-out, at nodes where such wipe-outs occur.

[9] This effect has no analogue in gBT, where only one inconsistency is sufficient to terminate processing at a node. It also does not arise in gFC if (as is the case in most studies, where simple (and usually also pure) binary instances are used) there is no more than one constraint in each set ϕ_{kf} .

$$\delta(\mathrm{gFC}\ k\ \overline{X}_k\ \ clp)\ = \begin{cases} 1 & \text{if gFC generates a node at site } \overline{X}_k \\ & \qquad \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

in terms of which we may express the number of nodes generated at level k as

$$N(\mathrm{gFC}\ k\ clp) = \sum_{\overline{X}_k \,\epsilon\, D_{\overline{X}_k}} \delta(\mathrm{gFC}\ k\ \overline{X}_k\ \ clp) \tag{27}$$

The total number of nodes generated by gFC given in (22) may then be expressed as

$$N(\mathrm{gFC}\ clp) = \sum_{k=1}^{n} \ \sum_{\overline{X}_k \,\epsilon\, D_{\overline{X}_k}} \delta(\mathrm{gFC}\ k\ \overline{X}_k\ \ clp) \tag{28}$$

Whereas for gBT, a given constraint $C_{\phi_k^i}$ is checked at most once at a level k node, for gFC a constraint $C_{\phi_k^i}$ may be checked up to $m_{f_k^i}$ times — once for each of the $m_{f_k^i}$ initial domain values for the variable f_k^i which is by definition the future variable whose domain is filtered by $C_{\phi_k^i}$. Accordingly, it is useful to introduce an indicator function for whether constraint $C_{\phi_k^i}$ is checked exactly t times, for $0 \le t \le m_{f_k^i}$.

$$\delta(\mathrm{gFC}\ k\ i\ t\ \overline{X}_k\ \ clp)\ = \begin{cases} 1 & \text{if gFC checks } C_{\phi_k^i} \text{ exactly } t \text{ times at site } \overline{X}_k \\ & \qquad \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

In terms of this, the number of checks of constraint $C_{\phi_k^i}$ can be expressed as

$$C(\mathrm{gFC}\ k\ i\ \overline{X}_k\ \ clp) = \sum_{t=0}^{m_{f_k^i}} t\ \delta(\mathrm{gFC}\ k\ i\ t\ \overline{X}_k\ \ clp) \tag{30}$$

Note the inclusion of t as a factor of the summed terms. Combining this with the above equalties (23), (24) and (25), gives

$$C(\mathrm{gFC}\ clp) = \sum_{k=1}^{n-1} \ \sum_{\overline{X}_k \,\epsilon\, D_{\overline{X}_k}} \ \sum_{i=1}^{|\phi_k|} \ \sum_{t=0}^{m_{f_k^i}} t\ \delta(\mathrm{gFC}\ k\ i\ t\ \overline{X}_k\ \ clp) \tag{31}$$

for the total number of checks performed by gFC in solving instance clp. The expected values of the total number of nodes generated and of constraint checks performed by gFC in solving an instance are derived in a subsequent paper of this series where equations (28) and (31) provide the starting points for the respective analyses. One last indicator function that will be useful for these analyses can be defined in terms of that in (29). It is

$$\delta(\text{gFC } k\ i \geq 1\ \overline{X}_k\ clp) = 1 - \delta(\text{gFC } k\ i\ 0\ \overline{X}_k\ clp) = \sum_{t=1}^{m_{f_k^i}} \delta(\text{gFC } k\ i\ t\ \overline{X}_k\ clp) \tag{32}$$

Clearly this equals 1 if gFC performs *at least* one check of constraint $C_{\phi_k^i}$ at node $\overline{X}_k$ when solving instance *clp*, and equals 0 otherwise.

1.3. More on gFC Constraint Sets

Table 3 defines several additional constraint (index) sets for gFC that will be useful in our later analyses. Examples of these new sets and the earlier constraint sets for gFC are given in tables 4, 5 and 6. Lastly, table 7 presents without proof some simple relationships between these sets. This section parallels section 2.3 of [9] for gBT, and tables 3 to 7 here correspond tables 4 to 6 of [9] for gBT.

In words, the new constraint sets in table 3 are as follows:

- Φ_k and Φ_{kf} are cummulative versions of ϕ_k and ϕ_{kf} respectively in the same way that Ψ_k is a cummulative version of ψ_k — see (3.26). Thus Φ_k is the set of constraints forward-checkable at some level at or before the k-th and Φ_{kf} is the set of constraints forward-checkable against variable f at some level at or before the k-th.
- $\phi_k(<i)$ is the analogue of $\psi_k(<i)$ for gBT. It denotes the first i–1 constraints forward-checkable at level k.
- $\Phi_k(<i)$ is a cummulative version of $\phi_k(<i)$, and is to $\phi_k(<i)$ as $\Psi_k(<i)$ was to $\psi_k(<i)$ for gBT. $\Phi_k(<i)$ contains those constraints that are forward-checkable at some level prior to k or that are amongst the first i–1 forward-checkable at level k.

Table 3: Constraint-related definitions for algorithm gFC

Symbol	Defined as	Defined for
Φ_{kf}	$\dot{\bigcup}_{h=0}^{k} \phi_{hf}$	$0 \leq k \leq n-1,\ f \epsilon F_k$
Φ_k	$\dot{\bigcup}_{h=0}^{k} \phi_h$	$0 \leq k \leq n-1$
Sets induced by imposing constraint-check orders on the ϕ_k		
$\phi_k(<i)$	$\begin{cases} \{\phi_k^1\ \phi_k^2 \ldots \phi_k^{i-1}\} \\ \emptyset \end{cases}$	$0 \leq k \leq n-1, \begin{cases} 2 \leq i \leq \mid \phi_k \mid \\ i=1 \text{ or } k=0,\ n \end{cases}$
$\phi_{kf}(<i)$	$\phi_{kf} \bigcap \phi_k(<i)$	$0 \leq k \leq n-1,\ 1 \leq i \leq \mid \phi_k \mid,\ f \epsilon F_k$
$\Phi_k(<i)$	$\dot{\bigcup}_{h=0}^{k-1} \phi_h\ \dot{\bigcup}\ \phi_k(<i)$	$0 \leq k \leq n-1,\ 1 \leq i \leq \mid \phi_k \mid$
$\Phi_{kf}(<i)$	$\dot{\bigcup}_{h=0}^{k-1} \phi_{hf}\ \dot{\bigcup}\ \phi_{kf}(<i)$	$0 \leq k \leq n-1,\ 1 \leq i \leq \mid \phi_k \mid,\ f \epsilon F_k$

- ϕ_{kf} $(<i)$ denotes the set of constraints forward-checkable against variable f from amongst the first i–1 forward-checkable constraints at level k. Note that it does *not* mean the first i–1 constraints forward-checkable against f at level k.
- Φ_{kf} $(<i)$ is a cummulative version of ϕ_{kf} $(<i)$, as $\Phi_k(<i)$ was for $\phi_k(<i)$. It contains the constraints forward-checkable against f at some level prior to k or forward-checkable against f from amongst the first i–1 forward-checkable constraints at level k.

Tables 4 to 6 should prove useful in developing familiarity with these and the earlier constraint sets ϕ_k and ϕ_{kf} for gFC. The tables refer to a full and simple 4-ary CLP instance on $n = 5$ variables, as did table 3.5, and they use the same convention for indexing constraints as used in table 3.5. Table 4 shows the partitioning of the problem constraints into sets ϕ_{kf} of (indices of) constraints forward-checkable against variable f at level k, and shows how these are combined to form the sets Ψ_k and Φ_{kf} (the former being as given in (3.10) for gBT). Table 5 emphasizes the alternate partitioning of constraints into the sets ϕ_k of constraints forward-checkable at level k, and their combination into sets Φ_k.

Note that in both these tables, an underlying constraint-check ordering is intended at each level k for the constraints in the corresponding set ϕ_k — and this is the top-to-bottom ordering of constraints given in the ϕ_k^i column as the index i increases. Such a constraint-check ordering underlies the definitions in table 3 of the sets $\phi_k(<i)$, ϕ_{kf} $(<i)$, $\Phi_k(<i)$ and Φ_{kf} $(<i)$. Examples of these sets for the table 4 situation are given in table 6.

Note that in table 4 (and 5), ϕ_3 is an **f-exhaustive** constraint-check ordering, defined in section 1.1 above, where all constraints relevant to a given future variable are employed before proceeding to use those for a different future variable. The ordering of ϕ_2 on the other hand might be called **f-cyclic** since successive constraints filter the domains of *different* future variables till all variables $f \in F_k$ have been filtered by one constraint, with this cycle of domain filtering then repeating till all constraints of ϕ_k have been employed.

Table 7 presents, without proof, some simple relationships between constraint index sets for gFC. They follow more or less directly from the definitions. The results shown for pure and simple A-ary instances should be quite easy to understand in the context of tables 4 to 6, although the latter are for full and simple (rather than for pure and simple) instances. The related discussion in [12] may be helpful.

2. Generalized word-wise Forward Checking (gwFC)

In this section we describe a version of gFC adapted to perform multiple constraint-checking in parallel by exploiting the computer's parallel bit-handling capabilities. Being based on gFC, but being able to check a whole "machine-word-full" of domain values at once, we call the algorithm **generalized word-wise Forward Checking** or **gwFC.** A less general version appears in [6] and in [4] where it is called *bit-parallel Forward Checking.*

One can adapt gFC to perform multiple constraint-checking in parallel by using a bit-vector representation for domains and constraints of a CLP instance. In gFC, the current filtered domain d^f of a future variable f is represented directly as a list of all its member values, and forward checking is performed against each of these values in turn. In gwFC on the other hand, d^f is represented in terms of a machine word with one bit corresponding to each value of the initial domain d_f. Set $d^f \subseteq d_f$ is represented by setting the i-th bit of the machine word to 1 iff the corresponding value of d_f is in d^f. This is a standard way of representing sets, and is treated for example in [1], where it is called the *characteristic vector* representation. We assume for the moment that each domain is small enough to require no more than one machine word. That is, if there are b bits in a machine word, we assume $m_{z_i} \leq b$ for all problem variables.

Besides the domains, gwFC also represents the CLP instance constraints in bit-vector form. In filtering the domain of a variable f, parallel constraint-checking is achieved by machine *anding* an appropriate bit-vector from the constraint representation with the bit vector representing d^f. The following makes this more explicit.

As given in (19), to each constraint $C_j = (Z_j \ T_j)$ and instantiation order X, there corresponds a unique future variable $f_j \in Z_j$ whose domain is filtered by that constraint.[10] At the stage when C_j is

[10] Since this variable f_j is a function of X, the structure of the gwFC bit-vector representation for constraints — described below — is also a function of X. Use of a different instantiation order will require a reconfiguration of the data structure represent-

Table 4: Constraint-index sets ϕ_{kf} , Φ_{kf} and Ψ_k for a full and simple 4-ary instance with $n = 5$ variables when using the generic instantiation order $X = (x_1\ x_2\ x_3\ x_4\ x_5)$. (Compare with table 3.5.)

k	$\vert \phi_k \vert$	i	ϕ_k^i	$\vert \phi_{kf} \vert$
1	4	1	12	1
		2	13	
		3	14	
		4	15	
2	6	1	23	2
		2	24	
		3	25	
		4	123	
		5	124	
		6	125	
3	8	1	34	4
		2	134	
		3	234	
		4	1234	
		5	35	
		6	135	
		7	235	
		8	1235	
4	7	1	45	7
		2	145	
		3	245	
		4	345	
		5	1245	
		6	1345	
		7	2345	

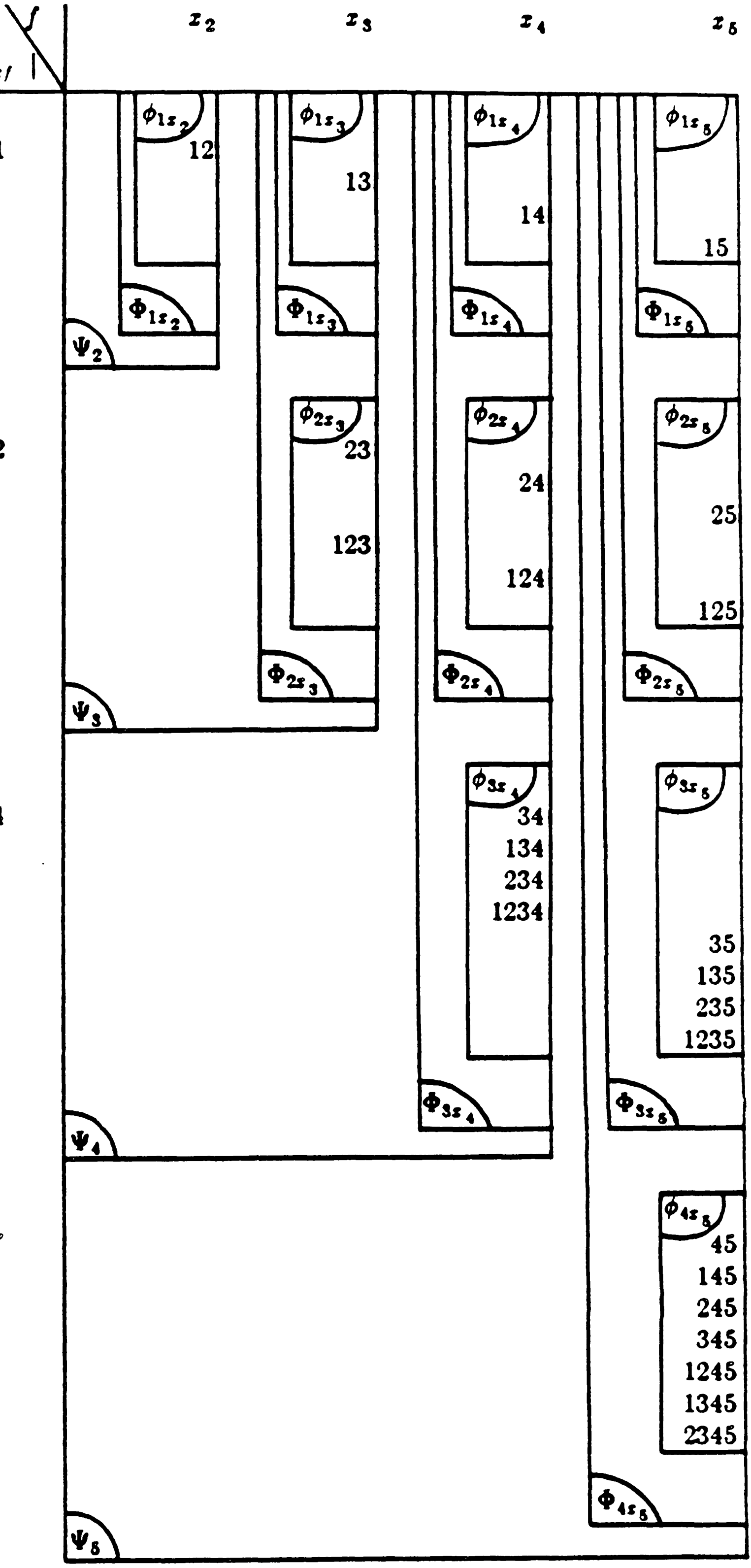


Table 5: A repeat of the table 4 situation showing constraint-index sets ϕ_{kf} , ϕ_k and Φ_k .
(Compare with table 3.5.)

k	$\lvert\phi_k\rvert$	i	ϕ_k^i	$\lvert\phi_{kf}\rvert$
1	4	1	12	1
		2	13	
		3	14	
		4	15	
2	6	1	23	2
		2	24	
		3	25	
		4	123	
		5	124	
		6	125	
3	8	1	34	4
		2	134	
		3	234	
		4	1234	
		5	35	
		6	135	
		7	235	
		8	1235	
4	7	1	45	7
		2	145	
		3	245	
		4	345	
		5	1245	
		6	1345	
		7	2345	

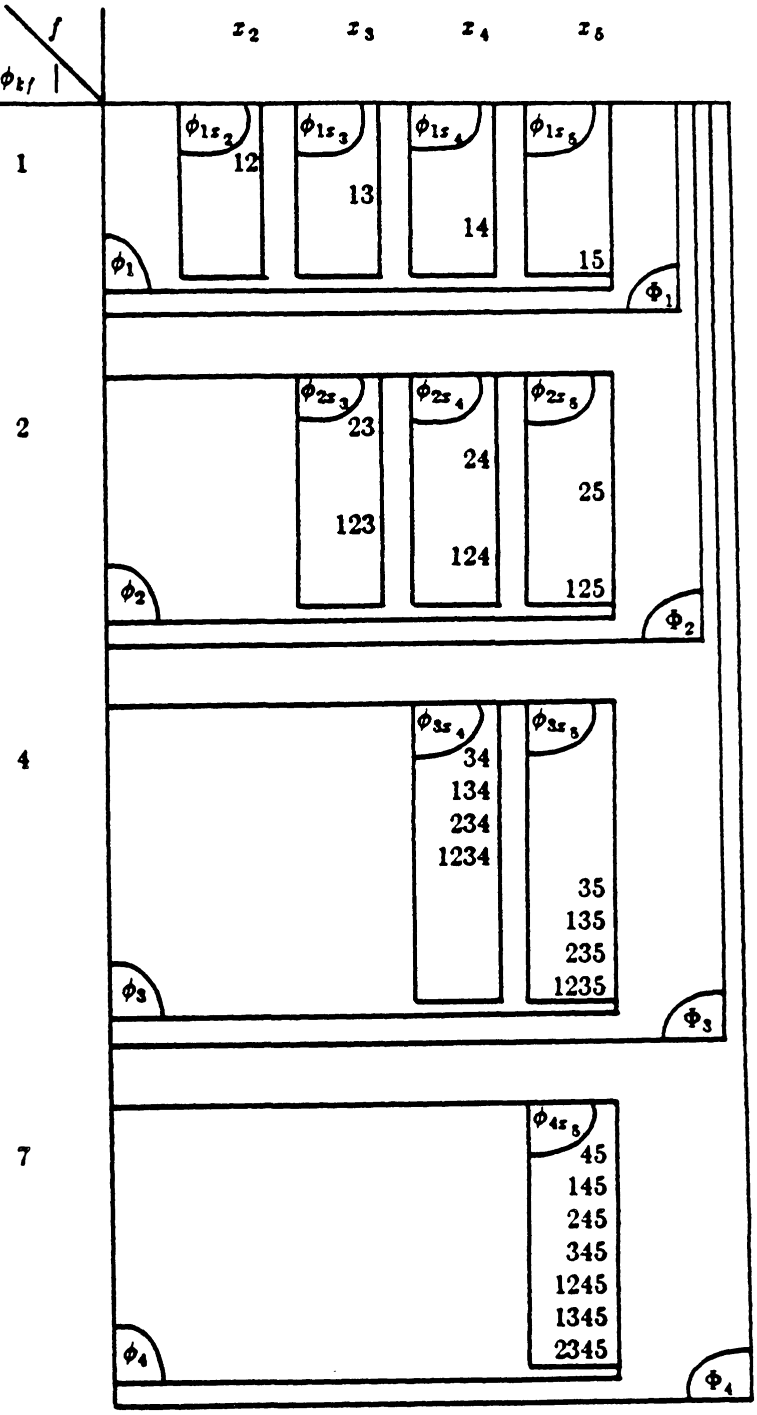
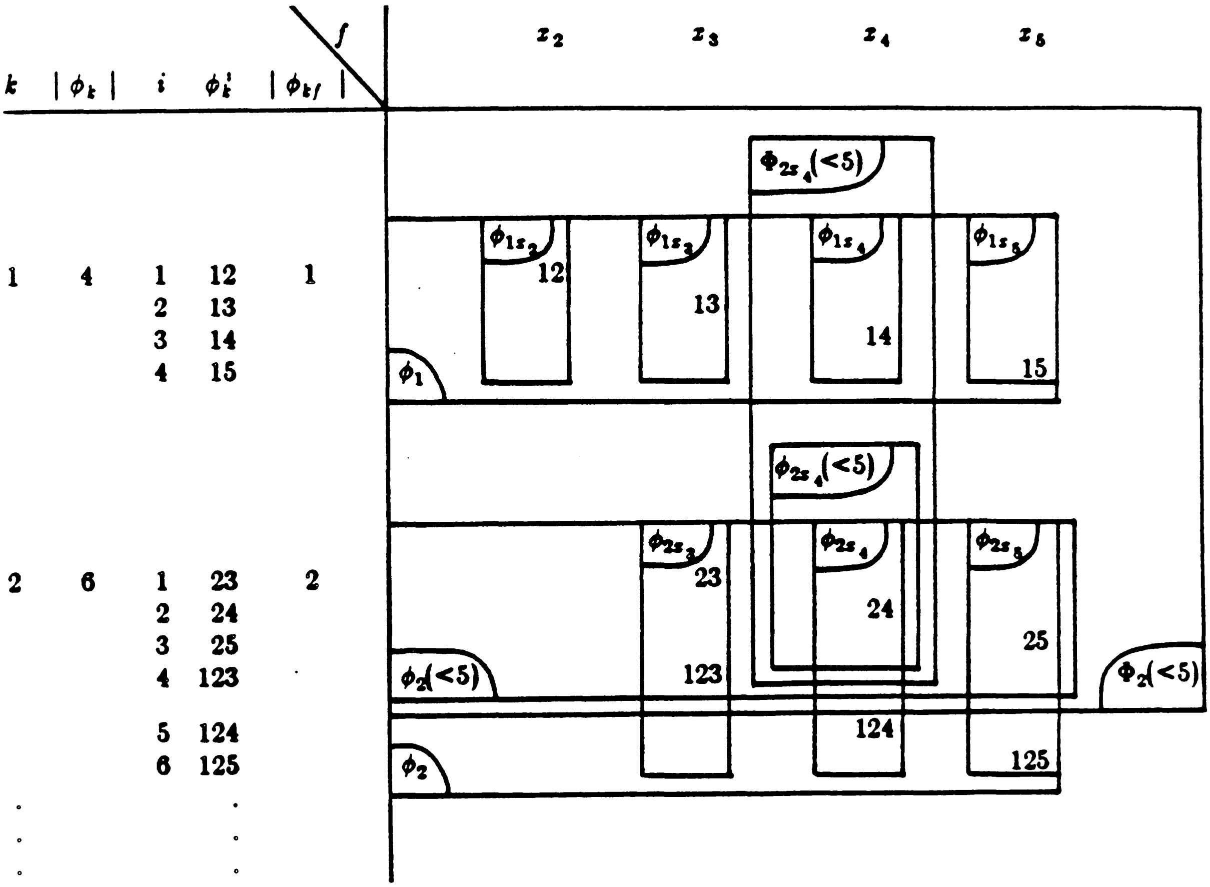


Table 6: A repeat of the table 4 situation showing constraint-check-order-dependent sets $\phi_k(<i)$, $\phi_{kf}(<i)$, $\Phi_k(<i)$ and $\Phi_{kf}(<i)$ defined in table 3.

(Compare with table 3.5.)



k	$\lvert\phi_k\rvert$	i	ϕ_k^i	$\lvert\phi_{kf}\rvert$
1	4	1	12	1
		2	13	
		3	14	
		4	15	
2	6	1	23	2
		2	24	
		3	25	
		4	123	
		5	124	
		6	125	
.			.	
.			.	
.			.	

$$\phi_2(<5) = \{\ 23\ 24\ 25\ 123\ \}$$

$$\Phi_2(<5) = \phi_1 \uplus \phi_2(<5) = \{\ 12\ 13\ 14\ 15\ \} \uplus \{\ 23\ 24\ 25\ 123\ \} = \{\ 12\ 13\ 14\ 15\ 23\ 24\ 25\ 123\ \}$$

$$\phi_{2x_3}(<5) = \phi_{2x_3} \cap \phi_2(<5) = \{\ 23\ 123\ \} \cap \{\ 23\ 24\ 25\ 123\ \} = \{\ 23\ 123\ \}$$

$$\phi_{2x_4}(<5) = \phi_{2x_4} \cap \phi_2(<5) = \{\ 24\ 124\ \} \cap \{\ 23\ 24\ 25\ 123\ \} = \{\ 24\ \}$$

$$\phi_{2x_5}(<5) = \phi_{2x_5} \cap \phi_2(<5) = \{\ 25\ 125\ \} \cap \{\ 23\ 24\ 25\ 123\ \} = \{\ 25\ \}$$

$$\Phi_{2x_3}(<5) = \phi_{1x_3} \uplus \phi_{2x_3}(<5) = \{\ 13\ \} \uplus \{\ 23\ 123\ \} = \{\ 13\ 23\ 123\ \}$$

$$\Phi_{2x_4}(<5) = \phi_{1x_4} \uplus \phi_{2x_4}(<5) = \{\ 14\ \} \uplus \{\ 24\ \} = \{\ 14\ 24\ \}$$

$$\Phi_{2x_5}(<5) = \phi_{1x_5} \uplus \phi_{2x_5}(<5) = \{\ 15\ \} \uplus \{\ 25\ \} = \{\ 15\ 25\ \}$$

Table 7: Relations between Constraint Index Sets for gFC

Arbitrary instances:

$$\{\,1\;2\,..\,c\,\} = \dot{\bigcup_{k=0}^{n}} \; \dot{\bigcup_{f\in F_k}} \phi_{kf} = \dot{\bigcup_{k=0}^{n}} \phi_k = \Phi_n = \Phi_{n-1} = \dot{\bigcup_{k=0}^{n}} \psi_k = \Psi_n \tag{33}$$

$$\Phi_{kf} = \Phi_{k-1,f} \;\dot{\bigcup}\; \phi_{kf} \qquad (\Phi_{0f} = \phi_{0f} = \phi_{nf} = \emptyset) \tag{34}$$

$$\Phi_k = \Phi_{k-1} \;\dot{\bigcup}\; \phi_k \qquad (\Phi_0 = \phi_0 = \phi_n = \emptyset) \tag{35}$$

$$\Phi_k(<i) = \Phi_{k-1} \;\dot{\bigcup}\; \phi_k(<i) \qquad (\Phi_0(<i) = \phi_0(<i) = \phi_n(<i) = \emptyset) \tag{36}$$

$$\Phi_{kf}(<i) = \Phi_{k-1,f} \;\dot{\bigcup}\; \phi_{kf}(<i) \qquad (\Phi_{0f}(<i) = \phi_{0f}(<i) = \phi_{nf}(<i) = \emptyset) \tag{37}$$

$$\Phi_{kf}(<1) = \Phi_{k-1,f} \tag{38}$$

$$\Phi_{kx_{k+1}} = \psi_{k+1} \tag{39}$$

$$|\,\phi_k(<i)\,| = i-1 \tag{40}$$

$$|\,\Phi_{kf}\,| = \sum_{h=0}^{k} |\,\phi_{hf}\,| = |\,\Phi_{k-1,f}\,| + |\,\phi_{kf}\,| \tag{41}$$

$$|\,\Phi_{kf}(<i)\,| = \sum_{h=0}^{k-1} |\,\phi_{hf}\,| + |\,\phi_{kf}(<i)\,| = |\,\Phi_{k-1,f}\,| + |\,\phi_{kf}(<i)\,| \tag{42}$$

$$\Psi_k = \dot{\bigcup_{h=0}^{k-1}} \Phi_{hx_{k+1}} = \dot{\bigcup_{h=0}^{k-1}} \; \dot{\bigcup_{t=1}^{h}} \phi_{tx_{k+1}} \tag{43}$$

$$|\,\Psi_k\,| = \sum_{h=0}^{k-1} |\,\Phi_{hx_{k+1}}\,| \tag{44}$$

$$|\,\Psi_n\,| = \sum_{h=0}^{n} |\,\Phi_{hx_{k+1}}\,| = c \tag{45}$$

Table 7: Continued

Pure and Simple A-ary instances ($A \geq 2$):

$$| \phi_{kf} | = \binom{k-1}{A-2} \tag{46}$$

$$| \phi_{k} | = \binom{k-1}{A-2} (n-k) \tag{47}$$

$$| \Phi_{kf} | = \sum_{h=0}^{k} | \phi_{hf} | = \sum_{h=0}^{k} \binom{h-1}{A-2} = \binom{k}{A-1} \tag{48}$$

$$| \Phi_{kf} (<i) | = | \Phi_{k-1,f} | + | \phi_{kf} (<i) | = \binom{k-1}{A-1} + | \phi_{kf} (<i) | \tag{49}$$

$$| \Psi_{k} | = \sum_{h=0}^{k-1} | \Phi_{hx_{k+1}} | = \sum_{h=0}^{k-1} \binom{h}{A-1} = \binom{k}{A} - \binom{0}{A-1} = \binom{k}{A} \quad \text{since } A \geq 2 \tag{50}$$

$$| \Psi_{n} | = c = \binom{n}{A} \tag{51}$$

forward-checked, all its argument variables except this f_j have been instantiated. We denote this instantiated subset of arguments by $Z_j - f_j$, and a list of instantiations of this subset we denote by $\overline{Z_j - f_j}$. Algorithm gwFC requires that each relation T_j be stored as an array indexed by value-tuples $\overline{Z_j - f_j}$. For a given such value-tuple, the array contains $T_j(\overline{Z_j - f_j})$, a machine word representing, in characteristic vector form, the set of values $\overline{f_j}$ acceptable for variable f_j with respect to constraint C_j given the instantiations in $\overline{Z_j - f_j}$ for the other arguments of the constraint. In other words, $T_j(\overline{Z_j - f_j})$ is the machine word representing the set of values

$$\{ \overline{f_j} \mid \overline{f_j} \in d_{f_j} \text{ and } \overline{Z_j - f_j} \,||\, \overline{f_j} \in T_j \}$$

With these machine words available, the loop from lines 17 to 22 of function Filter in figure 2 can be replaced by a single machine *and* as follows

$$d^f \leftarrow \text{and}(\ d^f \ \ T_j(\overline{Z_j - f_j})\ ) \tag{52}$$

which is the generalized form of Haralick's expression on page 271 of [4]. The number of *ands* performed by gwFC is thus never greater than the number of constraint-checks performed by gFC since a single *and* is performed by gwFC iff at least one check is performed in gFC in the loop at lines 17 to 22 of figure 2.

However, as mentioned, we have assumed that no initial domain has more members than there are bits in a machine word — so that one machine word suffices to represent any of its subsets. More generally however, if domain d_{z_i} has size m_{z_i} and if a machine word contains b bits then

$$w_{z_i} = \lceil m_{z_i}/b \rceil \tag{53}$$

machine words are required to represent the subsets of d_{z_i}. The loop over $\overline{f}$ values in Filter is then replaced not by a single *and* as in (52), but by the following loop over machine words

$$\text{FOR } w = 1 \text{ TO } w_f \text{ DO} \tag{54}$$

$$d^f(w) \leftarrow \text{and}(\ d^f(w) \ \ T_j(\overline{Z_j - f_j} \ \ w)\ )$$

where w is being used to index the w_f seperate words required to represent d^f and the corresponding constraint information.

When more than one word is required to represent a given domain, the possibility arises that gFC may perform less checks than gwFC does *ands* in solving a given instance.[11] In forward checking a constraint against a variable f whose current filtered domain is d^f, gFC performs $|\ d^f\ |$ constraint-checks (at lines 17 to 22 of figure 2). Algorithm gwFC on the other hand will perform w_f machine *ands* in the loop of (54), independent of the size of the current filtered domain $d^f \subseteq d_f$. So the relative efficiency of the two algorithms depends in part on the size of a machine word relative to the sizes of the various filtered domains d^f encountered throughout the search tree. At lower nodes (larger value of k) in the tree, d^f will tend to be small, so that $w_f > |\ d^f\ |$. The relative inefficiency of gwFC in such cases may very well offset the savings gained by that algorithm at the the higher nodes, where $w_f < |\ d^f\ |$. (This suggests a hybrid algorithm, using the gwFC approach at higher nodes and the gFC approach at lower nodes.)

For certain instances with domains for which more than one word is required, it is therefore conceivable that gFC is preferable to gwFC. The gwFC analysis of a subsequent paper in this series will allow this possibility to be studied theoretically, since the analysis will be for domains and machine words of arbitrary size. Simplified versions of gFC and gwFC are compared in our earlier paper [11].

Whether or not gwFC proves superior to gFC in terms of the respective complexity measures of *ands* and checks, it should be noted that gwFC incurs what may be a significant extra overhead in setting up the above-mentioned bit-vector representation of the constraints. This preprocessing is not required by gFC. On the other hand, if gFC is to incorporate heuristics of the type developed in [11], which make use of constraint satisfiability values S_j, it may be necessary to apply an equivalent amount of preprocessing to extract the S_j values. This will depend on the form of the constraints for the problem and on which extraction methods are applicable from those discussed in section 4 of [8].

[11] Haralick in [4] found word-wise Forward Checking to be better than Forward Checking, but he used domain sizes small enough to require only a single word.

Postscript

The next paper in this series deals with probablity models over the class of CLP instances. It defines events of the type needed for our later complexity analyses of algorithms gBT, gFC and gwFC, and derives the probabilities of these events as well as certain basic expected values. However all results of the following paper are presented at a level of generality that makes them independent of any algorithm — so that they reflect only the nature of CLP and the probability models used. Later papers in this series will specialize these results as required for the algorithm analyses.

REFERENCES

[1] Aho A. V., Hopcroft J. E., and Ullman J. D., *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass, 1974.

[2] Gaschnig J., *Performance Measurement and Analysis of Certain Search Algorithms,* Dept. Computer Science, Carnegie-Mellon University, May 1979, Ph.D. dissertation.

[3] Goldberg A., "Average case complexity of the satisfiability problem," *Proc. 4-th Workshop on Automated Deduction,* pp. 1-6, Austin, Texas, 1979.

[4] Haralick R. M. and Elliot G. L., "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence,* vol. 14, pp. 263-313, 1980.

[5] Mackworth A. K., "Consistency in networks of relations," *Artificial Intelligence* , vol. 8, pp. 99-118, 1977.

[6] McGregor J. J., "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms," *Information Sciences,* vol. 19, pp. 229-250, 1979.

[7] Nadel B. A., "Consistent Labeling Problem, Part 1: Background and Problem Formulation," Report DCS-TR-164, Computer Science Dept., Rutgers University., New Brunswick, N.J., 1985, Also appears as Report CRL-TR-13-85, Dept. Electrical Engineering and Computer Science, U. of Michigan, Ann Arbor, MI, 1985.

[8] Nadel B. A., "Consistent Labeling Problem, Part 2: Subproblems, Enumerations and Constraint Satisfiability," Report DCS-TR-165, Computer Science Dept., Rutgers University., New Brunswick, N.J., 1985, Also appears as Report CRL-TR-14-85, Dept. Electrical Engineering and Computer Science, U. of Michigan, Ann Arbor, MI, 1985.

[9] Nadel B. A., "Consistent Labeling Problem, Part 3: The Generalized Backtracking Algorithm ," Report DCS-TR-166, Computer Science Dept., Rutgers University., New Brunswick, N.J., 1985, Also appears as Report CRL-TR-12-85, Dept. Electrical Engineering and Computer Science, U. of Michigan, Ann Arbor, MI, 1985.

[10] Nudel B. A., "Consistent Labeling Problems and their algorithms," *Proc Nat. Conf. on Artificial Intelligence (AAAI),* pp. 128-132, Pittsburgh, August 1982.

[11] Nudel B. A., "Consistent-labeling problems and their algorithms: expected-complexities and theory based heuristics," *Artificial Intelligence (Special Issue on Search and Heuristics),* vol. 21, no. 1 and 2, pp. 135-178, March 1983, Also in book: Search and Heuristics, North-Holland, Amsterdam 1983.

[12] Nudel B. A., "Solving the general consistent labeling (or constraint satisfaction) problem: Two algorithms and their expected complexities," *Proc Nat. Conf. on Artificial Intelligence (AAAI),* pp. 292-296, Washigton D.C., August 1983, Also available as report DCS-TR-128, Computer Science Dept., Rutgers University, New Brunswick, N.J., 1983.

AIIM SCANNER TEST CHART # 2

Spectra

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Times Roman

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Century Schoolbook Bold

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

News Gothic Bold Reversed

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Bodoni Italic

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789
10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Greek and Math Symbols

4 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°><≯≮⩾⩽≡
6 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°><≯≮⩾⩽≡
8 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°><≯≮⩾⩽≡
10 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°><≯≮⩾⩽≡

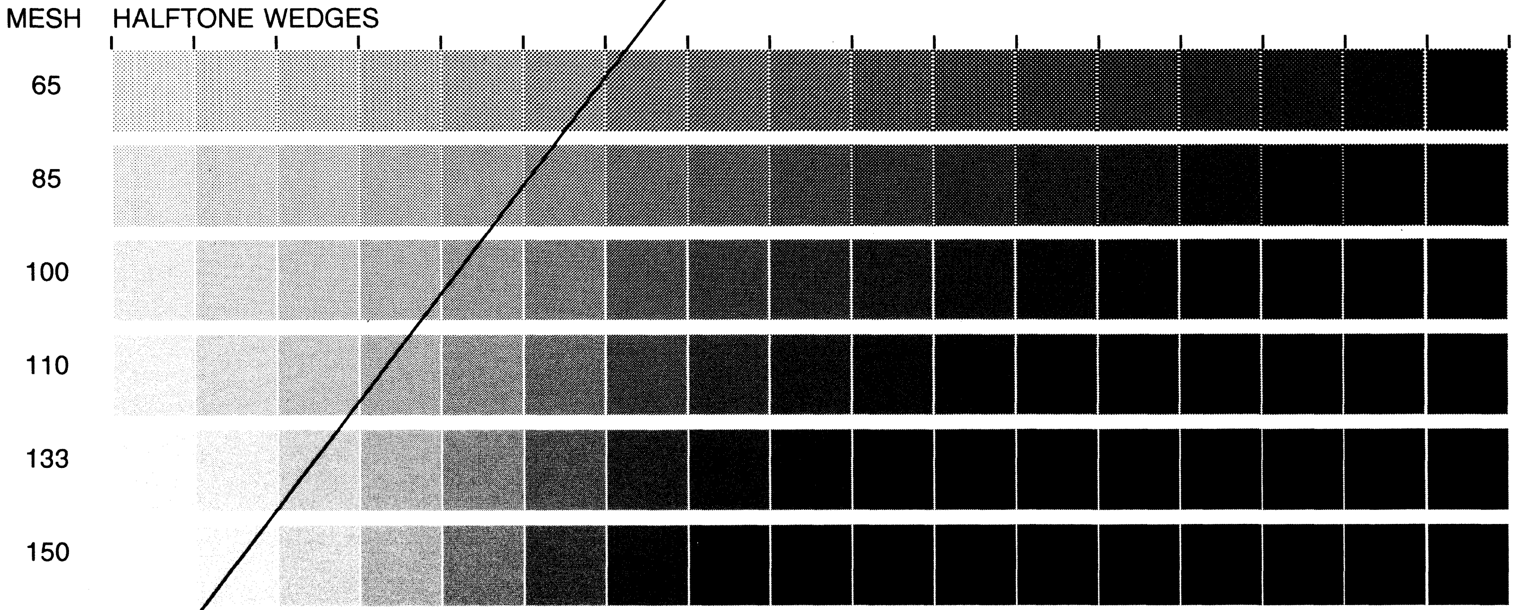
White □

Black ■

Isolated Characters

e	m	1	2	3	a
4	5	6	7	o	°
8	9	0	h	l	B

0123456 A4 Page 6543210



RIT ALPHANUMERIC RESOLUTION TEST OBJECT, RT-1-71

PRODUCED BY GRAPHIC ARTS RESEARCH CENTER

ROCHESTER INSTITUTE OF TECHNOLOGY, ONE LOMB

MEMORIAL DRIVE, ROCHESTER, NEW YORK 14623