THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

THREE CONSTRAINT SATISFACTION ALGORITHMS AND THEIR COMPLEXITIES: SEARCH-ORDER DEPENDENT AND EFFECTIVELY INSTANCE-SPECIFIC RESULTS

BERNARD NADEL

CRL-TR-3-86

1079 East Engineering Bldg.
Ann Arbor, MI 48109

THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY*

THREE CONSTRAINT SATISFACTION ALGORITHMS AND THEIR COMPLEXITIES: SEARCH-ORDER DEPENDENT AND EFFECTIVELY INSTANCE-SPECIFIC RESULTS

BERNARD NADEL

CRL-TR-3-86

January 1986

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 author/s and do not necessarily reflect the views of the funding agency.

THREE CONSTRAINT SATISFACTION ALGORITHMS and THEIR COMPLEXITIES: SEARCH-ORDER DEPENDENT and EFFECTIVELY INSTANCE-SPECIFIC RESULTS[1]

Prof. Bernard A. Nadel[2]

Dept. Electrical Engineering and Computer Science

University of Michigan

Ann Arbor, MI 48109

January 1986

[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-171 at Rutgers U. and CRL-TR-3-86 at U. Michigan.

[2] Previously: Bernard A. Nudel.

THREE CONSTRAINT SATISFACTION ALGORITHMS and THEIR COMPLEXITIES: SEARCH-ORDER DEPENDENT and EFFECTIVELY INSTANCE-SPECIFIC RESULTS

ABSTRACT

The Consistent Labeling (or Constraint Satisfaction) Problem[3] CLP, constitutes one of the central problem-solving paradigms in Artificial Intelligence. Many algorithms have been developed for solving CLP instances. But the effort in algorithm-design has not been paralleled by an effort in algorithm analysis — which could provide a much-needed formal basis for intelligent problem-solving decision-making. In this paper we present and analyze three important CLP algorithms: Backtracking, Forward Checking and word-wise Forward Checking. The results, besides their obvious use in choosing between algorithms, capture for the first time the search-order dependent effects that are so important for CLP instances, and as such are capable of providing theory-based guidance in selecting good search orderings. In addition, due to the use of "homogeneous" classes underlying the analyses, the *expected-case* results for classes can be used as virtually *exact-case* complexities for individual instances, and may be used predictively as such. Besides allowing theory-based algorithm selection and search-order selection to thus be done on an instance-by-instance basis, this instance specificity even allows (apparently for the first time) theory-based representation selection for individual instances (as seen in [18]). To exploit to the full these possibilities for instance-specific theory-based guidance, the results of this paper are for a version of CLP that is far more general than that usually considered. This means that instance-specific guidance can be had for arbitrary, *realistic* instances that arise in practice rather than only for the special-form instances usually studied — in particular, studied analytically in [11], [21], [22].

1. INTRODUCTION

The Consistent Labeling (or Constraint Satisfaction) Problem has long been of interest in Artificial Intelligence and Operations Research. Yet an adequate theoretical understanding has remained lacking — in spite of its great potential for improving problem-solving efficiency, by providing a formal basis for intelligent decision-making.

A detailed presentation of CLP (and various partitions and probability models) appears in [19]. It is formulated there far more generally than is usual in systematic studies — whether theoretical or empirical. This generality is maintained in all the complexity analyses of the present paper. The main reason we have sought such generality is because of the ability of our *expected-case* values to approximate well *exact-case* values for individual instances. Since we can thus predict complexity for individual instances, we wish to take advantage of this ability to the full by having results that cover a problem class general enough to include all the various CLP instances that arise in real-world contexts. Our analytic results then become practical problem-solving tools, since they can guide decision-making on an instance-specific basis for arbitrary real-world instances. In particular — since we analyze here several CLP algorithms, as a function of search-order used, and with respect to homogeneous underlying classes — the results (as seen in [18]) may be used to provide effectively instance-specific, theory-based guidance for algorithm-selection, search-order-selection and even for problem-representation selection.

The development of CLP algorithms has received intense attention within Artificial Intelligence, as exemplified by [2], [3], [5], [6], [8]-[10], [12], [15]-[17], [24], [25]. The Ph.D. thesis [7] of Gaschnig is especially important in this regard, as is Haralick's paper [11] which contains an empirical comparison of seven different CLP algorithm. The algorithms analyzed here are generalizations of three of the seven compared by Haralick: Backtracking, Forward Checking and bit-parallel Forward Checking. The first of these was chosen because of its historical importance and its simplicity; the latter two because they were found by Haralick to be the best two of the seven he studied.

The following subsections briefly review some of the material from [19] needed as background for the present paper. The reader is encouraged to refer to [19], or to [18], for more details. Simplified

[3] The word *problem* will be used in two senses in this work. The first sense refers to the usual kind of problem — for which a specific answer is sought to some question about some specific situation (or problem structure). The second sense refers to a *class* of problems of the first kind (usually obtained by allowing the problem structure to vary by use of a few parameters). To reduce ambiguity we will often use *Problem* (with an upper case P) when referring to such a class, and *problem* (with a lower case p), *Problem instance* or just *instance* when referring to a class member. (This convention is not always adhered to, but the intended sense will generally be clear from context.) Analogously we use CLP for the class of problems we are interested in here, while *clp* will refer to an individual member of CLP.

versions of this work have appeared in [20]-[22].

1.1. CLP Instances

An instance of CLP requires finding all ways to instantiate variables to values from their associated domains so that a given set of constraints is satisfied. The following instance, which we call clp_0, reflects much of the generality of structure that we allow in the present formulation. It was introduced in [19] and will be used as a running example below. Consider the problem of assigning values to the three variables z_1, z_2 and z_3 from their respective value domains {0 1}, {0 1} and {0 1 2}, in such a way that the following four constraints C_1 to C_4 are satisfied.

$$C_1: \quad z_1 \vee z_2 \tag{1}$$

$$C_2: \quad (z_3 - 3)^2 + (z_1 - 4)^2 \leq 25 \tag{2}$$

$$C_3: \quad \binom{z_3+1}{z_1} = z_1 + 2 \tag{3}$$

$$C_4: \quad \exp|z_1 + z_2 + z_3| > 1.0 \tag{4}$$

As in this example, we do not require that the variables of a CLP instance all have the same number of candidate values. Also, as with C_4 of clp_0, we do not restrict constraints to be binary (i.e. having only two argument variables). Neither need the constraints of an instance all have the same number of argument variables (compare C_1 and C_4) and there may be more than one constraint over the same set of arguments (C_2 and C_3). Some subsets of variables (such as $\{z_2\ z_3\}$) may have no constraint over them, and this will be explicitly modeled rather than using the usual unrealistic expedient of modeling such "missing constraints" by universal constraints.

These aspects of generality are captured in the following formalization of a CLP instance. A relatively simple canonical form will suffice because for the algorithms and their complexity measures of interest here, all instances that have the same canonical form also have the same complexity of solution. A canonical-form CLP instance $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$ is a five-tuple $(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$[4], where:

- n is the number of problem variables. These variables we denote z_i, and the set of all problem variables is $Z = \{ z_1\ z_2\ .\ .\ z_n \}$.
- $\mathbf{m} = (m_{z_1}\ m_{z_2}\ .\ .\ m_{z_n})$ is the vector of domain sizes m_{z_i}, where m_{z_i} is the number of candidate values in the domain d_{z_i} for variable z_i. The domain itself is denoted $d_{z_i} = \{ z_{i1}\ z_{i2}\ .\ .\ z_{im_{z_i}} \}$, z_{ij} being the j-th candidate value for z_i. We will also use $\overline{z}_i$ to denote an arbitrary value of z_i from d_{z_i}, and $\mathbf{d} = (d_{z_1}\ d_{z_2}\ .\ .\ d_{z_n})$ to denote the vector of all n domains.
- c is the number of constraints.
- A (not necessarily canonical) constraint C_j is some way of specifying which tuples of values for a certain argument set $Z_j \subseteq Z$ of variables are mutually compatible. The algorithms discussed later may have such constraints specified extensively as tables, or intensively as subprograms. But all that counts for the complexity of solving a CLP instance is the canonical form $C_j = (Z_j\ T_j)$ for each constraint C_j. The first component, $Z_j \subseteq Z$, is the set of argument variables which C_j constrains. The second component T_j is the relation induced by constraint C_j; that is, T_j is the set of value tuples that satisfies C_j, where values are of course chosen for each argument variable z_i from its respective domain d_{z_i}. Thus, denoting by $D_j = \bigtimes_{z_i \in Z_j} d_{z_i}$ the cartesian product of the domains of the argument variables of constraint C_j, we have that $T_j \subseteq D_j$ consists of all value tuples $\overline{Z}_j \in D_j$ that satisfy C_j.
- The two parameters $\mathbf{Z}$ and $\mathbf{T}$ of the generic CLP instance $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$ are the vectors of constraint argument sets Z_j and of constraint relations T_j respectively

[4] Note that parameters n and c are actually redundant since $n = |\mathbf{m}|$ and $c = |\mathbf{Z}| = |\mathbf{T}|$. They are however retained for clarity.

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

A **consistent labeling** is an assignment of values to all n problem variables which satisfies all c constraints. The goal in solving a CLP instance is to find all consistent labelings. That is, the goal is to find the set

$$T = \{\ \bar{Z}\ \mid\ \bar{Z} \in D \text{ and } \bar{Z}_j(\bar{Z}) \in T_j,\ \ 1 \le j \le c\ \} \tag{5}$$

where $D = \underset{z_i \in Z}{\times} d_{z_i}$ is the cartesian product of the domains of all problem variables, and $\bar{Z}_j(\bar{Z})$ denotes the projection of the value-tuple $\bar{Z}$ onto the argument set Z_j.[5]

The following are some other parameters of a CLP instance that will be useful. Table 1-1 and figure 1-1 make these and earlier parameters more concrete, by giving their values for the particular case of our running example clp_0.

- We use $M_j = \mid D_j \mid$ to denote the size of the cartesian product of (the domains of the argument variables of) the j-th constraint. Of course, from the definition of D_j we have that $M_j = \prod_{z_i \in Z_j} m_{z_i}$.
- We use $S_j = \mid T_j \mid$ to denote the size of the relation T_j induced by the j-th constraint. This is called the **satisfiability** or **looseness** of constraint C_j,[6] since, by definition of T_j, it gives the number of value-tuples from D_j that satisfy C_j. The vector of satisfiabilities for the c problem constraints is denoted $\mathbf{S} = (S_1\ S_2\ .\ .\ S_c)$.

  Constraint satisfiability is a key parameter in the present work, as it allows us to analyze CLP in terms of classes (called **small-classes**) that are largely homogeneous. As a result, expected-case complexities for a small-class are good estimates for the complexities of individual instances in the class, and may thus provide virtually instance-specific information.
- The sum $SS = \sum_{1 \le j \le c} S_j$ of the satisfiabilities for the c constraints of an instance is called the **summed satisfiability**.
- The ratio $R_j = \mid T_j \mid / \mid D_j \mid = S_j / M_j$ is the **satisfiability ratio** of constraint C_j. Note that for any CLP instance we have that

$$\emptyset \subseteq T_j \subseteq D_j \qquad 0 \le S_j \le M_j \qquad 0 \le R_j \le 1 \qquad 0 \le SS \le \sum_{j=1}^{c} M_j \tag{6}$$

- The number of argument variables for constraint C_j is $A_j = \mid Z_j \mid$, called the **arity** of constraint C_j. Constraints with $A_j = 2$ and $A_j = 3$ arguments are called respectively **binary** and **ternary** constraints. Most CLP work in Artificial Intelligence has treated instances all of whose constraints are binary.

1.2. CLP and its Sub-Problems

Having defined the generic CLP instance $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$, the class CLP is defined as the set of all actual instances obtained from the generic instance as its parameters range over their respective allowed values. Specifically, the number of variables may be any integer $n > 1$, each component domain size of $\mathbf{m}$ may independently be any integer $m_{z_i} > 0$ and the number of constraints may be any integer $c > 0$. Each component constraint argument set of $\mathbf{Z}$ may independently be any subset $Z_j \subseteq Z$ (containing at least two variables) of the set of problem variables Z, and each component constraint relation of $\mathbf{T}$ may independently be any subset $T_j \subseteq D_j$ of the set D_j of possible value-tuples for the constraint argument variables.

Two types of sub-Problem of CLP are particularly important for our analyses. These we call **big-classes** and **small-classes.** CLP is partitioned by either type of class, but small-classes form a finer partition, and have the important property of homogeneity as a result of which small-class expectations provide effectively instance-specific predictions. Results are given for special-form big-classes in [11], [22] and

---

[5] For example, if $\bar{Z} = (g\ s\ 37\ e\ 12)$ for the variables of $Z = \{z_1\ z_2\ z_3\ z_4\ z_5\}$ and if $Z_j = \{z_2\ z_4\ z_5\}$ then the projection of $\bar{Z}$ onto Z_j is $\bar{Z}_j(Z) = (s\ e\ 12)$. We will use the convention that in tuples Z_j, values $\bar{z}_i$ are arranged left to right so that their associated variables are in order of increasing index.

[6] In [20] and [21] it was called the compatibility of the argument variables of C_j. Volume or simply size of constraint C_j would also be appropriate names for parameter S_j.

Table 1-1: CLP Instance clp_0 — our running example.

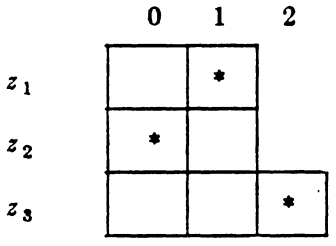
Variables: $n = 3$

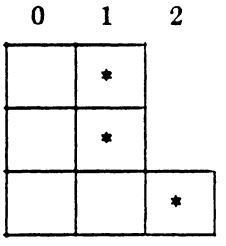
i	z_i	d_{z_i}	m_{z_i}
1	z_1	{ 0 1 }	2
2	z_2	{ 0 1 }	2
3	z_3	{ 0 1 2 }	3

Constraints: $c = 4$

j	Z_j	T_j	A_j	M_j	S_j	R_j
1	$\{z_1\ z_2\}$	{ (01) (10) (11) }	2	4	3	3/4
2	$\{z_1\ z_3\}$	{ (00) (01) (02) (10) (11) (12) }	2	6	6	1
3	$\{z_1\ z_3\}$	{ (12) }	2	6	1	1/6
4	$\{z_1\ z_2\ z_3\}$	{ (001) (002) (010) (011) (012) (100) (101) (102) (110) (111) (112) }	3	12	11	11/12

Fig. 1-1: The two solutions of instance clp_0, and their projections onto the various constraint argument sets.





$\bar{Z}$:	(1 0 2)	(1 1 2)
$\bar{Z}_1(\bar{Z})$:	(1 0)	(1 1)
$\bar{Z}_2(\bar{Z})$:	(1 2)	(1 2)
$\bar{Z}_3(\bar{Z})$:	(1 2)	(1 2)
$\bar{Z}_4(\bar{Z})$:	(1 0 2)	(1 1 2)

[21]. The latter paper also treats special-form small-classes. In section 3 below we present the recently derived results for fully *arbitrary* big- and small-classes.

In his thesis [7], Gaschnig, besides also using (special-form) big-classes, studied CLP empirically using yet another type of CLP sub-Problem which might be called the (special-form) **middle-sized class.** His results show that middle-sized classes, like big-classes, do not have the desirable homogeneity property, and on page 241 of [7] he proposes small-classes as an improvement. Small-classes were however independently arrived at in this work, as a refinment of Haralick's big-classes. The definitions of these classes are as follows:

The generic big-class $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ is the set of all CLP instances having the generic values n, $\mathbf{m}$, c and $\mathbf{Z}$ respectively for the number of variables, vector of domain sizes, number of constraints and vector of constraint-argument sets. The component constraint-relations T_j of an instance's constraint-relation vector $\mathbf{T}$ are arbitrary subsets of the corresponding cartesian products D_j.

$$\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z}) = \{clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T}) \mid T_j \subseteq D_j \quad 1 \le j \le c\} \tag{7}$$

The generic middle-sized class $\mathrm{CLP}_\mu(n\ \mathbf{m}\ c\ \mathbf{Z}\ SS)$ is the set of all CLP instances having the generic values n, $\mathbf{m}$, c and $\mathbf{Z}$ for the corresponding features. In addition, the constraint-relations T_j of an instance are such that the sum of their sizes is SS.

$$\mathrm{CLP}_\mu(n\ \mathbf{m}\ c\ \mathbf{Z}\ SS) = \{clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T}) \mid \sum_{j=1}^{c} |T_j| = SS\} \tag{8}$$

The generic small-class $\mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$ is the set of all CLP instances having the generic values n, $\mathbf{m}$, c and $\mathbf{Z}$ for the corresponding features. In addition, each constraint-relation T_j of an instance has size S_j, the j-th component of the small-class constraint-satisfiability vector $\mathbf{S}$.

$$\mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S}) = \{clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T}) \mid |T_j| = S_j \quad 1 \le j \le c\} \tag{9}$$

The difference between the above three types of CLP sub-Problem is the degree to which the constraint-satisfiabilities S_j are restricted. An orthogonal kind of restriction of historic interest in inducing sub-Problems of CLP is one placed on $\mathbf{Z}$, the vector of argument-sets. It is often assumed that instances are **binary**, that is, have constraints only over *pairs* of variables. Moreover, it is often assumed, as is the case for n-queens instances (under the usual formulation), that *all* possible pairs of variables have exactly one constraint over them. Instances with *exactly* one constraint over each pair of variables, and having no other constraints, we call **pure and simple binary.** If there is *at least* one constraint for each pair of variables, and no other constraints, such instances are called **pure** (but not necessarily simple) **binary.**

Most systematic studies in Artificial Intelligence have considered only pure and simple binary instances. In particular this is the case in [11] where pure and simple binary big-classes are used, in [7] where pure and simple binary big- and middle-sized classes are used, and in [21] where pure and simple binary big- and small-classes are used.[7] The next level of generality occurs in [22] where we consider **pure and simple A-ary** big-classes — big-classes of instances having exactly one constraint over each combination of A variables, and no other constraints. (If each combination of size 2 through A variables has exactly one constraint, and no other constraints exist, such instances are called **full and simple A-ary**.) No other analysis had ever treated that general a case before. However, our results of section 3 below are far more general yet, being in terms of fully arbitrary small-classes and big-classes, with no restriction at all placed on the vector of argument sets Z, characterizing instances of the class. In particular, each constraint of an instance — independently of the other constraints — may be over an arbitrary subset Z_j of the problem variables Z, and hence each constraint may individually be of arbitrary arity $A_j = |Z_j|$. As a result, some subsets of variables may in fact have no constraint whereas other subsets may have more than one constraint over them. In addition, the domains of problem variables need not be the same, nor need they be of the same size.

1.3. Probability Models for CLP

This section introduces several probability models for the distribution of instances in the CLP small-classes and big-classes of the previous section. It is under these probability models that the

[7] In [21] the (pure and simple binary) big-classes and small-classes are called respectively, v-classes and c-classes.

expected-complexity of CLP problem-solving will be derived in section 3. Certain useful events in big-classes and small-classes are defined and their probabilities are given under the corresponding probability models. Note that the results of this section are in an algorithm-independent form. Section 3 will use them as the basis of the algorithm-dependent results derived there.

1.3.1. Some Notation

This section presents some notation and results that will be useful. This paper assumes a familiarity with the theory of probability for finite sample spaces, for which useful references are [4], [23] and [13].

Probability Spaces: We will be associating probability models with CLP big-classes and small-classes. To emphasize that these big- and small-classes then become sample-spaces — where CLP instances are outcomes — we write $\Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ and $\Omega_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$, or more simply Ω_β and Ω_σ, respectively for $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ and $\mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$. An instances $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$ or more simply clp, may accordingly be denoted as $\omega(n\ m\ c\ \mathbf{Z}\ \mathbf{T})$ or ω. When the distinction between big-class Ω_β and small-class Ω_σ is not important we use Ω to stand for either one.

We will often be discussing parameterized events $E = E(a_1 \ldots a_m)$ in Ω, where the a_i are arguments whose values determine the specific event intended. In this case, we abbreviate probability $P(E(a_1 \ldots a_m))$ of the event by $P(a_1 \ldots a_m)$. Of course, for events E_j, $j \in J$ that are mutually independent or mutually exclusive we have respectively

$$P(\bigcirc\!\!\!\!\cdot_{j \in J} E_j) = \prod_{j \in J} P(E_j) \quad \text{and} \quad P(\bigcup\!\!\!\!\cdot_{j \in J} E_j) = \sum_{j \in J} P(E_j) \tag{10}$$

where we use $\bigcup\!\!\!\!\cdot$ to denote the union over mutually disjoint sets (mutually exclusive events) and $\bigcirc\!\!\!\!\cdot$ to denote the intersection of mutually independent events. We will use the following definition of the expected value of a random variable $Q(\omega)$

$$\bar{Q} = \sum_{\omega \in \Omega} Q(\omega) P(\omega) \tag{11}$$

which is equivalent to the more usual definition as a sum over all possible values q of $Q(\omega)$, each value weighted by its probability $P(q)$.

Indicator Functions: A given event $E \subseteq \Omega$ can be represented by its **characteristic** or **indicator function**

$$\delta(E\ \omega) = \begin{cases} 1 & \text{if } \omega \in E \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

When the event E is given parametrically as above, then we abbreviate $\delta(E(a_1 \ldots a_m)\ \omega)$ as $\delta(a_1 \ldots a_m\ \omega)$. A result that will prove useful is that the expected value of the indicator function of an event E is equal to the probability of that event:

$$\bar{\delta}(E) = \sum_{\omega \in \Omega} \delta(E\ \omega)\ P(\omega) = \sum_{\omega \in E} P(\omega) = P(E) \tag{13}$$

Set Notation: We will also use $\binom{Z}{m}$, where Z is a set, to denote the *set* of all size-m subsets of Z. If $|Z| = n$, then of course $\left|\binom{Z}{m}\right| = \binom{n}{m}$. Note the analogy with $|2^Z| = 2^n$ for the size of the power set of Z. Lastly, we use J_n^m for the set of integers from n to m inclusive.

1.3.2. Probability Models over Big-Classes and Small-Classes

Within a given big-class or small-class, the instances $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$ differ only in the value of their constraint-relation vector $\mathbf{T} = (T_1\ T_2 \ldots T_c)$. Thus a probability model for the instances of a big-class or small-class is equivalent to one for the corresponding set of $\mathbf{T}$ vectors. By the definition of big-class and small-class (in section 2.5) we have that the relation-vectors $\mathbf{T}$ for the instances in a big-class $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ and in a small-class $\mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$ are respectively those given by

$$\mathbf{T} \in \bigtimes_{j=1}^{c} 2^{D_j} \quad \text{and} \quad \mathbf{T} \in \bigtimes_{j=1}^{c} \binom{D_j}{S_j} \tag{14}$$

The following are three probability models for the distribution of these **T** vectors. Models 0 and 1 induce a distribution on the vectors given in the first part of (14), and hence induce a distribution on the CLP instances of a big-class. Model-2 induces a distribution on the vectors given in the second part of (14), and hence induces a distribution on the instances of a small-class.

Table 1-2: CLP Probability Models Used

In all three models, the vector $\mathbf{T} = (T_1\ T_2\ .\ .\ T_c)$ of constraint relations arises by c independent experiments, one for each component T_j.

0 **Model-0 on a big-class:** A given T_j arises as the result of M_j independent experiments, each determining for a different one of the value-tuples $\bar{Z}_j \in D_j$ whether or not it is in T_j. Each $\bar{Z}_j$ has probability p of belonging to T_j.

1 **Model-1 on a big-class:** As for model-0, but generalizes so that each $\bar{Z}_j$ in D_j has probability p_j of belonging to T_j.

2 **Model-2 on a small-class:** A given T_j arises by a single experiment of randomly selecting a subset of S_j value-tuples from D_j. All subsets of size S_j are equally likely.

We use $P_\beta^1(E\ ;\ \mathbf{p})$ to denote the probability that event $E \subseteq \Omega$ occurs when an instance is randomly selected from big-class $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ according to model-1 with satisfiability-rate vector $\mathbf{p} = (p_1\ .\ .\ p_c)$. The corresponding expression for model-0 is $P_\beta^0(E\ ;\ p)$. We use $P_\sigma^2(E)$ to denote the probability that event E occurs when a CLP instance is randomly selected from a small-class according to model-2. For big-classes under model-0 and under model-1, and for small-classes under model-2, the expected value of a random variable $Q(clp)$ is given by the corresponding versions of (11), which are respectively

$$\bar{Q}_\beta^0(n\ \mathbf{m}\ c\ \mathbf{Z}\ ;\ p) = \sum_{clp\ \in\ \Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})} Q(clp)\ P_\beta^0(clp\ ;\ p) \tag{15}$$

$$\bar{Q}_\beta^1(n\ \mathbf{m}\ c\ \mathbf{Z}\ ;\ \mathbf{p}) = \sum_{clp\ \in\ \Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})} Q(clp)\ P_\beta^1(clp\ ;\ \mathbf{p}) \tag{16}$$

$$\bar{Q}_\sigma^2(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S}) = \sum_{clp\ \in\ \Omega_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})} Q(clp)\ P_\sigma^2(clp) \tag{17}$$

1.3.3. Events and their Probabilities

Table 1-3 gives events that will be needed later when deriving the expected complexity of solving CLP instances. The events are with respect to a big-class or a small-class as the sample-space. In [18] and [19] the corresponding event probabilities have been determined under model-1 and model-2, and these probabilities are summarized in table 1-4. Note that all events and probabilities of this section are independent of any algorithm, and reflect only the nature of CLP itself and the corresponding probability model. In section 3 below the results of this section will be used to determine algorithm-dependent probabilities and expectations.

As indicated in table 1-3, the events we consider can be divided into three types, based on (i) whether constraints of an instance are satisfied by a given value-tuple, (ii) whether constraint relations of an instance are equal to given sets and (iii) whether constraint relations of an instance have a certain size. The following paraphrases and expands on the formal specification of the events in table 1-3.

- $E(j\ \bar{Z}_j)$ is the set of instances in Ω whose j-th constraint relation $T_j(clp)$ is satisfied by a given value-tuple $\bar{Z}_j$ from the corresponding cartesian product.
- $E(j\ \bar{X})$ is the set of instances in Ω whose j-th constraint relation $T_j(clp)$ is satisfied by a given value-tuple $\bar{X}$ labeling $X \subseteq Z$. X need not equal the list Z_j of argument variables for T_j, but may

Table 1-3: Some Basic Subsets of $\Omega = \Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ or $\Omega_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$.
(The corresponding event probabilities are in table 1-4.)

Symbol	Defined as	Defined for
Events based on constraint satisfaction		
$E(j\ \bar{Z}_j)$	$\{clp \mid \bar{Z}_j \in T_j(clp)\}$	$\bar{Z}_j \in D_j,\ j \in J_1^c$
$E(j\ \bar{X})$	$\{clp \mid \bar{Z}_j(\bar{X}) \in T_j(clp)\}$	$X \subseteq Z,\ Z_j \subseteq X,\ j \in J_1^c$
$E(J\ \bar{X})$	$\{clp \mid \bar{Z}_j(\bar{X}) \in T_j(clp)\ \forall\ j \in J\}$	$X \subseteq Z,\ J \subseteq J_1^c,\ Z_j \subseteq X\ \forall\ j \in J$
$E(J\ f\ t\ \bar{X})$	$\{clp \mid \exists$ exactly t values $\bar{f} \in d_f$ s.t. $[\bar{Z}_j(\bar{X}\ \bar{f}) \in T_j(clp)\ \forall\ j \in J]\}$	$X \subset Z,\ J \subseteq J_1^c,\ f \notin X,$ $\{f\} \subseteq Z_j \subseteq X \cup \{f\}\ \forall\ j \in J,\ t \in J_0^{m_f}$
$E(J\ f\ \geq 1\ \bar{X})$	$\{clp \mid \exists$ at least one value $\bar{f} \in d_f$ s.t. $[\bar{Z}_j(\bar{X}\ \bar{f}) \in T_j(clp)\ \forall\ j \in J]\}$	$X \subset Z,\ J \subseteq J_1^c,\ f \notin X,$ $\{f\} \subseteq Z_j \subseteq X \cup \{f\}\ \forall\ j \in J$
Events based on constraint equality		
$E(j\ T_j)$	$\{clp \mid T_j(clp) = T_j\}$	$\begin{cases} T_j \in 2^{D_j} & \text{Model-1} \\ T_j \in \binom{D_j}{S_j} & \text{Model-2} \end{cases}$ $j \in J_1^c$
$E(\mathbf{T})$	$\{clp \mid \mathbf{T}(clp) = \mathbf{T}\}$	$\begin{cases} \mathbf{T} \in \bigtimes_{j=1}^{c} 2^{D_j} & \text{Model-1} \\ \mathbf{T} \in \bigtimes_{j=1}^{c} \binom{D_j}{S_j} & \text{Model-2} \end{cases}$
Events based on constraint-size equality		
$E(j\ S_j)$	$\{clp \mid S_j(clp) = S_j\}$	$\begin{cases} S_j \in J_0^{M_j} & \text{Model-1} \\ S_j = S_j(\Omega_\sigma) & \text{Model-2} \end{cases}$ $j \in J_1^c$
$E(\mathbf{S})$	$\{clp \mid \mathbf{S}(clp) = \mathbf{S}\}$	$\begin{cases} \mathbf{S} \in \bigtimes_{j=1}^{c} J_0^{M_j} & \text{Model-1} \\ \mathbf{S} = \mathbf{S}(\Omega_\sigma) & \text{Model-2} \end{cases}$

subsume it. In this case, T_j is satisfied by $\bar{X}$ if the projection $\bar{Z}_j(\bar{X})$ of $\bar{X}$ onto Z_j is in T_j.

- $E(J\ \bar{X})$ is the set of instances in Ω whose constraints in the set indexed by J are all satisfied by a given value-tuple $\bar{X}$ labeling $X \subseteq Z$.
- $E(J\ f\ t\ \bar{X})$ is the set of instances in Ω for which exactly t of the m_f possible values in domain d_f for variable f each allow all the constraints that are indexed by J to be satisfied when other argument variables take their values as given by $\bar{X}$. Thus $E(J\ f\ t\ \bar{X})$ can be viewed as the event that exactly t values for f each "survive" all the constraints in $T_J = \{T_j \mid j \in J\}$ given the instantiations in $\bar{X}$. Note that as indicated in table 1-3, this set of instances is well defined only for f not in X and when each constraint in T_J has f as one argument and takes its other arguments only from the variables X. This and the next set of instances will be important in the analysis of algorithm gFC.
- $E(J\ f\ \geq 1\ \bar{X})$ is as for $E(J\ f\ t\ \bar{X})$ except that instead of requiring exactly t surviving values from d_f, we require at least 1.
- $E(j\ T_j)$ is the set of instances in Ω whose j-th constraint relation $T_j(clp)$ is equal to a given relation T_j.
- $E(\mathbf{T})$ is the set of instances in Ω whose vector $\mathbf{T}(clp) = (T_1(clp)\ .\ .\ T_c(clp))$ of constraint relations is equal to a given relation vector $\mathbf{T} = (T_1\ .\ .\ T_c)$. As mentioned, within a given big or small-class there is a one-to-one correspondence between vectors $\mathbf{T}$ and instances $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$, so that the set $E(\mathbf{T})$ is really just the singleton set $\{clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})\}$. As a consequence, the probability of event $E(\mathbf{T})$ is the probability of outcome $clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T})$.
- $E(j\ S_j)$ is the set of instances in Ω for which the size $S_j(clp) = |\ T_j(clp)\ |$ of the j-th constraint relation $T_j(clp)$, is equal to a given number S_j.
- $E(\mathbf{S})$ is the set of instances in Ω whose vector $\mathbf{S}(clp) = (S_1(clp)\ .\ .\ S_c(clp))$ of constraint relation sizes, is equal to a given vector $\mathbf{S} = (S_1\ .\ .\ S_c)$ of sizes. In table 1-3, $\mathbf{S}(\Omega_\sigma)$ denotes the satisfiability vector $\mathbf{S}$ characterizing the small-class $\Omega_\sigma = \mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$, and $S_j(\Omega_\sigma)$ denotes its j-th component.

In addition, we will need the following two results from [19] for the expected value of t under distribution $P(J\ f\ t\ \bar{X})$ for models 1 and 2 respectively:

$$\sum_{t=0}^{m_f} t\ P_\beta^1(J\ f\ t\ \bar{X}\ ;\ \mathbf{p}) = m_f \prod_{j \in J} p_j \tag{18}$$

$$\sum_{t=0}^{m_f} t\ P_\sigma^2(J\ f\ t\ \bar{X}) = m_f \prod_{j \in J} R_j \tag{19}$$

Table 1-4: Probabilities of the Events in Table 1-3

Prob. Model	1	2
Ω	$\Omega_\beta = \mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$	$\Omega_\sigma = \mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$
$P(E)$	$P_\beta^1(E\ ;\ \mathbf{p})$	$P_\sigma^2(E)$
$P(j\ \bar{Z}_j)$	p_j	R_j
$P(j\ \bar{X})$	p_j	R_j
$P(J\ \bar{X})$	$\prod_{j\in J} p_j$	$\prod_{j\in J} R_j$
$P(J\ f\ t\ \bar{X})$	$\binom{m_f}{t}\left[\prod_{j\in J} p_j\right]^t\left[1-\prod_{j\in J} p_j\right]^{m_f-t}$	$\dfrac{\binom{m_f}{t}\sum_{l=0}^{m_f-t}(-1)^l\binom{m_f-t}{l}\prod_{j\in J}\binom{M_j-t-l}{S_j-t-l}}{\prod_{j\in J}\binom{M_j}{S_j}}$
$P(J\ f\ \geq 1\ \bar{X})$	$1-\left[1-\prod_{j\in J} p_j\right]^{m_f}$	$1-\dfrac{\sum_{l=0}^{m_f}(-1)^l\binom{m_f}{l}\prod_{j\in J}\binom{M_j-l}{S_j-l}}{\prod_{j\in J}\binom{M_j}{S_j}}$
$P(j\ T_j)$	$p_j^{\,S_j}(1-p_j)^{M_j-S_j}$	$\binom{M_j}{S_j}^{-1}$
$P(\mathbf{T}) =$ $P(clp(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{T}))$	$\prod_{j=1}^{c} p_j^{\,S_j}(1-p_j)^{M_j-S_j}$	$\prod_{j=1}^{c}\binom{M_j}{S_j}^{-1}$
$P(j\ S_j)$	$\binom{M_j}{S_j} p_j^{\,S_j}(1-p_j)^{M_j-S_j}$	1.0
$P(\mathbf{S})$	$\prod_{j=1}^{c}\binom{M_j}{S_j} p_j^{\,S_j}(1-p_j)^{M_j-S_j}$	1.0

2. CLP ALGORITHMS

This section presents three algorithms for solving CLP instances: Backtracking, Forward Checking and word-wise Forward Checking. Each of these will be analyzed in section 3. Our versions of these algorithms are generalizations of those in [11] so as to allow solution of the fully general instance of section 1.1 above, as well as to allow flexibility in the choice of instantiation order and constraint-check order. Examples are given of the use of these algorithms, emphasizing the dependence of problem-solving complexity on the particular instantiation order and constraint-check order used.

2.1. Preliminaries

This section lays some groundwork needed for the subsequent presentation of our CLP algorithms. Two notions are introduced, (1) consistency of partial labelings and (2) the instantiation ordering used by an algorithm. The concept of a constraint-check ordering parallels that of instantiation ordering, but is best left to be described in the context of the respective algorithms.

As mentioned, the goal in solving a CLP instance is to find the set T of all labelings $\bar{Z}$ of the problem variables that satisfy all the problem constraints. This solution set was therefore denoted explicitly in (5) as

$$T = \{ \bar{Z} \mid \bar{Z} \in D \text{ and } \bar{Z}_j(\bar{Z}) \in T_j \ \ \forall\, j \in J_1^c \} \tag{20}$$

which indicates that the members of T — the consistent labelings — must be such that their projection onto the argument sets of each constraint is a satisfying tuple for the corresponding constraint. It is convenient to generalize the notion of a labeling and its consistency as follows.

Definition:

A **labeling** of a set $A \subseteq Z$ of variables is a tuple of values from the respective domains of the variables

$$\bar{A} \in D_A = \bigtimes_{z_i \in A} d_{z_i}$$

If A is a proper subset of Z, then $\bar{A}$ is called a **partial labeling** of Z.

Definition:

A constraint $C_j = (Z_j, T_j)$ is said to be **applicable** to a given set A of variables if $Z_j \subseteq A$. The index set for the constraints applicable to A is therefore

$$\Psi_A = \{ j \mid 1 \le j \le c \text{ and } Z_j \subseteq A \} \tag{21}$$

Definition:

$\bar{A}$ is a **consistent** labeling of A if and only if it satisfies each instance constraint C_j applicable to set A. Thus, paralleling the above definition for the solution set T, the set of all consistent labelings of $A \subseteq Z$ is

$$T_A = \{ \bar{A} \mid \bar{A} \in D_A \text{ and } \bar{Z}_j(\bar{A}) \in T_j \ \ \forall\, j \in \Psi_A \} \tag{22}$$

where $\bar{Z}_j(\bar{A})$ denotes the projection of $\bar{A}$ onto Z_j, the argument set of constraint C_j.[8] The solution set T in (20) is a special case of (22) where $A = Z$. A labeling $\bar{A}$ of A that is not in T_A thus violates at least one applicable constraint and is an **inconsistent** labeling of A. Clearly such a labeling cannot be extended to form a consistent labeling of Z.

The CLP algorithms to be studied below will generate the solution set T by recursively extending partial labelings of Z. Starting with the empty labeling, a tree of partial labelings is built up by recursively extending each consistent partial labeling according to the various values that may be assigned to an as-yet unlabeled variable. The order in which variables are instantiated at successive levels of the

[8] For example, if $A = \{z_2\ z_3\ z_4\ z_6\ z_9\}$, $\bar{A} = (e\ a\ e\ g\ b)$ and $Z_j = \{z_2\ z_4\ z_6\}$ then $\bar{Z}_j(\bar{A}) = (e\ e\ g)$. In this example, as earlier, we use the convention that values $\bar{z}_i$ in labelings correspond to variables arranged in name order. However, it will often be convenient to have the values corresponding to variables arranged in instantiation order. All that matters is that for any given labeling it be clear what is the implied correspondence between variables and values.

search tree is called the **instantiation ordering** for variables.

The particular instantiation order used can have a great effect on the complexity of solving a given CLP instance. Analogously, the **constraint-check ordering** (described below) may also have considerable effect on problem-solving complexity. Yet virtually no theoretical basis exists for making appropriate choices for these orderings. One of the main aims of this work has been the derivation of analytic results that capture the effect of both instantiation order and constraint-check order on problem-solving complexity — and to show how these results may be used to make good choices for these orderings. The term **search ordering** will be used to refer to both instantiation orderings and constraint-check orderings.

Explicitly, the instantiation order X is one of the $n!$ possible permutations of the n problem variables in Z. If $\pi = (\pi_1\ \pi_2\ .\ .\ \pi_n)$ denotes a permutation of the integers 1 to n, then

$$X = (x_1\ x_2\ .\ .\ x_n) = (z_{\pi_1}\ z_{\pi_2}\ .\ .\ z_{\pi_n}) \tag{23}$$

Thus, the k-th variable to be instantiated along any path through the search tree generated will be $x_k = z_{\pi_k}$. As usual, $d_{x_k} = d_{z_{\pi_k}}$ denotes the domain of values for variable x_k and $\bar{x}_k \in d_{x_k}$ is an arbitrary such value for x_k. The set of the first k variables instantiated and an arbitrary labeling of this set are denoted respectively by X_k and $\bar{X}_k$, so that

$$X_k = \{x_1\ x_2\ .\ .\ x_k\} \tag{24}$$

$$\bar{X}_k = (\bar{x}_1\ \bar{x}_2\ .\ .\ \bar{x}_k) \in \bigtimes_{x_i \in X_k} d_{x_i} = D_{X_k} \tag{25}$$

This is analogous to the notation Z_j and $\bar{Z}_j$ introduced earlier. However note that in $\bar{X}_k$, values are arranged so that left to right their corresponding variables are in instantiation order while $\bar{Z}_j$ is arranged with corresponding variables in name order. The **concatenation** operator $||$ will be be useful in building up lists of labelings or of variables. If a is the list $(a_1\ a_2\ .\ .\ a_m)$ then $a\ ||\ a_t$ denotes the list $(a_1\ a_2\ .\ .\ a_m\ a_t)$. Thus corresponding to $X_k = X_{k-1} \cup x_k$ we have $\bar{X}_k = \bar{X}_{k-1}\ ||\ \bar{x}_k$, which says that we may form a labeling of X_k by concatenation of a labeling $\bar{X}_{k-1}$ of X_{k-1} and a value $\bar{x}_k$ for variable x_k.

We call x_k the **current variable** at level k and X_k the set of **instantiated variables** at level k. Sets

$$P_k = X_k - \{x_k\} = \{x_1\ x_2\ .\ .\ x_{k-1}\} \quad \text{and} \quad F_k = Z - X_k = \{x_{k+1}\ x_{k+2}\ .\ .\ x_n\}$$

are respectively the **past variables** and the **future variables** at level k. The latter will be particularly important in relation to the Forward Checking algorithm. Note that $F_0 = X_n = Z$, that $F_n = X_0 = \emptyset$, and that $F_k = F_{k-1} - \{x_k\}$.

The above definitions are summarized in table 2-1. For concreteness, table 2-2 shows some of the above quantities for the case of $n = 6$ variables and the instantiation ordering $X = (x_1\ x_2\ x_3\ x_4\ x_5\ x_6) = (z_5\ z_3\ z_1\ z_2\ z_6\ z_4)$ corresponding to permutation $\pi = (5\ 3\ 1\ 2\ 6\ 4)$.

2.2. Generalized Backtracking (gBT)

This section develops the gBT algorithm in terms of a recursive function for the solution set T of a CLP instance. The algorithm is essentially the well-known Backtracking algorithm treated for example in [8], [14] and [11]. But unlike the versions often studied in the literature, gBT explicitly allows arbitrary instantiation ordering and constraint-check ordering. Also, it is able to handle any problem of the general type defined in section 1.1 — including instances which, amongst other things, may have constraints corresponding to an arbitrary family of arities and an arbitrary family of argument sets. Several examples are given of the use of gBT, emphasizing the effect of instantiation order and constraint-check order upon problem-solving complexity.

2.2.1. Development of the Algorithm

Given a CLP instance to solve, and some instantiation order X for its variables in Z, assume that a labeling $\bar{X}_k$ is given for $X_k = \{x_1\ x_2\ .\ .\ x_k\}$ the first k variables in X. The set of all consistent labelings of Z that are extensions of such a labeling $\bar{X}_k$ can be expressed as $F\,1(\bar{X}_k\ k)$ given recursively as follows:

Table 2-1: Some definitions for algorithms gBT, gFC and wgFC

Symbol	Defined as	Defined for
π	$(\pi_1\ \pi_2\ .\ .\ \pi_n)$, a permutation of $\{1\ 2\ .\ .\ n\}$	
x_k	Variable z_{π_k}	$1 \le k \le n$
$\bar{x}_k$	An element of domain d_{x_k}	
X	$(\ x_1\ x_2\ .\ .\ x_n\ )$	
X_k	$\begin{cases} \{\ x_1\ x_2\ .\ .\ x_k\ \} \\ \emptyset \end{cases}$	$1 \le k \le n$ $k=0$
P_k	$\begin{cases} \{\ x_1\ x_2\ .\ .\ x_{k-1}\ \} \\ \emptyset \end{cases}$	$2 \le k \le n$ $k=0,\ 1$
F_k	$\begin{cases} \{\ x_{k+1}\ x_{k+2}\ .\ .\ x_n\ \} \\ \emptyset \end{cases}$	$0 \le k \le n-1$ $k=n$
D_{X_k}	$\bigtimes_{x_i \in X_k} d_{x_i}$	
$\bar{X}_k$	$(\bar{x}_1\ \bar{x}_2\ .\ .\ \bar{x}_k)$, an element of D_{X_k}	
$\bar{Z}_j(\bar{A})$	The projection of $\bar{A}$ onto Z_j	$Z_j \subseteq A \subseteq Z$

Table 2-2: An example with instantiation ordering $X = (z_5\ z_3\ z_1\ z_2\ z_6\ z_4)$.

k	x_k	X_k	F_k	D_{X_k}
0		$\emptyset$	$\{\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ \}$	$\emptyset$
1	z_5	$\{\ z_5\ \}$	$\{\ z_1\ z_2\ z_3\ z_4\ z_6\ \}$	d_{z_5}
2	z_3	$\{\ z_5\ z_3\ \}$	$\{\ z_1\ z_2\ z_4\ z_6\ \}$	$d_{z_5} \times d_{z_3}$
3	z_1	$\{\ z_5\ z_3\ z_1\ \}$	$\{\ z_2\ z_4\ z_6\ \}$	$d_{z_5} \times d_{z_3} \times d_{z_1}$
4	z_2	$\{\ z_5\ z_3\ z_1\ z_2\ \}$	$\{\ z_4\ z_6\ \}$	$d_{z_5} \times d_{z_3} \times d_{z_1} \times d_{z_2}$
5	z_6	$\{\ z_5\ z_3\ z_1\ z_2\ z_6\ \}$	$\{\ z_4\ \}$	$d_{z_5} \times d_{z_3} \times d_{z_1} \times d_{z_2} \times d_{z_6}$
6	z_4	$\{\ z_5\ z_3\ z_1\ z_2\ z_6\ z_4\ \}$	$\emptyset$	$d_{z_5} \times d_{z_3} \times d_{z_1} \times d_{z_2} \times d_{z_6} \times d_{z_4}$

$$F1(\overline{X}_k\ k) = \begin{cases} \emptyset \text{ if } \overline{X}_k \text{ is inconsistent} \\ \{\ \overline{X}_k\ \} \text{ if } \overline{X}_k \text{ is consistent and } k = n \\ \bigcup_{\overline{x}_{k+1} \in d_{x_{k+1}}} F1(\overline{X}_k\ ||\ \overline{x}_{k+1}\ k+1) \quad \text{otherwise} \end{cases} \tag{26}$$

In words, the set of consistent labelings of Z formable by extending labeling $\overline{X}_k$ is

- empty if $\overline{X}_k$ is itself not a consistent labeling of $X_k \subseteq Z$,
- the set containing only $\overline{X}_k$ itself, if $\overline{X}_k$ is consistent and of length n, since then all variables of Z are already consistently labeled
- otherwise, the union of all sets of consistent labelings of Z formable as extensions of the various extensions of $\overline{X}_k$ which result from appending to $\overline{X}_k$ the different possible values for the next variable to instantiate, x_{k+1}.

Since the solution set T for an instance is just the set of consistent labelings of Z that can be formed by extension of the empty labeling $\overline{X}_k = \emptyset$ of length $k = 0$, we have that

$$T = F1(\emptyset\ 0) \tag{27}$$

Figure 2-1 shows half of the symmetrical tree of recursive calls made in evaluating (27) for the case of the 4-queens problem[9] under the usual formulation in which a CLP variable is associated with each row of the chess board and a domain value with each column of the board. In this example, we are using the name order for variables in Z as the instantiation order X; in other words $x_i = z_i$ so that

$$X = (x_1\ x_2\ x_3\ x_4) = (z_1\ z_2\ z_3\ z_4). \tag{28}$$

A node is drawn in the search tree of figure 2-1 for each recursive call to $F1$. The node corresponding to call $F1(\overline{X}_k\ k)$ is called the node at $\overline{X}_k$, or simply node $\overline{X}_k$. Each node $\overline{X}_k$ of a tree shows the corresponding value of $\overline{X}_k$ using an array representation where rows of a node correspond to CLP problem variables and columns to values for those variables (with the particular correspondence as indicated at the root node). An instantiation of variable x_i to some value $\overline{x}_i$ is shown by darkening the corresponding cell in the array.

Note that for q-queens instances under the standard formulation, this representation scheme for $\overline{X}_k$ will simply give a "picture" of the corresponding placement of the first k queens on the board.[10] This makes the search trees for n-queen instances particularly easy to understand and is the reason that 4-queens is used here (and later for Forward Checking) as an introductory example. However, this pictorial property of our node representation is just an artifact of n-queens problems and the particular CLP formulation used here for them. In general the above representation for nodes is nothing more than a convenient display of the state of the solution, and has no particular geometrical significance.

Now in computing the solution set T using the above recursive function $F1$ of (26), it is necessary to check for each call $F1(\overline{X}_k\ k)$ whether $\overline{X}_k$ is consistent or not. From section 2.1, this means checking whether $\overline{X}_k$ satisfies all applicable constraints: those whose indices are in

$$\Psi_{X_k} = \{\ j\ |\ 1 \le j \le c \text{ and } Z_j \subseteq X_k\ \} \tag{29}$$

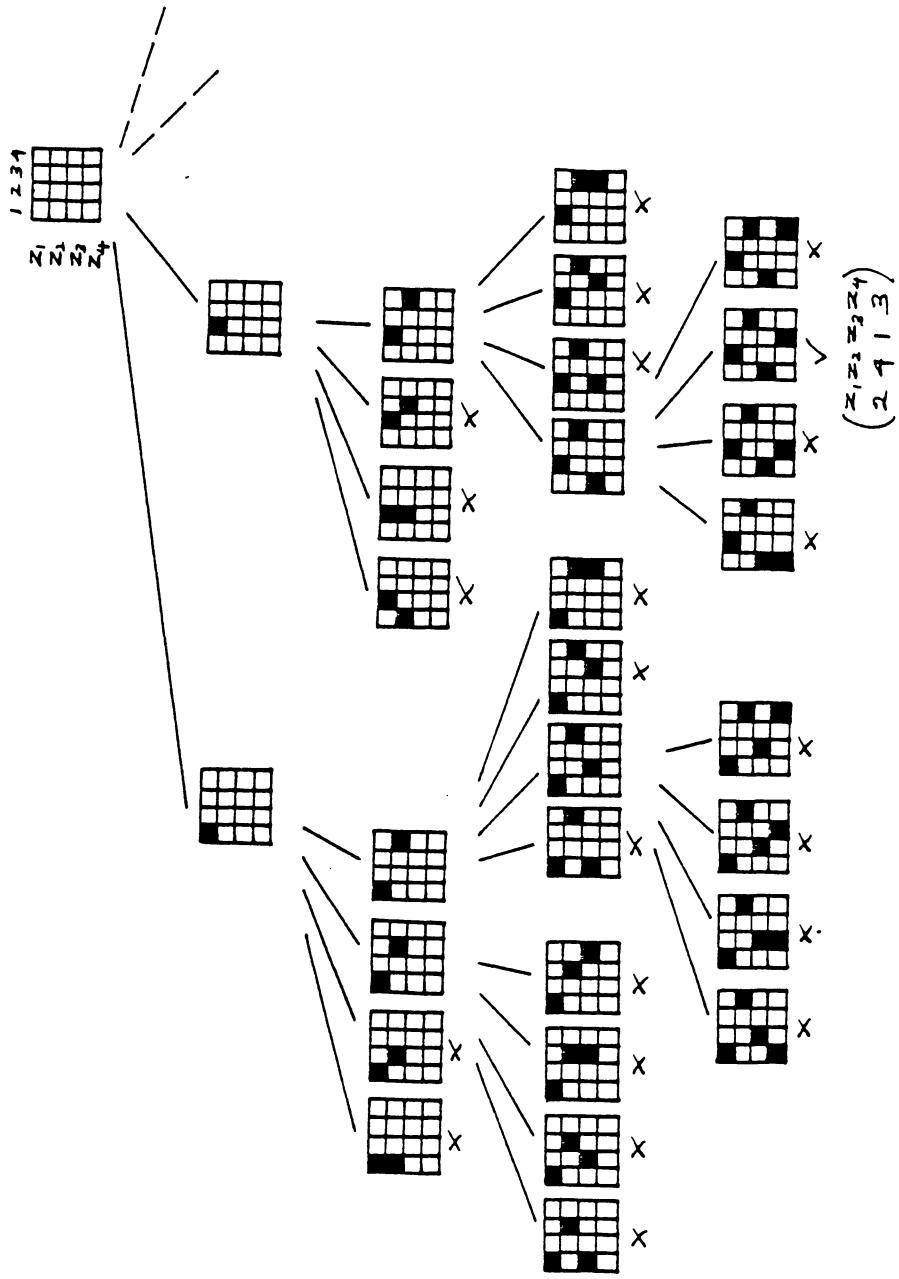
However the full set of constraints[11] Ψ_{X_k} never really need be checked. This is because for $1 \le k \le n$ at each call $F1(\overline{X}_k\ k)$ it can be shown by induction that $\overline{X}_{k-1}$ is consistent, that is, satisfies all constraints of

[9] Find all ways to place 4 queens on a 4 × 4 chess board so that no two queens attack each other. See [19] for a detailed discussion of 4-queens in relation to our formulation and notation for CLP — in particular, for the 4-queens analogues of table 1-1 and figure 1-1 above.

[10] Assuming that row i at a node is made to correspond to variable z_i (which corresponds to row i of the chess board) and that column i at a node corresponds to domain value i (which corresponds to column i of the chess board).

[11] Sets such as Ψ_{X_k} in fact do not contain constraints but only indices of constraints. However, to simplify the presentation we will often loosely refer to constraint indices as being the corresponding constraints themselves.

Fig. 2-1: Half of the symmetrical search tree generated by Backtracking in solving the 4-queens problem. (Compare with figure 2-5.)



$\Psi_{X_{k-1}}$. Therefore in testing the consistency of $\overline{X}_k$ it suffices to test only whether $\overline{X}_k$ satisfies all constraints of

$$\psi_k = \Psi_{X_k} - \Psi_{X_{k-1}} = \{\ j \mid 1 \le j \le c \text{ and } \{x_k\} \subseteq Z_j \subseteq X_k\ \} \tag{30}$$

These are the constraints whose arguments sets Z_j are subsumed by X_k but that contain the most recently instantiated variable x_k. In other words, one need check at level k only those constraints for which the current variable x_k is an argument whose instantiation at that level leaves no more arguments uninstantiated. The constraints with indices in ψ_k we call the constraints **checkable at level k**. (Not all constraints of ψ_k may however need checking at a given level k node since if one constraint is found to be violated by $\overline{X}_k$ then $\overline{X}_k$ is known to be inconsistent without checking further constraints.) Of course, which constraints constitute ψ_k will depend on the instantiation order chosen — and this is one of the two ways that instantiation order may effect problem-solving complexity.[12] Table 2-3 shows the sets ψ_k at each level, induced for each of the 3! = 6 instantiation orders possible for the 3 variables of our running example clp_0.

Note that as in this example, for any CLP instance each of the c constraints is checked at exactly one level from 0 to n. The ψ_k thus constitute a partition of the set of integers 1 to c, so that

$$\dot{\bigcup_{k=0}^{n}} \psi_k = \{\ 1\ 2\ .\ .\ c\ \} \qquad \text{and} \qquad \sum_{k=0}^{n} |\ \psi_k\ | = c \tag{31}$$

Since we are assuming that all constraints involve at least two argument variables, there are in fact never any constraints checkable before level 2, and we have that $\psi_0 = \psi_1 = \emptyset$. At levels 2 and higher, ψ_k will in general contain more than one constraint. It therefore becomes relevant to consider the order in which constraints of ψ_k are checked since — 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 ψ_k of (30) to have some arbitrary order imposed on it

$$\psi_k = (\ \psi_k^1\ \ \psi_k^2\ .\ .\ \psi_k^{|\psi_k|}\ ) \tag{32}$$

the i-th component ψ_k^i being the index of the constraint that is the i-th to be checked at a level k node. Such an ordering we call a **constraint-check ordering at level k**. There are of course $|\ \psi_k\ |!$ such orderings possible for a given set ψ_k, and this is to be chosen by the user at each level k of the search tree. For example, for any n-queens problem, there are $(k-1)!$ possible constraint-check orderings at level k, since at that level there are $k-1$ constraints to check — one between the current (or k-th) queen and each

Table 2-3: Sets ψ_k of constraints that are checkable at level k for each of the 6 possible instantiation orders X for instance clp_0.

X	ψ_0	ψ_1	ψ_2	ψ_3
$(z_1\ z_2\ z_3)$	$\emptyset$	$\emptyset$	{1}	{2 3 4}
$(z_1\ z_3\ z_2)$	$\emptyset$	$\emptyset$	{2 3}	{1 4}
$(z_2\ z_1\ z_3)$	$\emptyset$	$\emptyset$	{1}	{2 3 4}
$(z_2\ z_3\ z_1)$	$\emptyset$	$\emptyset$	$\emptyset$	{1 2 3 4}
$(z_3\ z_1\ z_2)$	$\emptyset$	$\emptyset$	{2 3}	{1 4}
$(z_3\ z_2\ z_1)$	$\emptyset$	$\emptyset$	$\emptyset$	{1 2 3 4}

[12] When the variables of an instance have non-equal domain sizes d_{x_i} then the instantiation order may also effect the complexity of problem-solving in a second way, namely by effecting the maximum number of nodes that may be generated at a level. See equation (36) below.

of the k–1 other queens that have already been placed on the board by level k. All such *past* queens must have their position tested for consistency with that just chosen for the current queen (see figure 2-1), and the order of pairing the current queen with past queens for this position-compatibility test is arbitrary.

The **natural constraint-check ordering** ψ_k at a level is that where the corresponding constraints C_j are checked in order of increasing index j. (Of course, the ψ_k themselves, as sets, depend first of all on the instantiation order X, as seen in the example of table 2-3.) The **vector of constraint-check orderings** for gBT at the various levels we denote by

$$\psi = (\psi_0 \ \psi_1 \ \psi_2 \ . \ . \ \psi_n) = (\emptyset \ \emptyset \ \psi_2 \ . \ . \ \psi_n) \tag{33}$$

Although we have spoken above of allowing an arbitrary instantiation order X and arbitrary constraint-check orders ψ_k at the various levels, it should be noted that we do still require 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 ψ_k be the same at all nodes of level k.

The above considerations lead us to the **generalized Backtracking** algorithm or **gBT** presented in figure 2-2 using a Pascal-like language. It is *generalized* with respect to the versions usually appearing in the literature, in that

1) the instantiation order X may be any permutation of Z (though this same order is to be used along all paths through the search tree),

2) the constraint-check order at each level may be any permutation of the set ψ_k of constraints checkable at level k (though this same order is to be used at all nodes of level k),

3) it is able to solve any instance of the very general type defined in section 1.1.

At line 15, algorithm gBT checks the current partial labeling $\overline{X}_k$ with respect to constraint C_j by determining if the corresponding projection onto Z_j satisfies C_j. The use of the notation $\overline{Z}_j \in T_j$ is however in general just a convenient shorthand. It is not intended to imply that the constraint relation T_j is necessarily available extensively (although this is one possibility). This test will more commonly be made by a subroutine which represents the constraint C_j intensively.

2.2.2. Some Examples Using gBT

This section shows several examples of the use of algorithm gBT. Our running example clp_0 is solved using a variety of instantiation orders X and constraint-check orders ψ_k. We will see that the choice of these orders may have a strong effect on the algorithmic complexity of gBT. For more realistic CLP instances, much greater variation is possible (and chapter 8 of [18] provides theory-based heuristics for choosing these gBT search orderings). Note that the complete data for solving clp_0 using gBT (and gFC) under all possible instantiation orders and constraint-check orders appears in chapter 6 of [18].

Figures 2-3 and 2-4 represent the tree of recursive calls to gBT when solving clp_0 with the instantiation orderings $X = (z_1 \ z_3 \ z_2)$ and $X = (z_3 \ z_2 \ z_1)$ respectively. The same conventions as described for figure 2-1 are used here. The family of sets ψ_k induced by each instantiation order used here can be seen in the corresponding row of table 2-3. 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 upon the induced sets ψ_k. The number of nodes generated at each level k when solving instance clp using gBT is denoted $N(\text{gBT } k \ clp)$, and these are shown in the table associated with each figure. Also shown in these tables, at the bottom of the $N(\text{gBT } k \ clp)$ column, are the sums

$$N(\text{gBT } clp) = \sum_{k=1}^{n} N(\text{gBT } k \ clp) \tag{34}$$

giving the total number of nodes generated (excluding the root node). Note that with these two instantiation orders, the total number of nodes generated varies markedly from 10 to 21 (a factor of two difference). We will see that the variation in the number of checks performed is even greater.

As mentioned, an instantiation order completely determines the set ψ_k of constraints checkable at each level k of the search, but the constraint-check *order* at each level may be any permutation of the corresponding set — and these orders may have a considerable influence on the number of constraint-checks performed. Both figures 2-3 and 2-4 show numbers of constraint-checks performed using two different sets of constraint-check orderings — permutations of the corresponding sets ψ_k shown in table 2-3. Each such family of orderings heads a separate column in the table associated with the figure. In each such column appear the corresponding numbers $C(\text{gBT } k \ clp)$ of constraint-checks performed at each level

Fig. 2-2: Algorithm gBT and its subroutine Consistent.

```
1   PROCEDURE gBT( X̄_k  k );                    : Enter node X̄_k
2   IF Consistent( X̄_k  k ) THEN                : Dummy call at k = 0, 1
3      IF k = n THEN print( X̄_k )
4               ELSE FOR ALL x̄_{k+1} ∈ d_{z_{k+1}} DO
5                  BEGIN
6                  X̄_{k+1} ← X̄_k || x̄_{k+1};
7                  gBT( X̄_{k+1} k+1 );          : Generating node X̄_{k+1} = X̄_k || x̄_{k+1}
8                  END;
9   END;

10  FUNCTION Consistent( X̄_k  k );
11  FOR i = 1 TO | ψ_k | DO                      : | ψ_k | = 0 for k = 0, 1
12     BEGIN
13     j ← ψ_k^i;                               : Index of constraint to check
14     Z̄_j ← Z̄_j(X̄_k);                          : Projection onto arguments of C_{ψ_k^i}
15     IF Z̄_j ∉ T_j                             : Checking constraint C_{ψ_k^i}
16           THEN RETURN false;
17     END;
18  RETURN true;
19  END;
```

Initial call:

gBT(∅ 0)

Global instance parameters:

n, $\mathbf{T} = (\ T_1\ T_2\ .\ .\ T_c\ )$, $\mathbf{Z} = (\ Z_1\ Z_2\ .\ .\ Z_c\ )$, $\mathbf{d} = (\ d_{z_1}\ d_{z_2}\ .\ .\ d_{z_n}\ )$

Global algorithm parameters:

X, $(\ \psi_2\ \psi_3\ .\ .\ \psi_n\ )$

Fig. 2-3: Solving clp_0 by gBT using a best instantiation-order, $X = (z_1\ z_3\ z_2)$, with a best (and a worst) set of constraint-check orders $\{\psi_2\ \psi_3\}$ for that X. (Compare with figures 2-4, 2-7, 2-8)

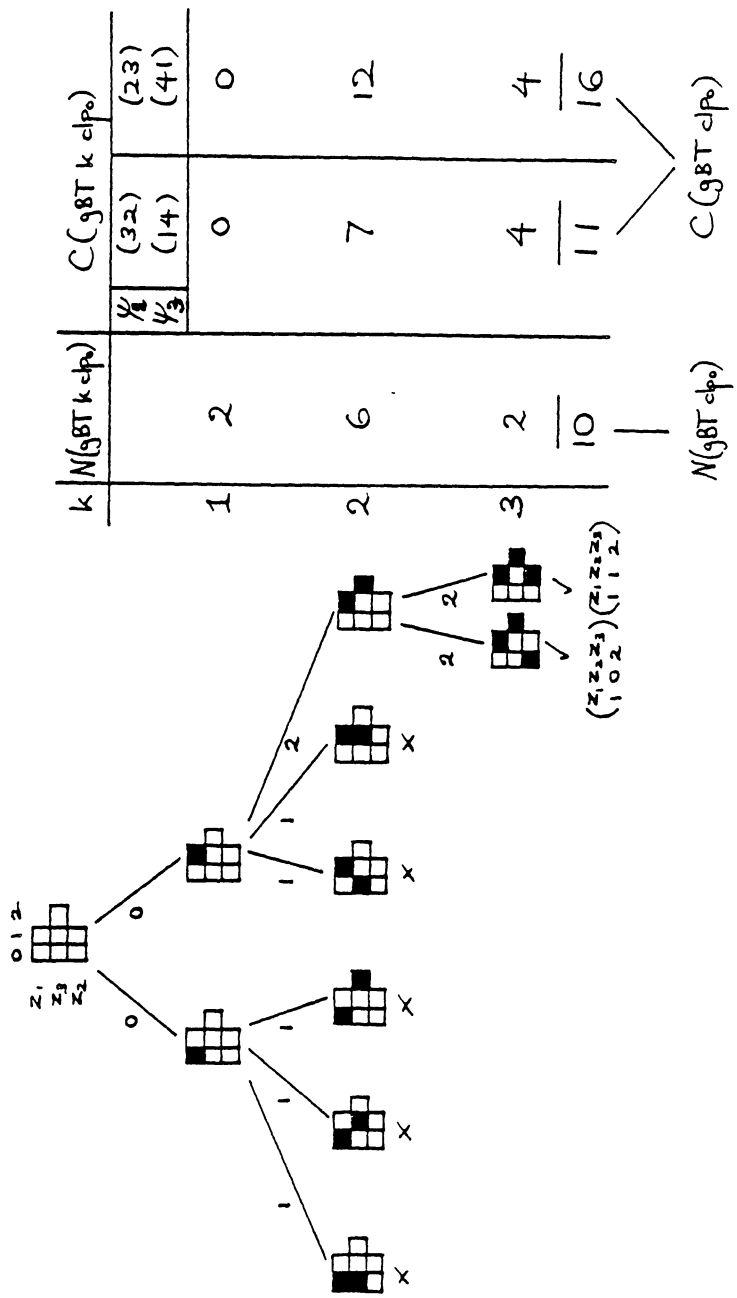
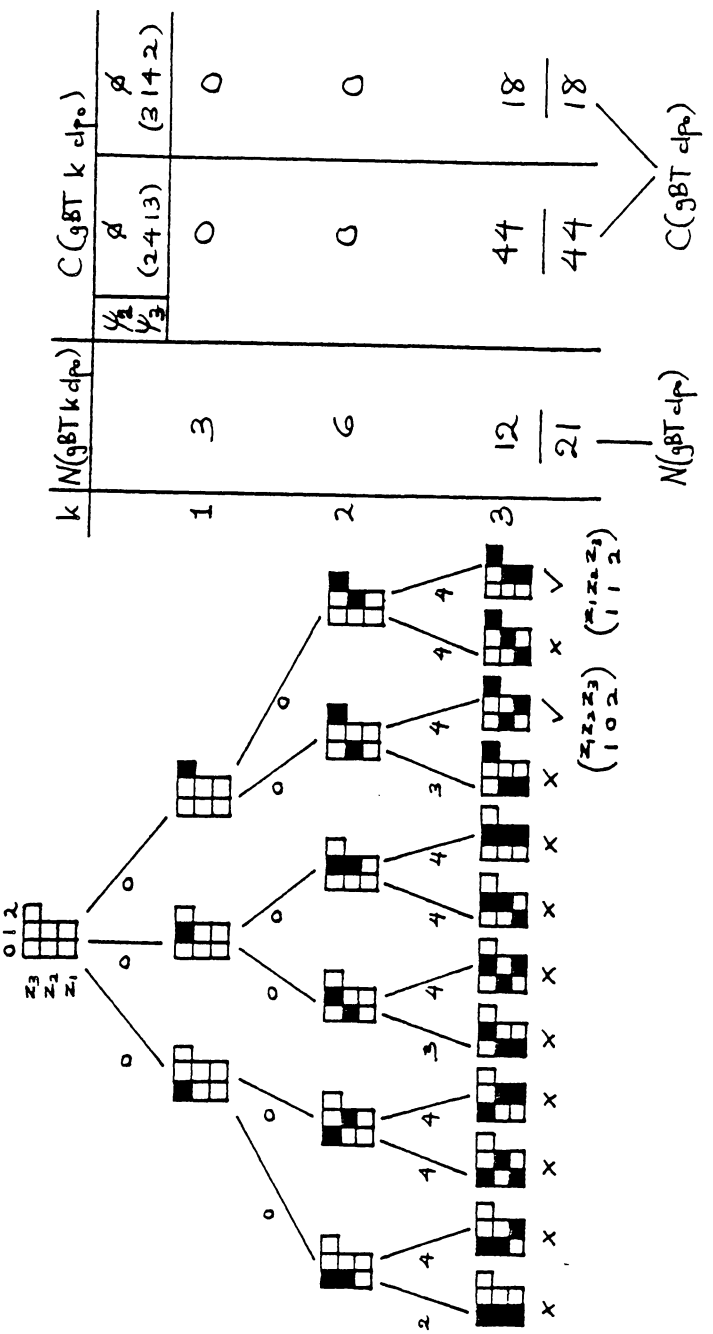


Fig. 2-4: Solving clp_0 by gBT using a worst instantiation-order, $X = (z_3\ z_2\ z_1)$, with a worst (and a best) set of constraint-check orders $\{\psi_2\ \psi_3\}$ for that X. (Compare with figures 2-3, 2-7, 2-8)



k of the search tree. These $C(\text{gBT } k \; clp)$ values are of course the sum of the number of constraint-checks performed at each node at level k, so that

$$C(\text{gBT } k \; clp) = \sum_{\bar{X}_k \, \epsilon \, D_{X_k}} C(\text{gBT } k \; \bar{X}_k \; clp) \tag{35}$$

where $C(\text{gBT } k \; \bar{X}_k \; clp)$ is the number of constraint-checks performed by gBT at node $\bar{X}_k$ in solving instance clp. This sum is actually over all **node sites** $\bar{X}_k \in D_{X_k}$ at level k, rather than only over all *generated* nodes, with the understanding that $C(\text{gBT } k \; \bar{X}_k \; clp) = 0$ at a node site where no node is generated. The number of such node-sites at level k of the search tree for gBT (and for gFC below) is just the number of different instantiations $\bar{X}_k$ that are possible, and this is of course

$$\sum_{\bar{X}_k \, \epsilon \, D_{X_k}} 1 = \mid D_{X_k} \mid = \mid \underset{z_i \, \epsilon \, X_k}{\times} d_{z_i} \mid = \prod_{z_i \, \epsilon \, X_k} \mid d_{z_i} \mid = \prod_{z_i \, \epsilon \, X_k} m_{z_i} \tag{36}$$

For the case of the first (leftmost) family of constraint-check orders ψ_k appearing in the associated table, figures 2-3 and 2-4 label the arc leading to a node by the number of checks $C(\text{gBT } k \; \bar{X}_k \; clp)$ performed at that node. The total number of constraint-checks performed when solving clp is the sum

$$C(\text{gBT } clp) = \sum_{k=2}^{n} C(\text{gBT } k \; clp) \tag{37}$$

of the numbers of checks performed at the different levels of the search tree. Note that the $k = 0$ and $k = 1$ summands are excluded here because gBT never has any constraints to check at those levels. The total numbers of checks performed are shown at the bottom of the corresponding $C(\text{gBT } k \; clp)$ columns in the tables. We see that as mentioned, changing the family of constraint-check orders may have a considerable effect on the number of constraint-checks performed. This effect is possible because the constraint-check order at a level determines at what stage any inconsistency of $\bar{X}_k$ is discovered. As soon as such an inconsistency is detected all further processing at that node is unnecessary.

The cumulative effect of such savings over many nodes in the search tree may be considerable. (Of course, for nodes where no inconsistency is detected, each constraint of ψ_k is checked once and $C(\text{gBT } k \; \bar{X}_k \; clp) = \mid \psi_k \mid$ irrespective of the order used for checking the constraints in ψ_k.) In figure 2-4 for example, changing from family $\psi_2 = \emptyset$, $\psi_3 = (2\ 4\ 1\ 3)$ to family $\psi_2 = \emptyset$, $\psi_3 = (3\ 1\ 4\ 2)$ reduces the total number of checks from 44 to 18. Using instantiation order $X = (z_1\ z_3\ z_2)$ and constraint-order family $\psi_2 = (3\ 2)$, $\psi_3 = (1\ 4)$ of figure 2-3 further reduces the total number of checks performed to 11. As mentioned, this factor of 4 reduction from 44 to 11 checks is representative of much larger reductions (both relatively and absolutely) that are often possible for more complex instances. Similar effects will be seen in section 2.3.2 for algorithm gFC, where instance clp_0 is again used as the test case.

Equation (34) expresses the total number of nodes in a gBT search tree in terms of the number at each level. It will prove useful to refine this a little further, expressing the number of nodes at a level in terms of the number at each node site at that level. This latter number, which is of course simply either 0 or 1, we denote using the following indicator function

$$\delta(\text{gBT } k \; \bar{X}_k \; clp) = \begin{cases} 1 & \text{if gBT generates a node at site } \bar{X}_k \\ & \quad \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

In terms of this we may express the number of nodes generated at level k as

$$N(\text{gBT } k \; clp) = \sum_{\bar{X}_k \, \epsilon \, D_{X_k}} \delta(\text{gBT } k \; \bar{X}_k \; clp) \tag{39}$$

and hence the total number of nodes generated by gBT given in (34) may be expressed as

$$N(\text{gBT } clp) = \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gBT } k\ \bar{X}_k\ clp) \tag{40}$$

Similarly, to indicate whether $C_{\psi_{k,i}}$, the i-th checkable constraint at level k is actually checked by gBT at node site $\bar{X}_k$ when solving instance clp, we introduce

$$\delta(\text{gBT } k\ i\ \bar{X}_k\ clp) = \begin{cases} 1 & \text{if gBT checks } C_{\psi_{k,i}} \text{ at site } \bar{X}_k \\ & \qquad \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

Remember that some constraints in ψ_k may not be checked at $\bar{X}_k$ because of an inconsistency detected at that node when checking an earlier constraint in ψ_k, or because the node at $\bar{X}_k$ is not even generated — and it is in these cases that the above indicator function takes on value 0. In terms of (41), the number of constraint checks performed at node $\bar{X}_k$ can be expressed as

$$C(\text{gBT } k\ \bar{X}_k\ clp) = \sum_{i=1}^{|\psi_k|} \delta(\text{gBT } k\ i\ \bar{X}_k\ clp) \tag{42}$$

so that using this together with (35) and (37), we have for the total number of checks performed by gBT in solving instance clp

$$C(\text{gBT } clp) = \sum_{k=2}^{n} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} \delta(\text{gBT } k\ i\ \bar{X}_k\ clp) \tag{43}$$

Quantities $N(\text{gBT } clp)$ and $C(\text{gBT } clp)$ are specific to the given instance clp and as such are effectively intractable. Their expected values *are* tractable however, as will be seen in section 3.1 where equations (40) and (43) provide the starting points for the respective analyses.

2.2.3. More on gBT Constraint Sets

Table 2-4 defines two additional constraint (index) sets for gBT that will be useful in section 3.1. Examples of these new sets and the earlier constraint sets for gBT are given in table 2-5. Lastly, table 2-6 presents without proof some simple relationships between these sets. From now on we use Ψ_k as an abbreviation for Ψ_{X_k}.

Table 2-4: Constraint-related definitions for algorithm gBT

Symbol	Defined as	Defined for
$\psi_k(<i)$	$\begin{cases} \{ \psi_k^1 \psi_k^2 \ldots \psi_k^{i-1} \} \\ \emptyset \end{cases}$	$0 \le k \le n, \begin{cases} 2 \le i \le \mid \psi_k \mid \\ i=1 \text{ or } k = 0, 1 \end{cases}$
$\Psi_k(<i)$	$\bigcup_{h=0}^{k-1} \psi_h \cup \psi_k(<i)$	$0 \le k \le n,\ 1 \le i \le \mid \psi_k \mid$

In words, these new constraint sets are as follows:

- $\psi_k(<i)$ denotes the first i–1 constraints checkable at level k.
- $\Psi_k(<i)$ is in a sense a cumulative version of $\psi_k(<i)$. It contains those constraints that are checkable at some level prior to level k or are one of the first i–1 checkable at level k.

Table 2-5 should prove useful in developing familiarity with these and the earlier constraint sets ψ_k and Ψ_k for gBT. Both table 2-5 and its analogue, table 2-10 for gFC, use the example of a full and simple A-ary CLP instance having $n = 5$ variables and constraints of maximum arity $A = 4$. As defined in section 1.2, such instances have exactly one constraint corresponding to each combination of A' variables, $2 \le A' \le A$. This is convenient here because when not more than one constraint exists having any given set of variables as arguments (as by definition is the case for all *simple* instances) we may use the argument set itself, instead of the less informative constraint indices $1 \le j \le c$, to uniquely index the constraints — and such argument-set-indexing is more transparent for the purposes of the present example. If the constraint over argument set $\{x_2\ x_3\ x_5\}$ were say C_7 then the list $(x_2\ x_3\ x_5)$ may be used in place of the index 7 to denote the constraint. In tables 2-5 and 2-10 such lists are further abbreviated to just the concatenation of their subscripts so that $(x_2\ x_3\ x_5)$ becomes simply 235. Note however that the 235 stands for $(x_2\ x_3\ x_5)$ and not $(z_2\ z_3\ z_5)$, i.e. indices are given with respect to instantiation order not name order of the variables. This allows the example to be interpreted more generally since it is valid for arbitrary instantiation order $X = (x_1\ x_2 \ldots x_5)$. If indices were interpreted as being with respect to name order then it would be a valid example only for the particular instantiation ordering $X = (z_1\ z_2 \ldots z_n)$.

For a full and simple 4-ary instance on $n = 5$ variables, a column of table 2-5 for a given k shows all the indices (using the above argument-set-indexing convention) in ψ_k, the index set of checkable constraints at level k. Their order from top to bottom denotes the intended constraint-check order at each level. Of course for each k there are $\mid \psi_k \mid !$ such orderings possible, of which we chose to show the one corresponding to the natural, lexographical ordering of the argument-set indices of the constraints.

Table 2-6 presents, without proof, some simple relationships between constraint index sets for gBT. 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 using the example of table 2-5, although the latter is for full and simple (rather than for pure and simple) instances. The related discussion in [22] may be helpful.

2.3. Generalized Forward Checking (gFC)

Paralleling the above section on gBT, this section develops the generalized Forward Checking algorithm gFC, based on an alternative recursive function for the solution set T of a CLP instance. The algorithm is a version of the Forward Checking algorithm of [11] which also appears anonymously in [16]. 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 type defined in section 1.1 — 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 [11]. The algorithm found best was a version of Forward Checking which we will study in generalized form below. Several examples are given here of the use of gFC, again emphasizing the effect of instantiation-order and constraint-check order upon problem-solving complexity.

Table 2-5: Example constraint-check orders ψ_k and some induced constraint index-sets for gBT solving a full and simple 4-ary instance with $n = 5$ variables using the generic instantiation order $X = (x_1\ x_2\ x_3\ x_4\ x_5)$. See text for details.
(Compare with tables 2-10, 2-11 and 2-12.)

k	2	3	4	5
x_k	x_2	x_3	x_4	x_5
$\lvert \psi_k \rvert$	1	3	7	14
i	ψ_2	ψ_3	$\psi_4(<6)$	ψ_5
1	12	13	14	15
2		23	24	25
3		123	34	35
4			124	45
5			134	125
6			234	135
7			1234	145
8				235
9				245
10				345
11				1235
12				1245
13				1345
14				2345

Ψ_2, Ψ_3, $\Psi_4(<6)$, Ψ_4, Ψ_5

$\psi_4(<6) = \{\ 14\ 24\ 34\ 124\ 134\ \}$

$\Psi_4(<6) = \psi_2 \cup \psi_3 \cup \psi_4(<6) = \{\ 12\ \} \cup \{\ 13\ 23\ 123\ \} \cup \{\ 14\ 24\ 34\ 124\ 134\ \}$

$= \{\ 12\ 13\ 123\ 14\ 24\ 34\ 124\ 134\ \}$

Table 2-6: Relations between Constraint Index Sets for gBT

Arbitrary Instances:

$$\Psi_k = \Psi_{k-1} \dot{\cup} \psi_k \qquad (\Psi_1 = \Psi_0 = \emptyset) \tag{44}$$

$$\Psi_k = \dot{\bigcup_{h=0}^{k}} \psi_h \tag{45}$$

$$\Psi_n = \dot{\bigcup_{k=0}^{n}} \psi_k = \{ 1\ 2\ .\ .\ c \} \tag{46}$$

$$\Psi_k(<i) = \Psi_{k-1} \dot{\cup} \psi_k(<i) \tag{47}$$

$$\Psi_k(<1) = \Psi_{k-1} \tag{48}$$

$$| \psi_k(<i) | = i-1 \tag{49}$$

$$| \Psi_k | = \sum_{h=0}^{k} | \psi_h | \tag{50}$$

$$| \Psi_n | = \sum_{h=0}^{n} | \psi_h | = c \tag{51}$$

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

$$| \psi_k | = \binom{k-1}{A-1} \tag{52}$$

$$| \Psi_k | = \sum_{h=0}^{k} | \psi_h | = \sum_{h=0}^{k} \binom{h-1}{A-1} = \binom{k}{A} - \binom{0}{A-1} = \binom{k}{A} \quad \text{since } A \geq 2 \tag{53}$$

$$| \Psi_n | = c = \binom{n}{A} \tag{54}$$

2.3.1. Development 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 [7], [11], [15]. For example, in figure 2-1 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 [7] and [11]. Perhaps the simplest and yet most effective of these is the Forward Checking algorithm of [16] and [11]. 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

$$d_f^{\bar{X}_k} = \{ \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\bar{X}_k \,||\, \bar{f}) \in T_j \ \ \forall\, j \in \Psi_{X_k \cup \{f\}} \} \subseteq d_f \tag{55}$$

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 $\bar{X}_k \mid\mid \bar{f}$. Thus (55) 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

$$\mathbf{d}^{\bar{X}_k} = (d_{x_{k+1}}^{\bar{X}_k} \quad d_{x_{k+2}}^{\bar{X}_k} \quad . \; . \quad d_{x_n}^{\bar{X}_k}) \tag{56}$$

Anticipating the 4-queens example in figure 2-5 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

$$\mathbf{d}^{\bar{X}_k} = \mathbf{d}^{(2\ 4)} = (d_{x_3}^{(2\ 4)} \quad d_{x_4}^{(2\ 4)}) = (\ \{1\} \ \ \{1\ 3\}\ ) \tag{57}$$

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

$$F2(\bar{X}_k \ k) = \begin{cases} \emptyset \text{ if } \bar{X}_k \text{ is inconsistent} \\ \{\ \bar{X}_k\ \} \text{ if } \bar{X}_k \text{ is consistent and } k = n \\ \emptyset \text{ if } \bar{X}_k \text{ is consistent, } k < n \text{ and } \emptyset \in \mathbf{d}^{\bar{X}_k} \\ \bigcup_{\bar{x}_{k+1} \in d_{x_{k+1}}^{X_k}} F2(\bar{X}_k \mid\mid \bar{x}_{k+1} \ k+1) \text{ otherwise} \end{cases} \tag{58}$$

This is very much like the recursive function $F1$ of (26) on which gBT is based, except for the following two differences.

(i) The union of consistent labelings that are extensions of $\bar{X}_{k+1} = \bar{X}_k \mid\mid \bar{x}_{k+1}$ 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 (27) the solution set T for an instance can be expressed as

$$T = F2(\emptyset \ \ 0) \tag{59}$$

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 X_k is inconsistent. Thus for computing T, checking for consistency of $\bar{X}_k$ may be avoided on the right side of (58) giving the simpler version

$$F2(\bar{X}_k \ k) = \begin{cases} \{\ \bar{X}_k\ \} \text{ if } k = n \\ \emptyset \text{ if } k < n \text{ and } \emptyset \in \mathbf{d}^{\bar{X}_k} \\ \bigcup_{\bar{x}_{k+1} \in d_{x_{k+1}}^{X_k}} F2(\bar{X}_k \mid\mid \bar{x}_{k+1} \ k+1) \text{ otherwise} \end{cases} \tag{60}$$

The right side of (60) makes use of $\mathbf{d}^{\bar{X}_k}$. Anticipating its usefulness at the next level of recursion we include it as an explicit argument of $F2$ — passing along $\mathbf{d}^{\bar{X}_k}$ on the right side of (60) and receiving its previous level counterpart $\mathbf{d}^{\bar{X}_{k-1}}$ on the left. We thus obtain the modified version of $F2$

$$F2(\bar{X}_k \;\; \mathbf{d}^{\bar{X}_{k-1}} \;\; k) = \begin{cases} \{\, \bar{X}_k \,\} \text{ if } k = n \\ \\ \emptyset \text{ if } k < n \text{ and } \emptyset \in \mathbf{d}^{\bar{X}_k} \\ \\ \bigcup_{\bar{z}_{k+1} \in d_{z_{k+1}}^{\bar{X}_k}} F2(\bar{X}_k \,||\, \bar{z}_{k+1} \;\; \mathbf{d}^{\bar{X}_k} \;\; k+1) \text{ otherwise} \end{cases} \tag{61}$$

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

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

which is the list $\mathbf{d}$ of the original, unfiltered domains d_{z_i} for the variables, prefixed with a dummy element $\emptyset$. This prefixing of $\emptyset$ will allow a uniform implementation of function $F2$ as discussed below.

Paralleling figure 2-1, figure 2-5 shows half of the symmetrical search tree when using $F2$ to solve 4-queens. As in figure 2-1, we are again using for the instantiation order the standard name order given in (28). A node is drawn in the search tree for each recursive call to $F2$. The node corresponding to call $F2(\bar{X}_k \;\; \mathbf{d}^{\bar{X}_{k-1}} \;\; k)$ is called the node at $\bar{X}_k$, or simply node $\bar{X}_k$. Each node $\bar{X}_k$ of the tree shows the corresponding value of $\bar{X}_k$ using the same convention as described for figure 2-1. 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 $\bar{X}_k$ is not left empty but is used to show the set $d_f^{\bar{X}_k}$ of still viable values for f given the current instantiations $\bar{X}_k$.[13] 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 $\bar{X}_k$. 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, $d_f^{\bar{X}_k} = \emptyset$, a wavy line is shown through the corresponding row. This is called a **domain wipe-out.**

Note that in figure 2-5 (but not in later such search trees) we indicate in the above manner the (fully) filtered domain $d_f^{\bar{X}_k}$ for *each* future variable $f \in F_k$ at a node $\bar{X}_k$. 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 $d_f^{\bar{X}_k}$ from the inputs $d_f^{\bar{X}_{k-1}}$ was fully completed — with filtering of the inputs $d_f^{\bar{X}_{k-1}}$ possibly not having even been started for some future variables. In such a case all further filtering at that node may be avoided, and $d_f^{\bar{X}_k}$ 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.)

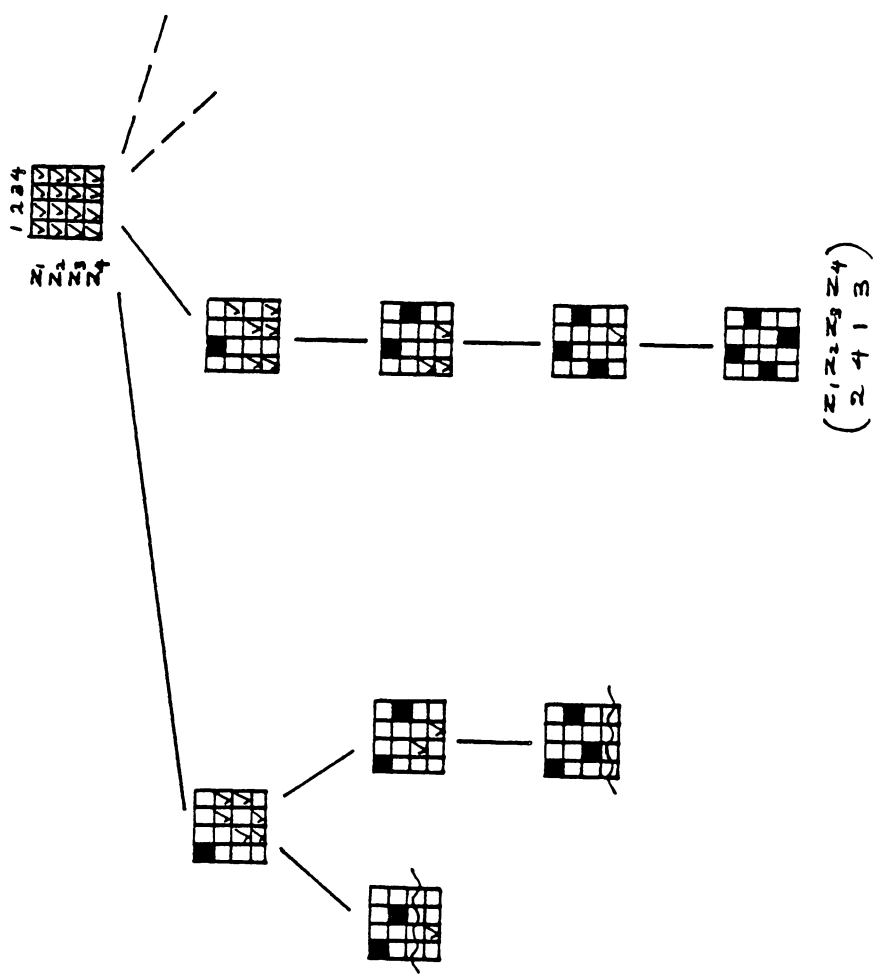
There is a significant difference in the number of nodes generated by gBT and gFC in figures 2-1 and 2-5. However, this does not provide a meaningful comparison between the two algorithms, since while gFC generates less nodes than gBT does, gFC in general does far more constraint checks at a node.[14] 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 (61) 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 z_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

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

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

Fig. 2-5: Half of the symmetrical search tree generated by Forward Checking in solving the 4-queens problem. (Compare with figure 2-1.)



definition, we have from (55) that

$$d_f^{\bar{X}_k} = \{\, \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\, \bar{X}_k \,||\, \bar{f}\,) \in T, \quad \forall\, j \in \Psi_{X_k \cup \{f\}} \,\} \tag{62}$$

$$d_f^{\bar{X}_{k-1}} = \{\, \bar{f} \mid \bar{f} \in d_f \text{ and } \bar{Z}_j(\, \bar{X}_{k-1} \,||\, \bar{f}\,) \in T, \quad \forall\, j \in \Psi_{X_{k-1} \cup \{f\}} \,\} \tag{63}$$

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. Furthermore, for these values much of the condition required in (62) has already been satisfied by the condition in (63). A little thought shows that for values in $d_f^{\bar{X}_{k-1}}$, the condition of (62) 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 (62) and (63) 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, (62) can be expressed in terms of (63) as

$$d_f^{\bar{X}_k} = \{\, \bar{f} \mid \bar{f} \in d_f^{\bar{X}_{k-1}} \text{ and } \bar{Z}_j(\, \bar{X}_k \,||\, \bar{f}\,) \in T, \quad \forall\, j \in \phi_{kf} \,\} \tag{64}$$

$$\text{where} \quad \phi_{kf} = \Psi_{X_k \cup \{f\}} - \Psi_{X_{k-1} \cup \{f\}} - \Psi_{X_k} \tag{65}$$

$$= \{\, j \mid j \in J_1^c \text{ and } \{x_k\ f\} \subseteq Z_j \subseteq X_k \cup \{f\} \,\} \tag{66}$$

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 2-7 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 2-3, 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

$$\dot{\bigcup_{k=0}^{n}} \; \dot{\bigcup_{f \in F_k}} \phi_{kf} = \{\, 1\ 2\ .\ .\ c \,\} \qquad \text{and} \qquad \sum_{k=0}^{n} \sum_{f \in F_k} |\, \phi_{kf} \,| = c \tag{67}$$

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 $\bar{X}_k$ generated at level $k = n$ is known to be consistent without need for further checking, as shown in (60) and (61). 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

Table 2-7: Sets ϕ_{kx_i} and ϕ_k at each level k for each of the 6 possible instantiation orders X for instance clp_0.

X	ϕ_{0x_1}	ϕ_{0x_2}	ϕ_{0x_3}	ϕ_{1x_2}	ϕ_{1x_3}	ϕ_{2x_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}

concatenation

$$\phi_k = \phi_{kf_{k1}} \,||\, \phi_{kf_{k2}} \,||\, \cdots \,||\, \phi_{kf_{k,n-k}} \tag{68}$$

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 (68) 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 2-10 below. Only f-exhaustive orderings were treated in our earlier paper [22]. The expected gFC complexities of section 3.2.4.1 assume f-exhaustive orderings. But the other subsections of section 3.2 give results for the fully general ordering scheme which we now introduce.

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

$$\phi_k = \bigcup_{f \in F_k} \phi_{kf} = \{\, j \mid j \in J_i^c \text{ and } \exists\, f \in F_k \text{ s.t. } \{x_k\ f\} \subseteq Z_j \subseteq X_k \cup \{f\} \,\} \tag{69}$$

This union we call the set of constraints **forward-checkable at level k** (against *some* future variable). Besides the sets ϕ_{kf}, table 2-7 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 (67) and (31) we have

$$\dot{\bigcup_{k=0}^{n}} \phi_k = \{1\ 2\ \ldots\ c\} \quad \text{and} \quad \sum_{k=0}^{n} |\, \phi_k \,| = c \tag{70}$$

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 (69) to have some arbitrary order imposed on it

$$\phi_k = (\, \phi_k^1\ \phi_k^2\ \ldots\ \phi_k^{|\phi_k|} \,) \tag{71}$$

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 (32). 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 2-5), and the order of pairing the current queen with future queens for this position-compatibility test is arbitrary.

The **natural constraint-check ordering** ϕ_k at each level is that where constraints C_j are checked in increasing order of 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 2-7.) The **vector of constraint-check orderings** for gFC at the various levels we denote by

$$\phi = (\phi_0 \ \phi_1 \ \phi_2 \ . \ . \ \phi_{n-1} \ \phi_n) = (\emptyset \ \phi_1 \ \phi_2 \ . \ . \ \phi_{n-1} \ \emptyset) \tag{72}$$

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 constraint-check ordering 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 [21].

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^j 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 (66) we see that the future variable corresponding to constraint ϕ_k^j, which we denote as f_k^j or as $f_{\phi_k^j}$, can be recovered as

$$f_k^j = f_{\phi_k^j} = Z_{\phi_k^j} - X_k \tag{73}$$

the set difference between the argument set of the constraint $C_{\phi_k^j}$ and the set of variables X_k.[15] The constraint-check order of (71) then induces the sequence

$$G_k = (\ f_k^1 \ f_k^2 \ . \ . \ f_k^{|\phi_k|}\ ) \tag{74}$$

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

$$G_k = (\ x_{k+3} \ x_{k+1} \ x_{k+1} \ x_{k+1} \ x_{k+3} \ x_{k+5}\ ) \tag{75}$$

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 2-8 shows some of the possible constraint-check orders ϕ_k (permutations of the sets ϕ_k in table 2-7) together with their induced domain-filtering orders G_k at each level for our running example clp_0.

The above considerations lead us finally to the **generalised Forward-Checking** algorithm or **gFC** presented in figure 2-6 using a Pascal-like language. As for gBT in section 2.2, gFC 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 type in section 1.1. Note that function Filter uses a copy $\mathbf{d}$ of $\mathbf{d}^{X_{k-1}}$. This is simply to avoid using $\mathbf{d}^{X_{k-1}}$ as the name of a variable whose value (once filtering is underway) does not agree with the definitions in (55) and (56). In practice this copying need not be done and the filtering can be performed directly with the components of the input vector $\mathbf{d}^{X_{k-1}}$.

At line 20, algorithm gFC checks the consistency of value $\overline{f}$ with respect to constraint $C_{\phi_k^j}$, given the prior instantiations $\overline{X}_k$. As for gBT it should be remembered that the test $\overline{Z}_j \notin T_j$ 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($\mathbf{d}\ d^f$) at line 24 replaces the previous filtered domain for variable f in $\mathbf{d}$ 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

[15] The right-hand side of (73) is really a singleton set containing variable f_k^j rather than variable f_k^j itself.

Table 2-8: 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
$(z_1\ z_3\ z_2)$	$\emptyset$ / $\emptyset$	(1 2 3) / $(z_2\ z_3\ z_3)$	(4) / (z_2)
	$\emptyset$ / $\emptyset$	(3 1 2) / $(z_3\ z_2\ z_3)$	(4) / (z_2)
	$\emptyset$ / $\emptyset$	(2 1 3) / $(z_3\ z_2\ z_3)$	(4) / (z_2)
$(z_2\ z_3\ z_1)$	$\emptyset$ / $\emptyset$	(1) / (z_1)	(2 4 3) / $(z_1\ z_1\ z_1)$
	$\emptyset$ / $\emptyset$	(1) / (z_1)	(3 4 2) / $(z_1\ z_1\ z_1)$
$(z_3\ z_2\ z_1)$	$\emptyset$ / $\emptyset$	(3 2) / $(z_1\ z_1)$	(4 1) / $(z_1\ z_1)$

(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 implementation in the case $k = 0$, it is convenient in the initial call gFC($\bar{X}_0\ \mathbf{d}^{\bar{X}_{-1}}\ 0$) to use

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

where we prefix a dummy entry $\emptyset$ 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 $(d_{x_1}\ d_{x_2}\ .\ .\ d_{x_n})$ 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 2.4, where a bit vector representation is used for sets d^f.

2.3.2. Some Examples Using gFC

Paralleling section 2.2.2 for gBT, this section shows several examples of using the gFC algorithm of figure 2-6. Again our running example clp_0 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 [21] provides theory-based heuristics for choosing these gFC search orderings and empirically studies the studies the savings that may result from their use. An analogous study appears for gBT in chapter 8 of [18]. Note that the complete data for solving clp_0 using gFC (and gBT) under all possible instantiation orders and constraint-check orders appears in chapter 6 of [18].

Figures 2-7 and 2-8 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 for figure 2-5 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 2-7. 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(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(gFC k clp) column, are the sums

$$N(\text{gFC } clp) = \sum_{k=1}^{n} N(\text{gFC } k\ clp) \tag{76}$$

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 2-3 and 2-4 where the same instance was being solved. It is easy to

Fig. 2-6: Algorithm gFC and its subroutine Filter.

```
1   PROCEDURE gFC( X̄_k  d^{X̄_{k-1}}  k );                : Enter node X̄_k
2   IF k = n THEN print( X̄_k ) ELSE
3         BEGIN
4         d^{X̄_k} ← Filter(X̄_k  d^{X̄_{k-1}}  k)          : Dummy call at k = 0
5         IF d^{X̄_k} ≠ (∅) THEN
6               FOR ALL x̄_{k+1} ∈ d^{X̄_k}_{x_{k+1}} DO    : d^{X̄_k}_{x_{k+1}} from d^{X̄_k}
7                  gFC( X̄_k || x̄_{k+1}  d^{X̄_k}  k+1);   : Generating node X̄_{k+1} = X̄_k || x̄_{k+1}
8         END;
9   END;

10  FUNCTION Filter( X̄_k  d^{X̄_{k-1}}  k );              : Filters domains in tail of d^{X̄_{k-1}}
11  d ← d^{X̄_{k-1}};
12  FOR i = 1 TO | φ_k | DO                               : | φ_k | = 0 for k = 0 (and n)
13     BEGIN
14     j ← φ_k^i;                                         : Index of constraint to check
15     f ← Z_j - X_k ;                                    : Determining f_k^i ∈ F_k whose domain will be filtered
16     d^f ← current domain for f in d;
17     FOR ALL f̄ ∈ d^f DO                                 : Filter domain d^f
18        BEGIN
19        Z̄_j ← Z̄_j( X̄_k || f̄ );                         : Projection onto arguments of C_{φ_k^i}
20        IF Z̄_j ∉ T_j                                    : Checking constraint C_{φ_k^i}
21              THEN d^f ← d^f - ( f̄ );                   : Filtering f̄ from d^f
22        END;
23     IF d^f = ∅ THEN RETURN (∅)                         : A domain wipe-out has occurred
24                ELSE d ← Update( d  d^f );              : Update domain of f in d
25     END;
26  RETURN tail (d);                                      : Filter (X̄_k  d^{X̄_{k-1}}  k) = d^{X̄_k}
27  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:

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

show that gFC never generates more nodes than gBT on a given instance with a given instantiation ordering.[16] 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 2-7 and 2-8 show numbers of constraint-checks performed using two different families of constraint-check orderings — permutations (appearing in table 2-8) of the corresponding sets ϕ_k shown in table 2-7. Each such family of orderings heads a separate 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

$$C(\text{gFC } k \; clp) = \sum_{\bar{X}_k \in D_{X_k}} C(\text{gFC } k \; \bar{X}_k \; clp) \tag{77}$$

where $C(\text{gFC } k \; \bar{X}_k \; clp)$ is the number of constraint-checks performed by gFC at node $\bar{X}_k$ in solving instance clp. As for equation (35), this sum is over all **node sites** $\bar{X}_k \in D_{X_k}$ at level k, rather than only over nodes actually generated, with the understanding that $C(\text{gFC } k \; \bar{X}_k \; clp) = 0$ 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 (36). For the case of the first (leftmost) family of constraint-check orders ϕ_k appearing in the associated table, figures 2-7 and 2-8 label the arc leading to a node by the number of checks $C(\text{gFC } k \; \bar{X}_k \; clp)$ 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

$$C(\text{gFC } k \; \bar{X}_k \; clp) = \sum_{i=1}^{|\phi_k|} C(\text{gFC } k \; i \; \bar{X}_k \; clp) \tag{78}$$

where $C(\text{gFC } k \; i \; \bar{X}_k \; clp)$ is the number of checks of the i-th forward-checkable constraint $C_{\phi_k^i}$ performed by gFC at node-site $\bar{X}_k$ when solving instance clp. As before, $C(\text{gFC } k \; i \; \bar{X}_k \; clp) = 0$ at a node-site $\bar{X}_k$ 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 $C(\text{gFC } k \; i \; \bar{X}_k \; clp)$ that make up the number of checks at a node, can be conveniently indicated on gFC search tree diagrams such as figures 2-7 and 2-8, 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 2-7 and 2-8 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 2-8), 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 2-7, at the right son $\bar{X}_1 = (1)$ of the root node, using constraint-check order $\phi_1 = (3\ 1\ 2)$, 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 2-8) with constraint C_1 being used between them to filter variable $f_1^2 = z_2$. The corresponding numbers of checks are $C(\text{gFC } k\ 1\ \bar{X}_k \; clp) = 3$ and $C(\text{gFC } k\ 3\ \bar{X}_k \; clp) = 1$ and these values appear in the sum $3 + 1$ beside the z_3 row of node $\bar{X}_1 = (1)$.

These last-mentioned constraint-specific numbers of checks at a node add, as in (78), 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 (77), 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

$$C(\text{gFC } clp) = \sum_{k=1}^{n-1} C(\text{gFC } k \; clp) \tag{79}$$

Note that the $k = 0$ and $k = n$ summands are excluded here since gFC never has any constraints to

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

Fig. 2-7: 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 2-3, 2-4, 2-8)

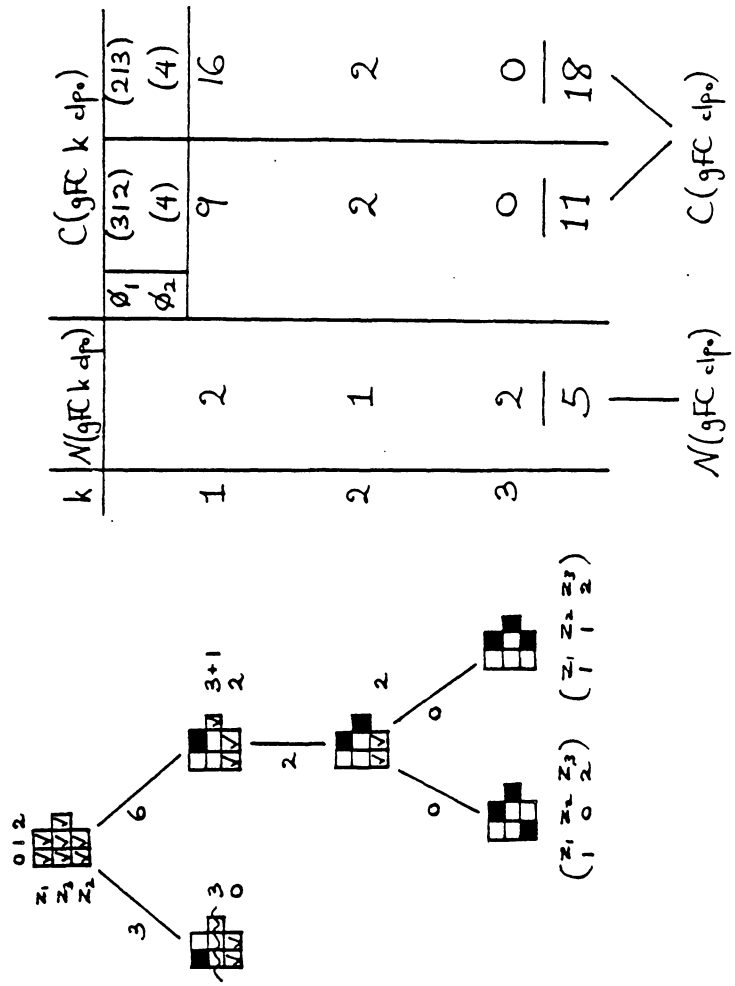
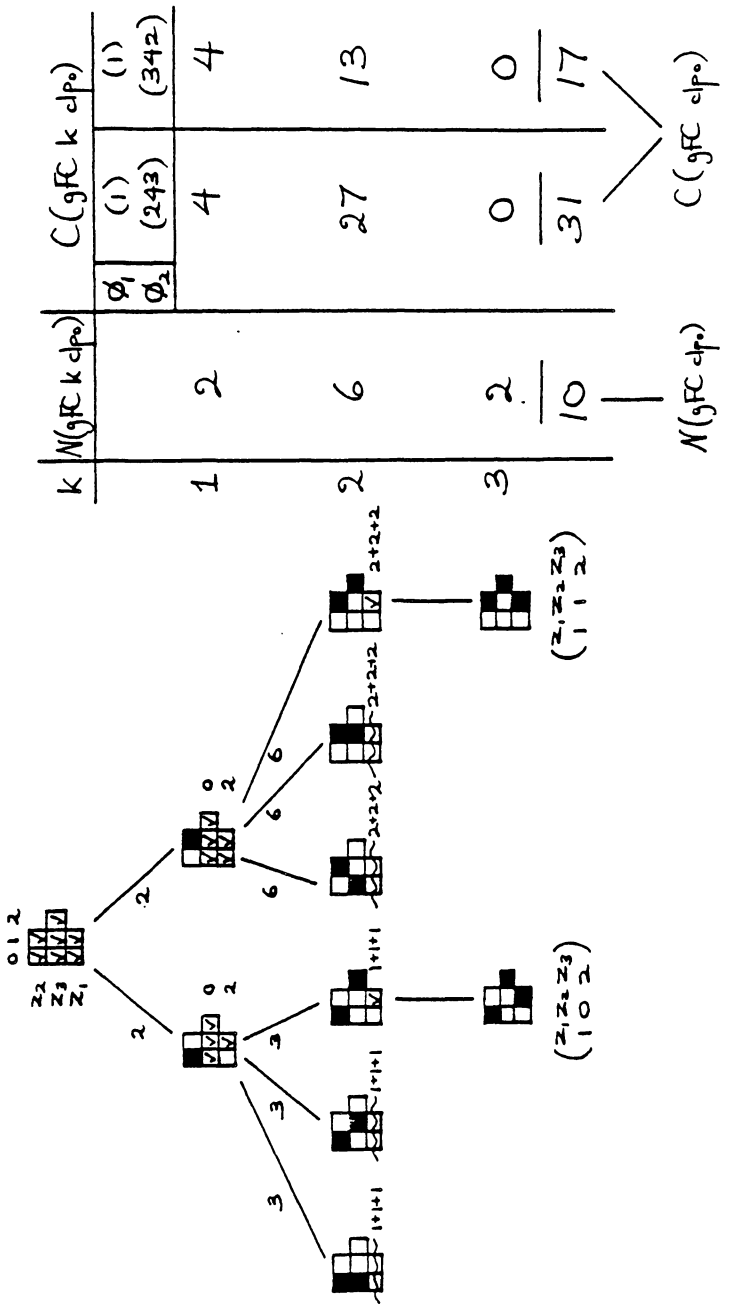


Fig. 2-8: Solving clp_0 by gFC using a worst instantiation-order, $X = (z_2\ z_3\ z_1)$, with a worst (and a best) set of constraint-check orders $\{\phi_1\ \phi_2\}$ for that X. (Compare with figures 2-3, 2-4, 2-7)



forward check at those levels. The total numbers of checks performed are shown at the bottom of the corresponding C(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 occurring at a node are discovered. As soon as such a wipe-out is detected, the termination condition $\emptyset \in \mathbf{d}^{\bar{X}_k}$ is satisfied and further processing at that node is unnecessary.[17] Note that for nodes $\bar{X}_k$ 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 $| d_{f_k^i}^{\bar{X}_{k-1}} |$ times. In this case, irrespective of the order used for checking the constraints in ϕ_k, we have that

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

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[18] is because the constraint-check order at a level determines at what stage a given value $\bar{f}$ is eliminated (if it is eliminated) from a given input domain $d_f^{\bar{X}_{k-1}}$. 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 $\bar{X}_k$ is determined by the order ϕ_k and over a whole search tree the cumulative 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 2-8 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 2-7 further reduces the total number of checks performed to 11. This figure, 11 (attained also for gBT in figure 2-3), 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 in section 2.2.2 for gBT, we also introduce here two indicator functions in terms of which N(gFC k *clp*) and C(gFC k i $\bar{X}_k$ *clp*) can be be further expanded. The first is

$$\delta(\text{gFC } k\ \bar{X}_k\ clp) = \begin{cases} 1 & \text{if gFC generates a node at site } \bar{X}_k \\ & \quad \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{80}$$

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

$$N(\text{gFC } k\ clp) = \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gFC } k\ \bar{X}_k\ clp) \tag{81}$$

The total number of nodes generated by gFC given in (76) may then be expressed as

[17] Figures 2-7 and 2-8 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.

[18] 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}.

$$N(\text{gFC } clp) = \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gFC } k\ \bar{X}_k\ clp) \tag{82}$$

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 $0 \le t \le m_{f_k^i}$.

$$\delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) = \begin{cases} 1 & \text{if gFC checks } C_{\phi_k^i} \text{ exactly } t \text{ times at site } \bar{X}_k \\ & \text{when solving instance } clp \\ 0 & \text{otherwise} \end{cases} \tag{83}$$

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

$$C(\text{gFC } k\ i\ \bar{X}_k\ clp) = \sum_{t=0}^{m_{f_k^i}} t\ \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) \tag{84}$$

Note the inclusion of t as a factor of the summed terms. Combining this with the above equalities (77), (78) and (79), gives

$$C(\text{gFC } clp) = \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} \sum_{t=0}^{m_{f_k^i}} t\ \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) \tag{85}$$

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 section 3.2 where equations (82) and (85) provide the starting points for the respective analyses. One last indicator function that will be useful in section 3.2 can be defined in terms of that in (83). It is

$$\delta(\text{gFC } k\ i \ge 1\ \bar{X}_k\ clp) = 1 - \delta(\text{gFC } k\ i\ 0\ \bar{X}_k\ clp) = \sum_{t=1}^{m_{f_k^i}} \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) \tag{86}$$

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

2.3.3. More on gFC Constraint Sets

Table 2-9 defines several additional constraint (index) sets for gFC that will be useful in section 3.2. Examples of these new sets and the earlier constraint sets for gFC are given in tables 2-10, 2-11 and 2-12. Lastly, table 2-13 presents without proof some simple relationships between these sets. This section parallels section 2.2.3 for gBT, and tables 2-9 to 2-13 here correspond tables 2-4 to 2-6 for gBT.

Table 2-9: Constraint-related definitions for algorithm gFC

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

In words, the new constraint sets in table 2-9 are as follows:

- Φ_k and Φ_{kf} are cumulative versions of ϕ_k and ϕ_{kf} respectively in the same way that Ψ_k is a cumulative version of ψ_k — see (45). 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 cumulative 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.
- $\phi_{kf}(<i)$ 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.
- $\Phi_{kf}(<i)$ is a cumulative version of $\phi_{kf}(<i)$, 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 2-10 to 2-12 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 2-5, and they use the same convention for indexing constraints as used in table 2-5. Table 2-10 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 (29) for gBT). Table 2-11 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.

Table 2-10: 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 2-5.)

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	

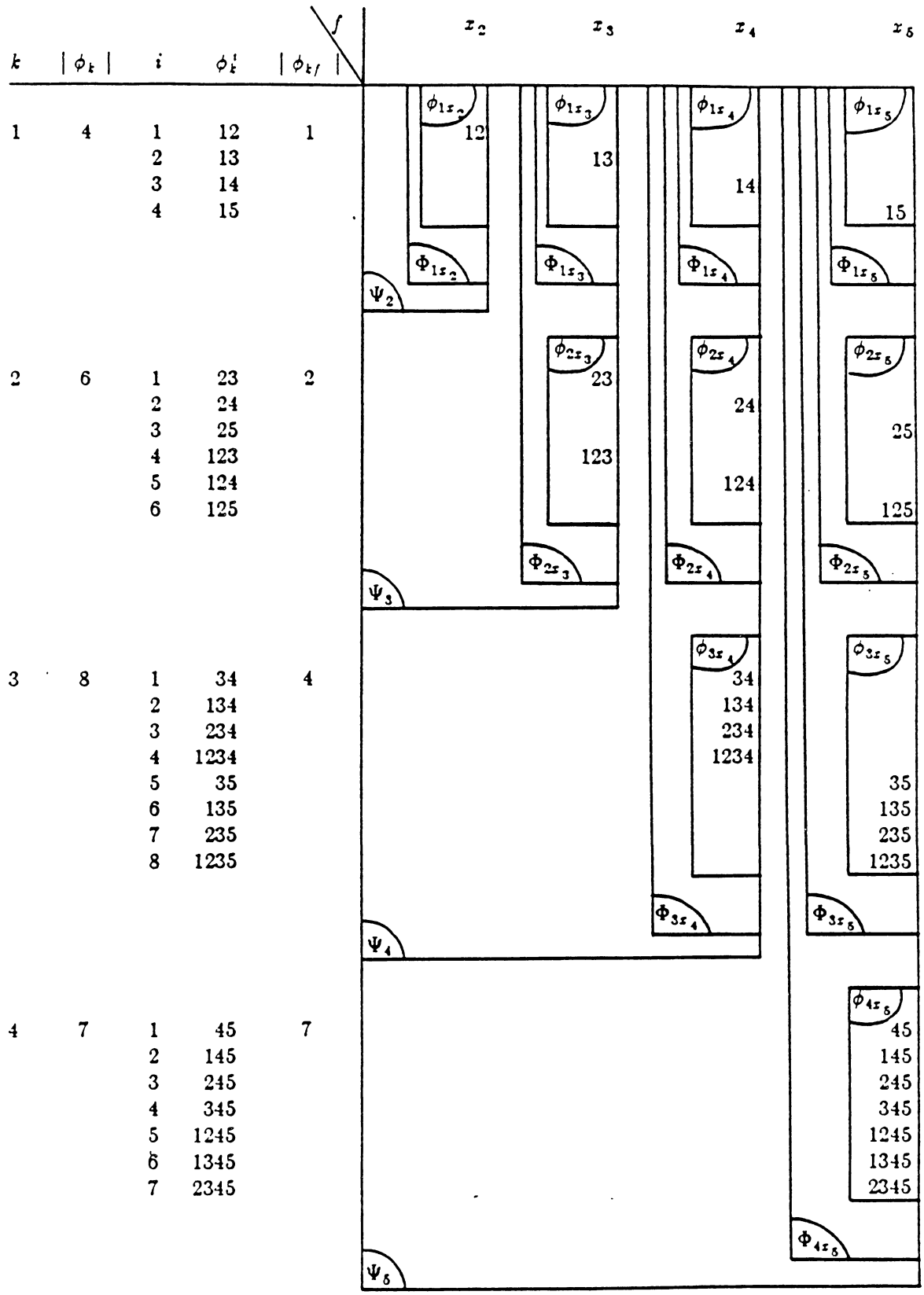


Table 2-11: A repeat of the table 2-10 situation showing constraint-index sets ϕ_{kf} , ϕ_k and Φ_k. (Compare with table 2-5.)

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	

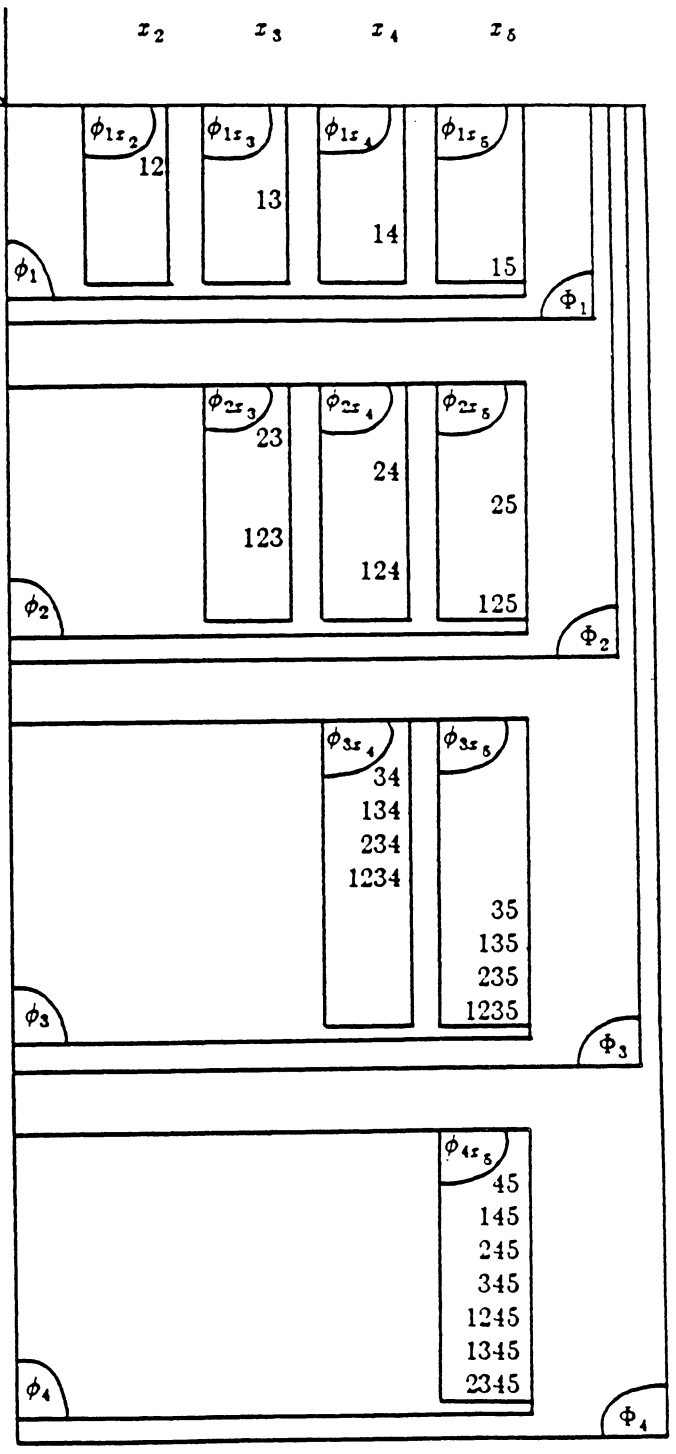
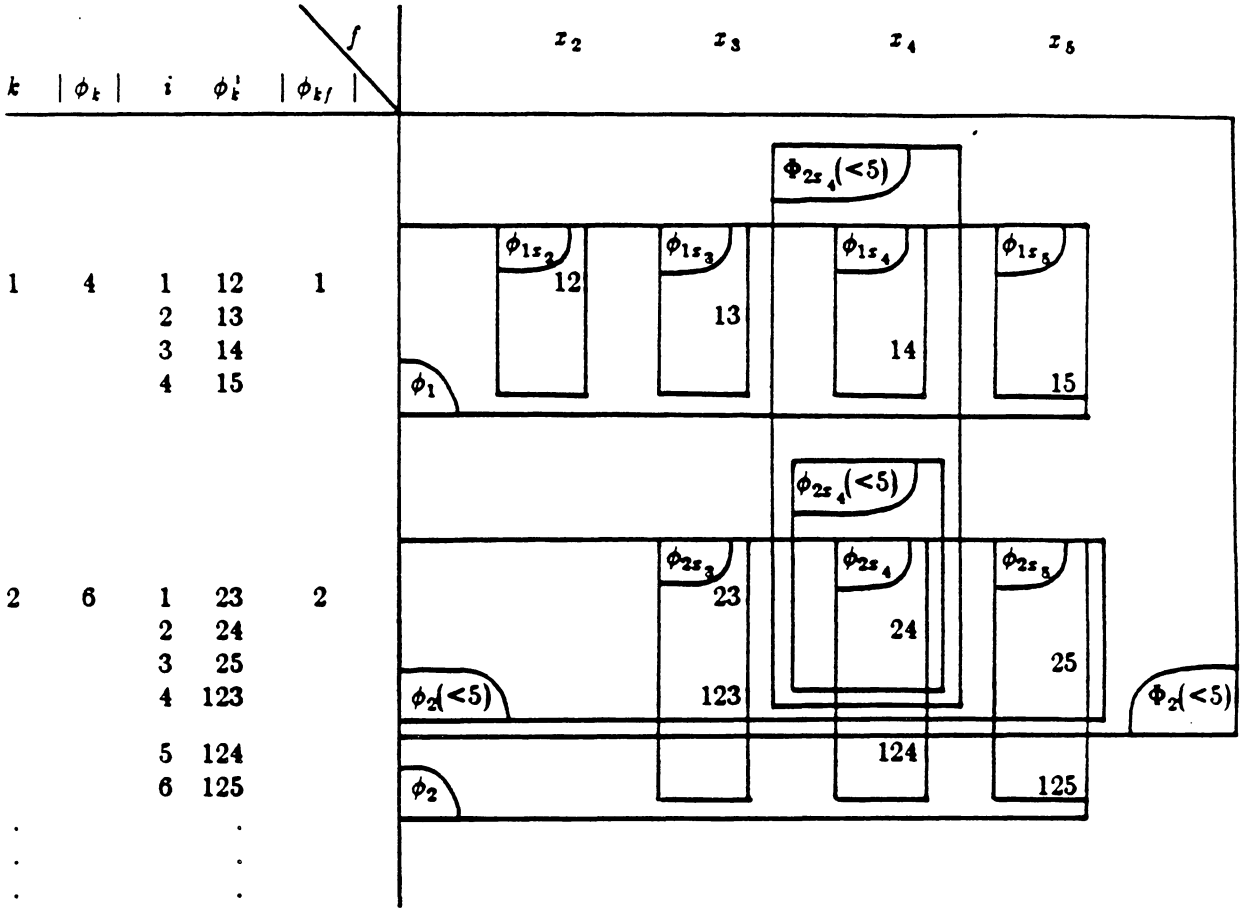


Table 2-12: A repeat of the table 2-10 situation showing constraint-check-order-dependent sets $\phi_k(<i)$, $\phi_{kf}(<i)$, $\Phi_k(<i)$ and $\Phi_{kf}(<i)$ defined in table 2-9.
(Compare with table 2-5.)



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

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

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

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

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

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

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

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

Table 2-13: Relations between Constraint Index Sets for gFC

Arbitrary Instances:

$$\{ 1\,2\,..\,c \} = \bigcup_{k=0}^{n} \bigcup_{f \in F_k} \phi_{kf} = \bigcup_{k=0}^{n} \phi_k = \Phi_n = \Phi_{n-1} = \bigcup_{k=0}^{n} \psi_k = \Psi_n \tag{87}$$

$$\Phi_{kf} = \Phi_{k-1,f} \cup \phi_{kf} \qquad (\Phi_{0f} = \phi_{0f} = \phi_{nf} = \emptyset) \tag{88}$$

$$\Phi_k = \Phi_{k-1} \cup \phi_k \qquad (\Phi_0 = \phi_0 = \phi_n = \emptyset) \tag{89}$$

$$\Phi_k (<i) = \Phi_{k-1} \cup \phi_k (<i) \qquad (\Phi_0(<i) = \phi_0(<i) = \phi_n (<i) = \emptyset) \tag{90}$$

$$\Phi_{kf} (<i) = \Phi_{k-1,f} \cup \phi_{kf} (<i) \qquad (\Phi_{0f} (<i) = \phi_{0f} (<i) = \phi_{nf} (<i) = \emptyset) \tag{91}$$

$$\Phi_{kf} (<1) = \Phi_{k-1,f} \tag{92}$$

$$\Phi_{kx_{k+1}} = \psi_{k+1} \tag{93}$$

$$| \phi_k (<i) | = i-1 \tag{94}$$

$$| \Phi_{kf} | = \sum_{h=0}^{k} | \phi_{hf} | = | \Phi_{k-1,f} | + | \phi_{kf} | \tag{95}$$

$$| \Phi_{kf} (<i) | = \sum_{h=0}^{k-1} | \phi_{hf} | + | \phi_{kf} (<i) | = | \Phi_{k-1,f} | + | \phi_{kf} (<i) | \tag{96}$$

$$\Psi_k = \bigcup_{h=0}^{k-1} \Phi_{hx_{h+1}} = \bigcup_{h=0}^{k-1} \bigcup_{t=1}^{h} \phi_{tx_{h+1}} \tag{97}$$

$$| \Psi_k | = \sum_{h=0}^{k-1} | \Phi_{hx_{h+1}} | \tag{98}$$

$$| \Psi_n | = \sum_{h=0}^{n} | \Phi_{hx_{h+1}} | = c \tag{99}$$

Table 2-13: Continued

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

$$| \phi_{kf} | = \binom{k-1}{A-2} \tag{100}$$

$$| \phi_k | = \binom{k-1}{A-2}(n-k) \tag{101}$$

$$| \Phi_{kf} | = \sum_{h=0}^{k} | \phi_{hf} | = \sum_{h=0}^{k} \binom{h-1}{A-2} = \binom{k}{A-1} \tag{102}$$

$$| \Phi_{kf}(<i) | = | \Phi_{k-1,f} | + | \phi_{kf}(<i) | = \binom{k-1}{A-1} + | \phi_{kf}(<i) | \tag{103}$$

$$| \Psi_k | = \sum_{h=0}^{k-1} | \Phi_{hz_{h+1}} | = \sum_{h=0}^{k-1} \binom{h}{A-1} = \binom{k}{A} - \binom{0}{A-1} = \binom{k}{A} \quad \text{since } A \geq 2 \tag{104}$$

$$| \Psi_n | = c = \binom{n}{A} \tag{105}$$

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 2-9 of the sets $\phi_k(<i)$, $\phi_{kf}(<i)$, $\Phi_k(<i)$ and $\Phi_{kf}(<i)$. Examples of these sets for the table 2-10 situation are given in table 2-12.

Note that in table 2-10 (and 2-11), ϕ_3 is an **f-exhaustive** constraint-check ordering, defined in section 2.3.1, where all constraints relevant to a given future variable are employed before proceeding to use those for a different future variable. Such orderings receive special consideration in section 3.2.4.1. 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 2-13 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 2-10 to 2-12, although the latter are for full and simple (rather than for pure and simple) instances. The related discussion in [22] may be helpful.

2.4. 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 [16] and in [11] 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 (73), 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.[19] At the stage when C_j is 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 $\bar{f}_j$ 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

$$\{\ \bar{f}_j\ \mid\ \bar{f}_j \in d_{f_j}\ \text{and}\ \overline{Z_j - f_j}\ ||\ \bar{f}_j \in T_j\ \}$$

With these machine words available, the loop from lines 17 to 22 of function Filter in figure 2-6 can be replaced by a single machine *and* as follows

[19] 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 representing problem constraints.

$$d^f \leftarrow \text{and}(\ d^f\ \ T_j(\overline{Z_j - f_j})\ ) \tag{106}$$

which is the generalized form of Haralick's expression on page 271 of [11]. 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 2-6.

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

$$w_{z_i} = \lceil m_{z_i}/b \rceil \tag{107}$$

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 (106), but by the following loop over machine words

$$\text{FOR } w = 1 \text{ TO } w_f \text{ DO} \tag{108}$$

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

where w is being used to index the w_f separate 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.[20] In forward checking a constraint against a variable f whose current filtered domain is $d^f \subseteq d_f$, gFC performs $|\ d^f\ |$ constraint-checks (at lines 17 to 22 of figure 2-6). Algorithm gwFC on the other hand will perform w_f machine *ands* in the loop of (108), independent of the size of the current filtered domain 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 section 3.3 allows this possibility to be studied theoretically, since the analysis is for domains and machine words of arbitrary size. Simplified versions of these algorithms are compared in our earlier paper [21].

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 the heuristics of the type developed in [21], 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 2.6 of [19].

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

3. EXPECTED COMPLEXITIES of CLP ALGORITHMS

In this section we derive the expected complexity of solving a CLP instance via each of the algorithms gBT, gFC and wgFC of the previous section. For each algorithm, expectations are obtained over both the generic big-class and over the generic small-class (see section 1.2) under the associated probability models 0, 1 and 2 (see section 1.3). In each case, results are given for two measures of complexity: number of nodes generated and number of constraint-checks performed (with number of machine *ands* relacing the latter measure for algorithm gwFC). Table 3-1 summarizes the various expected quantities derived in this section. The notation used in the table and below, is a specialization of that in (15) to (17). Note that this section uses the algorithm-independent results of the previous section to obtain algorithm-dependent results.

Note that our small-class expectations provide more precise information about the complexity of problem-solving within CLP than do the big-class expectations, by virtue of the fact that the partition of CLP into small-classes is a refinement of that into big-classes. Moreover, the homogeneity of small-classes makes small-class expectations particularly important in that they can be used to approximate the corresponding exact-case values for individual subsumed instances — so that the precision becomes effectively the maximum possible. Big-class expectations do not have this virtue, due to the inhomogeneity of big-classes. However (as explained in section 3.5.1 of [19]) big-class expectations, besides being interesting as expectations per se., are also useful for our purposes in that they can act as good approximations for small-class quantities, and hence indirectly may be used to approximate exact-case values.

Note that the results here are for the fully arbitrary big-class $CLP_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ and small-class $CLP_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$. They therefore model instances having arbitrary and not necessarily equal domain sizes, constraints of arbitrary and not necessarily equal arities, constraints over an arbitrary family of argument sets, multiple constraints on the same argument set, etc. Moreover the results also model the arbitrary instantiation order and arbitrary constraint-check order allowed in our three algorithms. The fully general results are presented under a sequence of successive specializations, allowing ultimately a comparison with the results for the highly specialized case studied by Haralick in [11]. Our specialized results will be seen to agree with those of Haralick, except as noted in section 3.2.4.2 where we find a subtle error in Haralick's expression for the expected number of constraint-checks performed by gFC.

The results here extend and generalize those in our earlier papers [20] and [21] which treated only the pure and simple binary case, and in [22] which treated the uniform degree-m, pure and simple A-ary case. Moreover the derivation methods differ. The results in [21] are based on a recursive (Markov chain) analysis whereas here we use a more natural, direct (non-recursive) approach.

Table 3-1: Expected complexities derived in this section

Probability space / Algorithm	Big-class, model-0 Ω_β^0		Big-class, model-1 Ω_β^1		Small-class, model-2 Ω_σ^2	
gBT	$\overline{N}_\beta^0(\text{gBT})$	$\overline{C}_\beta^0(\text{gBT})$	$\overline{N}_\beta^1(\text{gBT})$	$\overline{C}_\beta^1(\text{gBT})$	$\overline{N}_\sigma^2(\text{gBT})$	$\overline{C}_\sigma^2(\text{gBT})$
gFC	$\overline{N}_\beta^0(\text{gFC})$	$\overline{C}_\beta^0(\text{gFC})$	$\overline{N}_\beta^1(\text{gFC})$	$\overline{C}_\beta^1(\text{gFC})$	$\overline{N}_\sigma^2(\text{gFC})$	$\overline{C}_\sigma^2(\text{gFC})$
gwFC	$\overline{N}_\beta^0(\text{gwFC})$	$\overline{A}_\beta^0(\text{gwFC})$	$\overline{N}_\beta^1(\text{gwFC})$	$\overline{A}_\beta^1(\text{gwFC})$	$\overline{N}_\sigma^2(\text{gwFC})$	$\overline{A}_\sigma^2(\text{gwFC})$

3.1. Expected-Complexity of gBT

This section derives the expected numbers of nodes generated and constraint-checks performed in solving a CLP instance by the gBT algorithm of section 2.2. The expectations are obtained both over big-classes and over small-classes under the corresponding probability models 0, 1 and 2 of section 1.3. Much of the analysis can be presented in a model-independent manner[21], with specialization to individual models required only towards the end. Therefore the sample space Ω below is to be taken as referring to either the generic big-class $\Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z}) = \mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ or the generic small-class $\Omega_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S}) = \mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$.

3.1.1. Class- and Model-Independent Expressions

In section 2.2.2, the number of nodes generated and the number of constraint checks performed by gBT in solving instance *clp* were expressed respectively as

$$N(\mathrm{gBT}\ clp) = \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\mathrm{gBT}\ k\ \bar{X}_k\ clp) \tag{109}$$

$$C(\mathrm{gBT}\ clp) = \sum_{k=2}^{n} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} \delta(\mathrm{gBT}\ k\ i\ \bar{X}_k\ clp) \tag{110}$$

These are in terms of the indicator functions $\delta(\mathrm{gBT}\ k\ \bar{X}_k\ clp)$ and $\delta(\mathrm{gBT}\ k\ i\ \bar{X}_k\ clp)$ defined in (38) and (41). The former function has value 1 if gBT generates a node at the level-k node site $\bar{X}_k$ when solving *clp*, and otherwise has value 0. The latter indicator function has value 1 if gBT checks constraint $C_{\psi_{k,i}}$ at site $\bar{X}_k$ in solving *clp*, and otherwise has value 0. The two expressions (109) and (110) obtain the total number of nodes and checks by simply adding up over the whole search tree, the 1 values indicating the occurrence of a node or a constraint-check.

In these summations, instance *clp* is kept fixed and we vary the values of k, $\bar{X}_k$ and i which index respectively levels, nodes and constraints. One may however fix the latter values and let the instance *clp* vary. This is done in defining the sample-space subsets given in table 3-2. These are respectively the set of instances in Ω for which gBT generates a node at site $\bar{X}_k$ and the set of instances for which gBT checks constraint $C_{\psi_{k,i}}$ at site $\bar{X}_k$. These sets are defined directly in terms of the behavior of algorithm gBT itself. They can however be expressed in a more useful form as follows, in terms of properties of the instances themselves. For this we make use of the constraint-based sets defined in table 1-3.

Table 3-2: Some gBT-dependent subsets of $\Omega = \Omega_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ or $\Omega_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$

Symbol	Defined as	Defined for
$E(\mathrm{gBT}\ k\ \bar{X}_k)$	$\{\ clp\ \mid\ \delta(\mathrm{gBT}\ k\ \bar{X}_k\ clp) = 1\ \}$	$1 \le k \le n$
$E(\mathrm{gBT}\ k\ i\ \bar{X}_k)$	$\{\ clp\ \mid\ \delta(\mathrm{gBT}\ k\ i\ \bar{X}_k\ clp) = 1\ \}$	$2 \le k \le n,\ 1 \le i \le \mid \psi_k \mid$

[21] As was seen in a simpler context when finding the expected number of solutions of an instance, in section 3.5.2 of [19].

$$E(\text{gBT } k\ \bar{X}_k) = E(\Psi_{k-1}\ \bar{X}_k) \qquad 1 \le k \le n \tag{111}$$

$$E(\text{gBT } k\ i\ \bar{X}_k) = E(\Psi_k(<i)\ \bar{X}_k) \qquad 2 \le k \le n \tag{112}$$

$$E(\text{gBT } k\ \bar{X}_k) = E(\text{gBT } k\ 1\ \bar{X}_k) \qquad 2 \le k \le n \tag{113}$$

(111) holds since gBT generates node $\bar{X}$ when solving an instance iff for that instance all constraints of Ψ_{k-1} are satisfied by the instantiations of $\bar{X}_k$. This is so since Ψ_{k-1} is the set of all constraints checked by gBT at levels prior to level k, the level at which node site $\bar{X}_k$ occurs. If all these constraints are satisfied by $\bar{X}_k$, there can not have been a constraint violation at any ancestor node of $\bar{X}_k$ and node $\bar{X}_k$ is therefore generated.

(112) is an extension of the previous result. It holds since gBT checks constraint $C_{\psi_k^i}$ at site $\bar{X}_k$ when solving an instance iff for that instance all constraints of $\Psi_k(<i)$ are satisfied by $\bar{X}_k$. This is true, since $\Psi_k(<i) = \Psi_{k-1} \cup \psi_k(<i)$ is the set of all constraints checked at levels 1 to k plus those up to (but not including) the i-th checkable at level k. If all these are satisfied by $\bar{X}_k$, there can not have been a constraint violation at any ancestor node of $\bar{X}_k$, nor at node $\bar{X}_k$ before constraint $C_{\psi_k^i}$ is checked, and the latter constraint is therefore checked at node $\bar{X}_k$.

(113) holds since clearly a node is generated by gBT at site $\bar{X}_k$ iff the *first* constraint checkable at that level *is* actually checked at that site. However, this result cannot be used at level $k = 1$ where there are no constraints to check (remember all constraints are assumed to have arity 2 or more). Note that (113) can also be derived from (111), (112) and (48), since from these we have

$$E(\text{gBT } k\ 1\ \bar{X}_k) = E(\Psi_{k-1}\ \bar{X}_k) = E(\text{gBT } k\ \bar{X}_k)$$

With these preliminary results we can now derive the expected value of $N(\text{gBT } clp)$ and of $C(\text{gBT } clp)$ as follows. Note that, as mentioned in section 1.3, we use $P(a_1 \ldots a_m)$ to denote the probability of the parameterized event $E(a_1 \ldots a_m)$.

$$\bar{N}(\text{gBT}) = \sum_{clp \in \Omega} N(\text{gBT } clp)\, P(clp) \qquad \text{by (11)} \tag{114}$$

$$= \sum_{clp \in \Omega} \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gBT } k\ \bar{X}_k\ clp)\, P(clp) \qquad \text{by (109)} \tag{115}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \sum_{clp \in \Omega} \delta(\text{gBT } k\ \bar{X}_k\ clp)\, P(clp) \tag{116}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} P(\text{gBT } k\ \bar{X}_k) \qquad \text{by (13)} \tag{117}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} P(\Psi_{k-1}\ \bar{X}_k) \qquad \text{by (111)} \tag{118}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \prod_{j \in \Psi_{k-1}} P(j\ \bar{X}_k) \qquad \text{by } T_j \text{ independence} \tag{119}$$

$$\overline{C}(\text{gBT}) = \sum_{clp \in \Omega} C(\text{gBT } clp)\, P(clp) \quad (11) \tag{120}$$

$$= \sum_{clp \in \Omega} \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} \delta(\text{gBT } k\ i\ \overline{X}_k\ clp)\, P(clp) \quad \text{by (110)} \tag{121}$$

$$= \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} \sum_{clp \in \Omega} \delta(\text{gBT } k\ i\ \overline{X}_k\ clp)\, P(clp) \tag{122}$$

$$= \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} P(\text{gBT } k\ i\ \overline{X}_k) \quad \text{by (13)} \tag{123}$$

$$= \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} P(\Psi_k(<i)\ \overline{X}_k) \quad \text{by (112)} \tag{124}$$

$$= \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \sum_{i=1}^{|\psi_k|} \prod_{j \in \Psi_k(<i)} P(j\ \overline{X}_k) \quad \text{by } T_j \text{ independence} \tag{125}$$

$$= \sum_{k=2}^{n} \sum_{\overline{X}_k \in D_{X_k}} \Big[\prod_{j \in \Psi_{k-1}} P(j\ \overline{X}_k) \Big] \Big[\sum_{i=1}^{|\psi_k|} \prod_{j \in \psi_k(<i)} P(j\ \overline{X}_k) \Big] \quad \text{by (47)} \tag{126}$$

Note that in this last expression we could just as well have used

$$k' = \text{Min}\{\ k \mid 2 \le k \le n \text{ such that } |\psi_k| \neq 0\ \} \tag{127}$$

as the lower value for k, instead of $k = 2$. This simply corresponds to the fact that no constraint-checks are performed at any level k of the tree for which there are no constraints to check. Formally, this is valid because for k at which $|\psi_k| = 0$ in (126) the sum over i is degenerate with value 0. The corresponding summand in the sum over k is therefore also 0. By definition of k' we have that $|\psi_k| = 0$ for $2 \le k < k'$ so these values of k need not be considered. There may of course be other levels $k > k'+1$ for which no constraints exist to be checked, and their corresponding summands will also contribute zero to the overall sum.

Expressions (119) and (126) for $\overline{N}$(gBT) and $\overline{C}$(gBT) both require for their completion only an expanded form of the probability $P(j\ \overline{X}_k)$. This term *is* dependent on the probability model used, and the following subsections complete the derivations for each model individually as appropriate. Note that the sums over $\overline{X}_k$ in (119) and (126) in general introduce an undesirably large number of terms — see (36). Fortunately under each model used, we find that these sums can be collapsed because $P(j\ \overline{X}_k)$ is independent of $\overline{X}_k$.

3.1.2. Small-Classes Under Model-2

The gBT expected complexities (119) and (126) contain $P(j\ \overline{X}_k)$, the probability that constraint C_j is satisfied by the instantiations of $\overline{X}_k$. From table 1-4 we see that for a small-class CLP$_\sigma$(n **m** c **Z S**) under model-2 this probability becomes $P_\sigma^2(j\ \overline{X}_k) = R_j = S_j/M_j$, and this is independent of which value-tuple $\overline{X}_k$ we consider from the cartesian product D_{X_k}. This independence allows the sums over $\overline{X}_k$ in (119) and (126) to be collapsed by use of (36) to obtain respectively

$$\overline{N}_{\sigma}^{2}(\text{gBT}) = \sum_{k=1}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[\prod_{j \in \Psi_{k-1}} R_j \Big] \tag{128}$$

$$\overline{C}_{\sigma}^{2}(\text{gBT}) = \sum_{k=2}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[\prod_{j \in \Psi_{k-1}} R_j \Big] \Big[\sum_{i=1}^{|\psi_k|} \prod_{j \in \psi_k(<i)} R_j \Big] \tag{129}$$

3.1.3. Big-Classes Under Model-1

From table 1-4 we see that for a big-class $\text{CLP}_{\beta}(n\ \mathbf{m}\ c\ \mathbf{Z})$ under model-1 with satisfiability-rates vector $\mathbf{p} = (p_1\ p_2\ \ldots\ p_c)$, probability $P(j \mid \underline{X}_k)$ becomes $P_{\beta}^{1}(j \mid \underline{X}_k\ ;\ \mathbf{p}) = p_j$. As in the above model-2 situation, this probability is independent of which value-tuple $\underline{X}_k$ we consider from the cartesian product D_{X_k}. Thus again the sums over $\underline{X}_k$ in (119) and (126) may be collapsed by use of (36), giving

$$\overline{N}_{\beta}^{1}(\text{gBT}\ ;\ \mathbf{p}) = \sum_{k=1}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[\prod_{j \in \Psi_{k-1}} p_j \Big] \tag{130}$$

$$\overline{C}_{\beta}^{1}(\text{gBT}\ ;\ \mathbf{p}) = \sum_{k=2}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[\prod_{j \in \Psi_{k-1}} p_j \Big] \Big[\sum_{i=1}^{|\psi_k|} \prod_{j \in \psi_k(<i)} p_j \Big] \tag{131}$$

Of course these are just the same as their model-2 counterparts (128) and (129), except that the satisfiability ratios R_j characterizing the small class have been replaced by the satisfiability rates p_j characterizing the model-1 probability model itself. Thus using $\mathbf{p} = \mathbf{R}$ as suggested in section 3.5.1 of [19], in this case allows the model-1 results to agree *exactly* with their model-2 counterparts, as opposed to just providing good approximations. A similar situation occurred for the model-1 and model-2 expressions for the expected number of solutions, derived in section 3.5.2 of [19]. For gFC expected complexities below, the relationship between the model-1 and model-2 expressions are not this simple, and the asymptotic limit used in (113) of [19] must be invoked before exact equality of expectations is achieved. However, even for gFC the model-1 expectations have always been found to provide excellent approximations to their model-2 counterparts. (See chapter 6 of [18].)

3.1.4. Big-Classes Under Model-0

Probability model-0 is just a special case of model-1 where all satisfiability rates p_j are equal to the same value p. Note also that $|\ \psi_k(<i)\ | = i-1$, from the definition of $\psi_k(<i)$ in table 2-4. Thus, for the model-0 case over a big-class $\text{CLP}_{\beta}(n\ \mathbf{m}\ c\ \mathbf{Z})$, the gBT expected complexities (130) and (131) can be simplified to give respectively

$$\overline{N}_{\beta}^{0}(\text{gBT}\ ;\ p) = \sum_{k=1}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[p^{\,|\Psi_{k-1}|} \Big] \tag{132}$$

$$\overline{C}_{\beta}^{0}(\text{gBT}\ ;\ p) = \sum_{k=2}^{n} \Big[\prod_{x \in X_k} m_x \Big] \Big[p^{\,|\Psi_{k-1}|} \Big] \Big[\sum_{i=1}^{|\psi_k|} p^{\,i-1} \Big] \tag{133}$$

Special cases:

$p=1$:
$$\overline{N}_{\beta}^{0}(\text{gBT}\ ;\ p) = \sum_{k=1}^{n} \Big[\prod_{x \in X_k} m_x \Big] \tag{134}$$

$p=0$:
$$\overline{N}_{\beta}^{0}(\text{gBT}\ ;\ p) = \sum_{k=1}^{k} \Big[\prod_{x \in X_k} m_x \Big] \tag{135}$$

$p \neq 1$: $$\overline{C}_{\beta}^{0}(\text{gBT} ; p) = \sum_{k=k'}^{n} \left[\prod_{x \in X_k} m_x \right] \left[p^{|\Psi_{k-1}|} \right] \left[\frac{1 - p^{|\psi_k|}}{1 - p} \right] \tag{136}$$

$p = 1$: $$\overline{C}_{\beta}^{0}(\text{gBT} ; p) = \sum_{k=k'}^{n} \left[\prod_{x \in X_k} m_x \right] |\psi_k| \tag{137}$$

$p = 0$: $$\overline{C}_{\beta}^{0}(\text{gBT} ; p) = \prod_{x \in X_{k'}} m_x \tag{138}$$

(134) just says that every possible node is generated. This is expected, since for $p = 1$ each instance constraint is always satisfied by any combination of values of its arguments. Thus no constraint violation can occur to prevent a node from propagating successors and all m^k possible nodes given in (36) are generated at each level. Formally, (134) follows from (132) since $1^x = 1$, whether x equals 0 or not.

(135) says that all nodes are generated through to level k', the first level at which checkable constraints exist; see (127). This is of course also as expected, since when $p = 0$ each constraint is *never* satisfiable, no matter what values are assigned to its arguments. Thus as soon as there exists even a single constraint to check at a node, it will be found violated and that node will not propagate descendent nodes in the tree. Formally, (135) follows from (132) by use of (45). From the latter it follows that $\Psi_{k-1} = \emptyset$ for $k \leq k'$ so that $p^{|\Psi_{k-1}|} = 0^0 = 1$. For $k > k'$, $\Psi_{k-1} \neq \emptyset$ so that $p^{|\Psi_{k-1}|} = 0^{|\Psi_{k-1}|} = 0$.

(136) and (137) are obtained by analytically summing the geometric series in (133) for the corresponding two cases of p. When $p = 1$ every possible node is generated — as seen in (134) — and (137) agrees with the requirement that at each of these nodes each of the checkable constraints at that level is checked once, since no constraint violation can have occurred previously to prevent it being checked.

(138) can also be readily seen to give the required result as follows. When $p = 0$, every possible node is generated through to the first level k' at which there are checkable constraints — as mentioned in regards to (135). At each node of that level, exactly one constraint check occurs since it always results in a constraint violation and further constraint checking at those nodes is thus not performed. Result (138) is then the correct expected total, since no more checking occurs at later levels, and certainly none occurred at earlier levels where by definition no checkable constraints existed. Formally, (138) is obtained from (133) via the following three cases.

- $k < k'$ The contribution to (133) is 0 since by definition of k', for these k we have $\psi_k = \emptyset$. The sum over i in (133) is then degenerate with value 0.
- $k > k'$ The contribution to (133) is again 0 since $\Psi_{k-1} \neq \emptyset$ and hence $p^{|\Psi_{k-1}|} = 0^{|\Psi_{k-1}|} = 0$.
- $k = k'$ $\Psi_{k-1} = \emptyset$, so that $p^{|\Psi_{k-1}|} = 0^0 = 1$. Also $\psi_k \neq \emptyset$, so that the sum over i has value $0^0 = 1$ (corresponding to $i = 1$).

Thus (138) follows since the only non-zero contribution to the sum (133) occurs for $k = k'$.

Note that even for an individual instance *clp*, the number of nodes generated by either algorithm gBT or gFC is independent of the respective constraint-check orders ψ_k or ϕ_k used — as exemplified in figures 2-3, 2-4, 2-7 and 2-8. All expectations $\overline{N}(gBT)$ and $\overline{N}(gFC)$ above and below are therefore certainly independent of these orderings. On the other hand, the number of constraint-checks to solve an instance is a function of these orders for the two algorithms — as seen in figures 2-3, 2-4, 2-7 and 2-8. The expectations $\overline{C}(gBT)$ and $\overline{C}(gFC)$ thus both inherit this constraint-check order dependence in general. For gBT it expresses itself analytically in (129) and (131) through the products over $j \in \psi_k (< i)$. Note however that the specialization to the model-0 case of this section is sufficient to remove the constraint-check order dependence of $\overline{C}(gBT)$, even though the big-class concerned is still left arbitrary. We will see that for gFC below, significantly more specialization is required in the type of big-class before the constraint-check order no longer effects the expected number of checks performed.

Note also that since (128) and (129) have the same form as (130) and (131), they may also be specialized in the same way as the latter two were in this section, in the formally similar case of a small class

whose satisfiability ratios have the same value $R_j = R$ for all constraints. The results would just be the expressions of this section with p replaced throughout by R.

3.1.4.1. Uniform Degree-m, Pure & Simple A-ary Big-Classes

From table 2-6, we have that for pure and simple A-ary CLP instances $|\Psi_k| = \binom{k}{A}$ and $|\psi_k| = \binom{k-1}{A-1}$. When the instances are also of uniform domain size m, equations (132) to (138) reduce to

$$\overline{N}_p^0(\text{gBT} ; p) = \sum_{k=1}^{n} m^k \, p^{\binom{k-1}{A}} \tag{139}$$

$$\overline{C}_p^0(\text{gBT} ; p) = \sum_{k=A}^{n} m^k \, p^{\binom{k-1}{A}} \sum_{i=1}^{\binom{k-1}{A-1}} p^{i-1} \tag{140}$$

Special cases:

$p = 1$:
$$\overline{N}_p^0(\text{gBT} ; p) = \sum_{k=1}^{n} m^k \tag{141}$$

$p = 0$:
$$\overline{N}_p^0(\text{gBT} ; p) = \sum_{k=1}^{A} m^k \tag{142}$$

$p \neq 1$:
$$\overline{C}_p^0(\text{gBT} ; p) = \sum_{k=A}^{n} m^k \, p^{\binom{k-1}{A}} \frac{1 - p^{\binom{k-1}{A-1}}}{1 - p} \tag{143}$$

$p = 1$:
$$\overline{C}_p^0(\text{gBT} ; p) = \sum_{k=A}^{n} m^k \binom{k-1}{A-1} \tag{144}$$

$p = 0$:
$$\overline{C}_p^0(\text{gBT} ; p) = m^A \tag{145}$$

3.1.4.2. Uniform Degree-m, Pure & Simple Binary Big-Classes

Results for the uniform degree-m, pure and simple *binary* CLP under model-0 follow by using $A = 2$ in (139) to (145).

$$\overline{N}_p^0(\text{gBT} ; p) = \sum_{k=1}^{n} m^k \, p^{\binom{k-1}{2}} \tag{146}$$

$$\overline{C}_p^0(\text{gBT} ; p) = \sum_{k=2}^{n} m^k \, p^{\binom{k-1}{2}} \sum_{i=1}^{k-1} p^{i-1} \tag{147}$$

Special cases:

$p=1$: $$\overline{N}^{0}_{\beta}(\text{gBT}\ ;\ p) = \sum_{k=1}^{n} m^{k} \tag{148}$$

$p=0$: $$\overline{N}^{0}_{\beta}(\text{gBT}\ ;\ p) = \sum_{k=1}^{2} m^{k} \tag{149}$$

$p\neq 1$: $$\overline{C}^{0}_{\beta}(\text{gBT}\ ;\ p) = \sum_{k=2}^{n} m^{k}\ p^{\binom{k-1}{2}}\ \frac{1-p^{k-1}}{1-p} \tag{150}$$

$p=1$: $$\overline{C}^{0}_{\beta}(\text{gBT}\ ;\ p) = \sum_{k=2}^{n} m^{k}\ (k-1) \tag{151}$$

$p=0$: $$\overline{C}^{0}_{\beta}(\text{gBT}\ ;\ p) = m^{2} \tag{152}$$

It was this special uniform degree-m, pure and simple binary case that Haralick analyzed in [11] for both Backtracking and Forward Checking. Our results here agree with his. However, due to an error in Haralick's analysis of the gFC algorithm, we will see below that our gFC analog of (147) does *not* agree with that of Haralick.

3.2. Expected-Complexity of gFC

Paralleling the previous section on gBT, this section derives the expected numbers of nodes generated and constraint-checks performed in solving a CLP instance by the gFC algorithm of section 2.3. These expectations are obtained both over big-classes and over small-classes under the corresponding probability models 0, 1 and 2 of section 1.3. Again we find that much of the analysis can be presented in a model-independent manner, with specialization to individual models required only towards the end. As for gBT then, the sample space Ω below is to be taken as referring to either the generic big-class $\Omega_{\beta}(n\ \mathbf{m}\ c\ \mathbf{Z}) = \text{CLP}_{\beta}(n\ \mathbf{m}\ c\ \mathbf{Z})$ or the generic small-class $\Omega_{\sigma}(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S}) = \text{CLP}_{\sigma}(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$.

3.2.1. Class- and Model-Independent Expressions

In section 2.3.2, the number of nodes generated and the number of constraint checks performed by gFC in solving instance *clp* were expressed respectively as

Table 3-3: Some gFC-dependent subsets of $\Omega = \Omega_{\beta}(n\ \mathbf{m}\ c\ \mathbf{Z})$ or $\Omega_{\sigma}(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$.

Symbol	Defined as	Defined for
$E(\text{gFC}\ k\ \overline{X}_{k})$	$\{\ clp\ \mid\ \delta(\text{gFC}\ k\ \overline{X}_{k}\ clp) = 1\ \}$	$1 \le k \le n$
$E(\text{gFC}\ k\ i\ t\ \overline{X}_{k})$	$\{\ clp\ \mid\ \delta(\text{gFC}\ k\ i\ t\ \overline{X}_{k}\ clp) = 1\ \}$	$1 \le k \le n-1,\ 1 \le i \le \mid \phi_{k} \mid,$ $0 \le t \le m_{f_{k}^{i}}$
$E(\text{gFC}\ k\ i \ge 1\ \overline{X}_{k})$	$\{\ clp\ \mid\ \delta(\text{gFC}\ k\ i \ge 1\ \overline{X}_{k}\ clp) = 1\ \}$	$1 \le k \le n-1,\ 1 \le i \le \mid \phi_{k} \mid$

$$N(\text{gFC } clp) = \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gFC } k\ \bar{X}_k\ clp) \tag{153}$$

$$C(\text{gFC } clp) = \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} \sum_{t=0}^{m_{f_k^i}} t\ \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) \tag{154}$$

These are in terms of the indicator functions $\delta(\text{gFC } k\ \bar{X}_k\ clp)$ and $\delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp)$ of (80) and (83). The former function takes on value 1 if gFC generates a node at site $\bar{X}_k$ when solving clp, and otherwise has value 0. The latter indicator function has value 1 if gFC checks constraint $C_{\phi_k^i}$ exactly t times at site $\bar{X}_k$ in solving clp, and otherwise has value 0.

In (153) and (154) the instance clp is fixed and the values of k, $\bar{X}_k$, i and t are varied. One may however fix the latter values and let the instance clp vary. This is done in defining the sample-space subsets in table 3-3. The first two of these are respectively the set of instances in Ω for which gFC generates a node at site $\bar{X}_k$ and the set of instances for which gFC checks constraint $C_{\phi_k^i}$ exactly t times at site $\bar{X}_k$. The other set is defined in terms of a third indicator function $\delta(\text{gFC } k\ i \geq 1\ \bar{X}_k\ clp)$, given in (86), so that $E(\text{gFC } k\ i \geq 1\ \bar{X}_k)$ is the set of instances for which gFC checks constraint $C_{\phi_k^i}$ at least once at site $\bar{X}_k$. These sets of instances are defined directly in terms of the behavior of algorithm gFC itself. They can however be expressed in a more useful form as follows, in terms of properties of the instances themselves. For this we make use of the constraint-based sets defined in table 1-3.

$$E(\text{gFC } k\ \bar{X}_k) = E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k) \tag{155}$$

$$E(\text{gFC } k\ i\ t\ \bar{X}_k) = E(\Psi_k\ \bar{X}_k) \cap E(\Phi_{kf_k^i}(<i)\ f_k^i\ t\ \bar{X}_k) \bigcap_{g \in F_k - f_k^i} E(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \tag{156}$$

$$E(\text{gFC } k\ i \geq 1\ \bar{X}_k) = E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \tag{157}$$

$$E(\text{gFC } k\ \bar{X}_k) = E(\text{gFC } k\ 1 \geq 1\ \bar{X}_k) \tag{158}$$

---

(155) says that an instance has node $\bar{X}$ generated by gFC iff for that instance

(a) $\bar{X}_k$ is a consistent labeling of X_k (so that all applicable constraints — those in Ψ_k — are satisfied) and

(b) each future variable $g \in F_k$ has at least one value that allows an extension of $\bar{X}_k$ that is consistent with respect to the constraints of $\Phi_{k-1,g}$.

As mentioned in obtaining (60), if gFC generates a node $\bar{X}_k$ in solving a given instance then $\bar{X}_k$ is consistent. The instance thus belongs to $E(\Psi_k\ \bar{X}_k)$ and we have that $E(\text{gFC } k\ \bar{X}_k) \subseteq E(\Psi_k\ \bar{X}_k)$. Moreover, if a node is generated at $\bar{X}_k$ then each future variable $g \in F_k$ must have at least one still viable value at the end of all constraint checking at its parent node, otherwise the child node $\bar{X}_k$ would not have been generated due to an earlier domain wipe-out. The constraints checked against variable g to the end of checking at level k–1 are those in $\Phi_{k-1,g}$ and thus $E(\text{gFC } k\ \bar{X}_k) \subseteq \bigcap_{g \in F_k} E(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k)$. Combining the above two results we have

$$E(\text{gFC } k\ \bar{X}_k) \subseteq E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k) \tag{159}$$

In the other direction, if an instance is such that $\bar{X}_k$ is a consistent labeling and each future variable

$g \in F_k$ has at least one value that provides an extension of $\bar{X}_k$ that is consistent with respect to the constraints in $\Phi_{k-1,g}$ then gFC will generate node $\bar{X}_k$ for that instance because at each ancestor node of $\bar{X}_k$ there can be no reason for any domain wipe-outs. We therefore have that

$$E(\text{gFC } k\ \bar{X}_k) \supseteq E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k) \tag{160}$$

Together, (159) and (160) establish (155). Note that since $F_k = \emptyset$ for $k = n$, then at level n (155) becomes $E(\text{gFC } n\ \bar{X}_n) = E(\Psi_n\ \bar{X}_n)$ and hence $P(\text{gFC } n\ \bar{X}_n) = P(\Psi_n\ \bar{X}_n)$.

(156) extends (155) slightly. It says that an instance has constraint $C_{\phi_k^i}$ checked by gFC exactly t times at site $\bar{X}_k$ iff for that instance

(a) $\bar{X}_k$ is consistent (part of the condition for node $\bar{X}_k$ to be generated) and

(b) just before checking constraint $C_{\phi_k^i}$, the corresponding future variable f_k^i to be filtered has exactly t still viable values, and

(c) all other future variables have at least one still viable value left (otherwise a domain wipe-out would already have occurred, thus preventing the checking of $C_{\phi_k^i}$ from being reached). The proof for $1 \leq t \leq m_{f_k^i}$ is similar to that of (155). For $t = 0$ equation (156) is not valid, since for example an instance may be in $E(\text{gFC } k\ i\ 0\ \bar{X}_k)$ simply because no node was generated at site $\bar{X}_k$.

(157) is derived as follows:

$$E(\text{gFC } k\ i \geq 1\ \bar{X}_k)$$

$$= \bigcup_{t=1}^{m_{f_k^i}} E(\text{gFC } k\ i\ t\ \bar{X}_k) = \Omega - E(\text{gFC } k\ i\ 0\ \bar{X}_k) \quad \text{by (86)}$$

$$= E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k - f_k^i} E(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \bigcap \left[\bigcup_{t=1}^{m_{f_k^i}} E(\Phi_{kf_k^i}(<i)\ f_k^i\ t\ \bar{X}_k) \right] \quad \text{by (156)}$$

$$= E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k - f_k^i} E(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \bigcap E(\Phi_{kf_k^i}(<i)\ f_k^i \geq 1\ \bar{X}_k)$$

$$= E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k)$$

(158) is the analog of (113) for gBT. It follows since a node is generated by gFC at a level-k node site $\bar{X}_k$ iff the *first* checkable constraint *is* actually checked (at least once) at that site. The equality however is not valid at level $k = n$, where gFC has no constraints to check.[22] Note that (158) can also be obtained from (155), (157) and (92), since from these we have

$$E(\text{gFC } k\ 1 \geq 1\ \bar{X}_k) = E(\Psi_k\ \bar{X}_k) \bigcap_{g \in F_k} E(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k) = E(\text{gFC } k\ \bar{X}_k)$$

With these preliminary results we can now derive the expected value of $N(\text{gFC } clp)$ and of $C(\text{gFC } clp)$ as follows.

[22] Unlike gBT, algorithm gFC has already performed all constraint checking before level n. However, also unlike gBT, algorithm gFC *does* do constraint checking at level 1 — binary constraints are already checkable there because besides variable x_1, a future variable is also allowed as one of the arguments.

$$\bar{N}(\text{gFC}) = \sum_{clp \in \Omega} N(\text{gFC } clp)\, P(clp) \quad \text{by (11)} \tag{161}$$

$$= \sum_{clp \in \Omega} \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \delta(\text{gFC } k\ \bar{X}_k\ clp)\, P(clp) \quad \text{by (153)} \tag{162}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \sum_{clp \in \Omega} \delta(\text{gFC } k\ \bar{X}_k\ clp)\, P(clp) \tag{163}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} P(\text{gFC } k\ \bar{X}_k) \quad \text{by (13)} \tag{164}$$

$$= \sum_{k=1}^{n} \sum_{\bar{X}_k \in D_{X_k}} \Big[\prod_{j \in \Psi_k} P(j\ \bar{X}_k) \Big] \Big[\prod_{g \in F_k} P(\Phi_{k-1,g}\ g \geq 1\ \bar{X}_k) \Big] \tag{165}$$

by (155) and T_j independence

$$\bar{C}(\text{gFC}) = \sum_{clp \in \Omega} C(\text{gFC } clp)\, P(clp) \quad \text{by (11)} \tag{166}$$

$$= \sum_{clp \in \Omega} \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} \sum_{t=0}^{m_{f_k^i}} t\ \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp)\, P(clp) \quad \text{by (154)} \tag{167}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} \sum_{t=0}^{m_{f_k^i}} t \sum_{clp \in \Omega} \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp)\, P(clp) \tag{168}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} \sum_{t=0}^{m_{f_k^i}} t\, P(\text{gFC } k\ i\ t\ \bar{X}_k) \quad \text{by (13)} \tag{169}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \Big[\prod_{j \in \Psi_k} P(j\ \bar{X}_k) \Big] \cdots$$

$$\cdots \Big[\sum_{i=1}^{|\phi_k|} \prod_{g \in F_k - f_k^i} P(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \sum_{t=0}^{m_{f_k^i}} t\, P(\Phi_{kf_k^i}(<i)\ f_k^i\ t\ \bar{X}_k) \Big] \tag{170}$$

by (156) and T_j independence

Note that in this latter expression we could have used

$$k' = \text{Min}\{\ k\ \mid\ 1 \leq k \leq n-1 \text{ such that } \mid \phi_k \mid \neq 0\ \} \tag{171}$$

as the lower value for k, instead of $k = 1$. An analogous situation occurred for gBT where k' of (127) could have been used in (126). In both cases, use of the corresponding k' simply reflects the fact that no constraint-checks are performed at levels where there are no constraints to check. Note also that we have used in (169) the expansion (156) even for the $t = 0$ case, when the expansion is invalid. This use however is justified because of the multiplication of $P(\text{gFC } k\ i\ t\ \bar{X}_k)$ by t in (169), so that at $t = 0$ the erroneous form for $P(\text{gFC } k\ i\ t\ \bar{X}_k)$ becomes irrelevant.

Expressions (165) and (170) for $\overline{N}$(gFC) and $\overline{C}$(gFC) require expanded forms of the probabilities $P(j\ \overline{X}_k)$, $P(J\ f\ t\ \overline{X}_k)$ and $P(J\ f\ \geq 1\ \overline{X}_k)$. These *are* dependent on the probability model used, and the following subsections complete the derivations for each model individually as appropriate. Note that as for gBT above, we find in each case that the undesirable sum over $\overline{X}_k$ in (165) and (170) can be removed because $P(j\ \overline{X}_k)$, $P(J\ f\ t\ \overline{X}_k)$ and $P(J\ f\ \geq 1\ \overline{X}_k)$ are all independent of $\overline{X}_k$.

3.2.2. Small-Classes Under Model-2

The gFC expected complexities (165) and (170) may be specialized for the case of a small-class $\mathrm{CLP}_\sigma(n\ \mathbf{m}\ c\ \underline{\mathbf{Z}}\ \mathbf{S})$ under model-2 by use of the model-2 versions of $P(j\ \overline{X}_k)$, $P(J\ f\ \geq 1\ \overline{X}_k)$ and $\sum t\ P(J\ f\ t\ \overline{X}_k)$ from table 1-4 and equation (19). Moreover, since the substituted expressions are all independent of the particular $\overline{X}_k$ involved, this allows the sums over $\overline{X}_k$ in (165) and (170) to be collapsed using (36). This is analogous to what happened for gBT above, but there only the expression for $P(j\ \overline{X}_k)$ was needed. The resulting expected complexity expressions for gFC are then

$$\overline{N}^2_\sigma(\mathrm{gFC}) = \sum_{k=1}^{n} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} R_j \right] \left[\prod_{g \in F_k} P^2_\sigma(\Phi_{k-1,g}\ g \geq 1) \right] \tag{172}$$

$$\overline{C}^2_\sigma(\mathrm{gFC}) =$$

$$\sum_{k=1}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} R_j \right] \left[\sum_{i=1}^{|\phi_k|} m_{f^i_k} \prod_{j \in \Phi_{k f^i_k}(<i)} R_j \prod_{g \in F_k - f^i_k} P^2_\sigma(\Phi_{kg}(<i)\ g \geq 1) \right] \tag{173}$$

$$\text{where } P^2_\sigma(J\ f\ \geq 1) = 1 - \frac{\sum_{l=0}^{m_J} (-1)^l \binom{m_J}{l} \prod_{j \in J} \binom{M_j - l}{S_j - l}}{\prod_{j \in J} \binom{M_j}{S_j}} \tag{174}$$

Note that the argument $\overline{X}_k$ has been dropped throughout since its value is not relevant. Expressions (172) and (173) correspond respectively to the model-2 results (128) and (129) for gBT.

3.2.3. Big-Classes Under Model-1

The specialization of the gFC expected complexities (165) and (170) for the case of a big-class under model-1 is analogous to that above for small-classes under model-2. But now the model-1 versions of $P(j\ \overline{X}_k)$, $P(J\ f\ \geq 1\ \overline{X}_k)$ and $\sum t\ P(J\ f\ t\ \overline{X}_k)$ are required, from table 1-4 and equation (18). (In obtaining the corresponding model-1 result for gBT above, only the first of these was needed.) Again, since each of the substituted expressions is independent of the particular $\overline{X}_k$ involved, the sums over $\overline{X}_k$ in (165) and (170) can be collapsed using (36). The resulting gFC expected complexities for a big-class $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ under model-1 with satisfiability-rates vector $\mathbf{p} = (p_1\ p_2\ \ldots\ p_c)$ are therefore

$$\overline{N}^1_\beta(\mathrm{gFC}\ ;\ \mathbf{p}) = \sum_{k=1}^{n} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} p_j \right] \left[\prod_{g \in F_k} P^1_\beta(\Phi_{k-1,g}\ g \geq 1\ ;\ \mathbf{p}) \right] \tag{175}$$

$$\overline{C}^1_\beta(\mathrm{gFC}\ ;\ \mathbf{p}) =$$

$$\sum_{k=1}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} p_j \right] \left[\sum_{i=1}^{|\phi_k|} m_{f^i_k} \prod_{j \in \Phi_{k f^i_k}(<i)} p_j \prod_{g \in F_k - f^i_k} P^1_\beta(\Phi_{kg}(<i)\ g \geq 1\ ;\ \mathbf{p}) \right] \tag{176}$$

$$\text{where } P^1_\beta(J\ f\ \geq 1\ ;\ \mathbf{p}) = 1 - \left[1 - \prod_{j \in J} p_j \right]^{m_J} \tag{177}$$

Note that again the argument $\overline{X}_k$ has been dropped throughout since its value is not relevant. Expressions (175) and (176) correspond respectively to the model-1 results (130) and (131) for gBT. Due to the differences between $P^1_\beta(J\ f\ \geq 1\ ;\ \mathbf{p})$ and $P^2_\sigma(J\ f\ \geq 1)$ in (177) and (174) respectively, the above

model-1 and model-2 expected complexities for gFC — unlike their counterparts for gBT — are not related to each other by a simple interchange of parameters p_j and R_j. The model-2 expressions are in fact considerably more complex to compute values for, due to the greater complexity of expression (174) compared with its counterpart (177). This is why it is important to note that in the limit used in (113) of [19], the latter two probability expressions give the same value when parameters p_j are set equal to the value of their corresponding R_j. The model-2 expected complexities (172) and (173) also therefore become asymptotically equal to their respective model-1 counterparts (175) and (176) — this being a special case of the relationship between model-1 and model-2 expectations discussed in section 3.5.1 of [19].

In fact, in practice we find that the above asymptotic convergence of the corresponding gFC complexities under models 1 and 2 is apparently quite fast, and setting $p_j = R_j$ in the model-1 result (175) or (176) has always been found to provide an excellent approximation for the corresponding model-2 result (172) or (173). (See for example chapter 6 of [18].) Since gFC model-1 expressions are simpler to compute with than the model-2 expressions, their ability to approximate model-2 results becomes important when (as in [21]), we consider augmenting gFC with theory-based heuristics. For gBT however, the model-2 and model-1 expressions obtained earlier have the same structure and there is therefore no gain in computational speed, nor for that matter any loss in accuracy, when model-1 results are used to approximate those of model-2.

3.2.4. Big-Classes Under Model-0

Probability model-0 is just a special case of model-1 where all satisfiability rates p_j are equal to the same value p. For the model-0 case over a big-class $\mathrm{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$, the gFC expected complexities (175) and (176) can therefore be simplified to give respectively

$$\overline{N}_\beta^0(\mathrm{gFC}\ ;\ p) = \sum_{k=1}^{n} \left[\prod_{x \in X_k} m_x \right] \left[p^{\,|\Psi_k|} \right] \left[\prod_{g \in F_k} P_\beta^0(\Phi_{k-1,g}\ \ g \geq 1\ ;\ p) \right] \tag{178}$$

$$\overline{C}_\beta^0(\mathrm{gFC}\ ;\ p) =$$

$$\sum_{k=1}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[p^{\,|\Psi_k|} \right] \left[\sum_{i=1}^{|\phi_k|} m_{f_k^i}\, p^{\,|\Phi_{kf_k^i}(<i)|} \prod_{g \in F_k - f_k^i} P_\beta^0(\Phi_{kg}(<i)\ g \geq 1\ ;\ p) \right] \tag{179}$$

$$\text{where } P_\beta^0(J\ f\ \geq 1\ ;\ p) = 1 - \left[1 - p^{\,|J|} \right]^{m_f} \tag{180}$$

Special cases:

$p = 1$:
$$\overline{N}_\beta^0(\mathrm{gFC}\ ;\ p) = \sum_{k=1}^{n} \left[\prod_{x \in X_k} m_x \right] \tag{181}$$

$p = 0$:
$$\overline{N}_\beta^0(\mathrm{gFC}\ ;\ p) = \sum_{k=1}^{k'} \left[\prod_{x \in X_k} m_x \right] \tag{182}$$

$p = 1$:
$$\overline{C}_\beta^0(\mathrm{gFC}\ ;\ p) = \sum_{k=k'}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[\sum_{i=1}^{|\phi_k|} m_{f_k^i} \right] \tag{183}$$

$p = 0$:
$$\overline{C}_\beta^0(\mathrm{gFC}\ ;\ p) = \left[\prod_{x \in X_{k'}} m_x \right] m_{f_{k'}^1} \tag{184}$$

(181) just says that every possible node is generated. This is correct for $p = 1$, since then each instance constraint is always satisfied by any combination of values of its arguments and no constraint violation ever occurs. Thus no domain wipe-out ever prevents any node from propagating successors and all m^k possible nodes given in (36) are generated at each level. Formally, (181) follows from (178)

because $1^x = 1$, whether x equals 0 or not.

(182) says that all nodes are generated through to the first level k' at which forward checkable constraints exist; see (171). This is of course also as expected, since when $p = 0$ each constraint is *never* satisfiable, no matter what values are assigned to its arguments. Thus as soon as there exists even a single constraint to forward check at a node, it will be found violated and that node will not propagate descendent nodes in the tree. Formally, (182) is obtained from (178) in a manner analogous to that for obtaining (135) from (132) for gBT. It is necessary to note that for $k \leq k'$, $\Psi_k = \emptyset$ and $\Phi_{k-1,g} = \emptyset$ for all $g \in F_k$. But for $k > k'$, $\Phi_{k-1,g} \neq \emptyset$ for some $g \in F_k$. See (50), (69) and the definition of Φ_{kf} in table 2-9.

(183) can be seen to be correct for $p = 1$ because in this case every possible node is generated — as implicit in (181) — and at each of these nodes each of the forward-checkable constraints $C_{\phi_k^1}$ at that level is checked once for each of the $m_{f_k^1}$ original candidate values in $d_{f_k^1}$ for the corresponding future variable f_k^1 whose domain is being filtered. Each of the original values in $d_{f_k^1}$ is checked since with $p = 1$, no constraint violation can have occurred previously to cause it to be filtered out. Formally, (183) follows from (179) because $1^x = 1$, whether x equals 0 or not.

(184) is also readily seen to be correct. When $p = 0$, every possible node is generated through to the first level k' at which there are forward-checkable constraints — as seen in (182). At each node of level k', the first forward-checkable constraint $C_{\phi_{k'}^1}$ is checked once for each of the $m_{f_{k'}^1}$ values in the corresponding domain $d_{f_{k'}^1}$ being filtered. Since $p = 0$, all these values are found to violate the constraint. They are thus all filtered out of their domain, resulting in a domain wipe-out at each node straight after the first constraint has been forward-checked there and precluding any further constraint checking at those nodes. Result (184) is then the correct expected total, since no more checking occurs at later levels, and certainly none occurred at earlier levels where by definition no forward-checkable constraints existed. Formally, (184) is obtained from (179) via the following three cases.

- $k < k'$ The contribution to (179) is 0 since by definition of k' we have $\phi_k = \emptyset$. The sum over i in (179) is then degenerate with value 0.
- $k > k'$ The contribution to (179) is again 0 since by (50) we have $\Psi_k \neq \emptyset$ and hence $p^{|\Psi_k|} = 0^{|\Psi_{k-1}|} = 0$.
- $k = k'$ $\Psi_k = \emptyset$ so that $p^{|\Psi_{k-1}|} = 0^0 = 1$. Also $\phi_k \neq \emptyset$, so that the sum over i is non-zero with value $m_{f_{k'}^1}$, corresponding to $i = 1$ where both $\Phi_{kf_{k'}^1}(<i) = 0$ and $\Phi_{kg}(<i) = 0$ for all $g \in F_k - f_k^1$. For $i > 1$, at least one of these sets is non-empty, allowing no contribution to the $k = k'$ case of (179) for these higher i values. See the definition of $\Phi_{kf}(<i)$ in table 2-9.

Thus (184) follows since the only non-zero contribution to the sum (179) occurs for $k = k'$.

3.2.4.1. Uniform Degree-m, Pure & Simple A-ary Big-Classes

For a pure and simple A-ary instance we have from (53) or (104) that $|\Psi_k| = \binom{k}{A}$. When instances are also of uniform degree m, expressions (178) and (179) therefore become

$$\overline{N}_\beta^0(\text{gFC} ; p) = \sum_{k=1}^{n} m^k \, p^{\binom{k}{A}} \prod_{g \in F_k} P_\beta^0(\Phi_{k-1,g} \; g \geq 1 ; p) \tag{185}$$

$$\overline{C}_\beta^0(\text{gFC} ; p) = \sum_{k=A-1}^{n-1} m^{k+1} \, p^{\binom{k}{A}} \left[\sum_{i=1}^{|\phi_k|} p^{|\Phi_{kf_k^1}(<i)|} \prod_{g \in F_k - f_k^1} P_\beta^0(\Phi_{kg}(<i) \; g \geq 1 ; p) \right] \tag{186}$$

$$\text{where } P_\beta^0(J \; f \geq 1 ; p) = 1 - \left[1 - p^{|J|} \right]^m \tag{187}$$

Since from (102) we have that $|\Phi_{kf}| = \binom{k}{A-1}$, the same for all future variables $f \in F_k$, expression

(185) becomes

$$\overline{N}_{\beta}^{0}(\text{gFC} ; p) = \sum_{k=1}^{n} m^{k} \; p^{\binom{k}{A}} \left(1 - [1 - p^{\binom{k-1}{A-1}}]^{m}\right)^{n-k} \tag{188}$$

Special cases:

$$p=1: \qquad \overline{N}_{\beta}^{0}(\text{gFC} ; p) = \sum_{k=1}^{n} m^{k} \tag{189}$$

$$p=0: \qquad \overline{N}_{\beta}^{0}(\text{gFC} ; p) = \sum_{k=1}^{A-1} m^{k} \tag{190}$$

These latter two cases may be obtained directly from (188) in a manner analogous to that for obtaining (181) and (182) from (178). Or, they may be obtained by further specialization of (181) and (182) themselves. Either way, it should be noted that for the present case $| \phi_k | = \binom{k-1}{A-2}(n-k)$ from (101), and therefore the smallest value of k for which $| \phi_k | \neq 0$ is $k = A-1$.

To expand (186) in the analogous way to the above expansion of (185) requires the value of $| \Phi_{kf} (<i) |$. But, whereas $| \Phi_{k-1,f} |$ in (185) did not depend on more than the numbers $| \phi_k |$ of the forward checkable constraints at each level up to $k-1$, we see from table 2-9 that the value of $| \Phi_{kf} (<i) |$ depends not only on $| \phi_k |$ but also on the actual ϕ_k order used at level k. Thus further expansion of (186) is not possible without making additional assumptions about the orderings ϕ_k. Note that in the analogous situation for gBT, in obtaining (143) from (136), the precise ordering of ψ_k was irrelevant since (143) depended on ψ_k only through $| \psi_k |$.

In sections 2.3.1 and 2.3.3 we discussed two extreme classes of constraint orderings ϕ_k — f-cyclic and f-exhaustive orderings. For the situation of this section it can be shown that f-exhaustive orderings result in the minimal number of expected checks, while f-cyclic orderings give the maximal number of expected checks.[23] We therefore consider here the case of f-exhaustive constraint orderings ϕ_k. In the next section we specialize these results to pure and simple *binary* instances in which case the distinction between f-exhaustive and f-cyclic orderings disappears as will be explained.

As described in sections 2.3.1 and 2.3.3, an f-exhaustive constraint-check ordering completes the checking of all constraints of ϕ_{kf} relevant to a given future variable f before proceeding to check those relevant to another future variable. And as given in (68), this means that an f-exhaustive ordering ϕ_k is some concatenation

$$\phi_k = \phi_{kf_{k1}} \,||\, \phi_{kf_{k2}} \,||\, \ldots \,||\, \phi_{kf_{k,n-k}} \tag{191}$$

of orderings ϕ_{kf} — the sets of constraints forward-checkable at level k against the various future variables $f \in F_k$. The future variables may however be chosen in any order, and f_{kt} denotes the t-th such variable chosen in forming the f-exhaustive ordering ϕ_k. The j-th element in the t-th sub-ordering $\phi_{kf_{kt}}$ of ϕ_k is denoted $\phi_k^{jf_{kt}}$. Hence the elements of ϕ_k can be traversed in order from the first ϕ_k^1 to the last $\phi_k^{|\phi_k|}$ by varying j from 1 to $| \phi_{kf_{kt}} |$ for each t from 1 to $n-k$. Using this double indexing, (186) may be rewritten as

$$\overline{C}_{\beta}^{0}(\text{gFC} ; p) = \sum_{k=A-1}^{n-1} m^{k+1} \; p^{\binom{k}{A}} \sum_{t=1}^{n-k} \sum_{j=1}^{|\phi_{kf_{kt}}|} p^{|\Phi_{kf_{kt}}(<tj)|} \prod_{g \in F_k - f_{kt}} P_{\beta}^{0}(\Phi_{kg}(<tj)\; g \geq 1 ; p) \tag{192}$$

where $\Phi_{kf} (<tj)$ is the double-index form of the previously used set $\Phi_{kf} (<i)$. Explicitly, since $\Phi_{kf} (<i)$ is defined as

[23] This is not true for expected number of checks in the general case and is the reason we have maintained fully general constraint orderings till now. Nor is it true in general for *individual* instances of the present special-form big-class, but only in regards to the *expected* complexity of this class.

$$\Phi_{kf}(<i) = \Phi_{k-1,f} \cup (\phi_{kf} \cap \{\phi_k^1 \ldots \phi_k^{i-1}\}) \tag{193}$$

this means that

$$\Phi_{kf}(<tj) = \Phi_{k-1,f} \cup (\phi_{kf} \cap \{\phi_{kf_{k1}}^1 \ldots \phi_{kf_{kt}}^{j-1}\}) \tag{194}$$

From this we see that

$$\Phi_{kf_{ks}}(<tj) = \begin{cases} \Phi_{k-1,f_{ks}} \cup \phi_{kf_{ks}} = \Phi_{k,f_{ks}} & \text{if } 1 \le s \le t-1 \\ \Phi_{k-1,f_{ks}} \cup \emptyset = \Phi_{k-1,f_{ks}} & \text{if } t+1 \le s \le n-k \\ \Phi_{k-1,f_{ks}} \cup \{ \phi_{kf_{ks}}^1 \ldots \phi_{kf_{ks}}^{j-1} \} & \text{if } s = t \end{cases} \tag{195}$$

Therefore, since $| \Phi_{kf} | = \binom{k}{A-1}$ from (102), we have

$$| \Phi_{kf_{ks}}(<tj) | = \begin{cases} \binom{k}{A-1} & \text{if } 1 \le s \le t-1 \\ \binom{k-1}{A-1} & \text{if } t+1 \le s \le n-k \\ \binom{k-1}{A-1} + j - 1 & \text{if } s = t \end{cases} \tag{196}$$

Using these together with $| \phi_{kf} | = \binom{k-1}{A-2}$ from (100), and substituting in from (187), we can express (192) as

$$\overline{C}_\beta^0(\text{gFC} ; p) =$$

$$\sum_{k=A-1}^{n-1} m^{k+1} p^{\binom{k}{A}+\binom{k-1}{A-1}} \left[\sum_{j=1}^{\binom{k-1}{A-2}} p^{j-1} \right] \left[\sum_{t=1}^{n-k} (1 - [1 - p^{\binom{k}{A-1}}]^m)^{t-1} (1 - [1 - p^{\binom{k-1}{A-1}}]^m)^{n-k-t} \right] \tag{197}$$

Special cases:

$p = 1$: $$\overline{C}_\beta^0(\text{gFC} ; p) = \sum_{k=A-1}^{n-1} m^{k+1} \binom{k-1}{A-2} (n-k) \tag{198}$$

$p = 0$: $$\overline{C}_\beta^0(\text{gFC} ; p) = m^A \tag{199}$$

$p \neq 0, 1$: $$\overline{C}_\beta^0(\text{gFC} ; p) =$$

$$\sum_{k=A-1}^{n-1} m^{k+1} p^{\binom{k}{A}+\binom{k-1}{A-1}} \frac{1 - p^{\binom{k-1}{A-2}}}{1-p} (1 - [1 - p^{\binom{k-1}{A-1}}]^m)^{n-k-1} \frac{1 - \left[\dfrac{1 - [1 - p^{\binom{k}{A-1}}]^m}{1 - [1 - p^{\binom{k-1}{A-1}}]^m} \right]^{n-k}}{1 - \left[\dfrac{1 - [1 - p^{\binom{k}{A-1}}]^m}{1 - [1 - p^{\binom{k-1}{A-1}}]^m} \right]} \tag{200}$$

(198) is as required, since when $p = 1$ there is never any constraint violation and no values are filtered from any domain. Thus there will always be m constraint checks made for each of the $\binom{k-1}{A-2}$ constraints of ϕ_{kf} for each of the $n-k$ future variables $f \in F_k$ at each of the m^k nodes at level k — and this occurs at each level from $k = A-1$, the first at which an A-ary constraint can be forward checked, to $k = n-1$, the last for which there are any constraints to forward check. Note that (198)

is valid independent of the earlier assumption of f-exhaustive constraint orderings. For $p = 1$ all orderings are equivalent.

(199) is also as required, since for $p = 0$ there will be a domain wipe-out at each node straight after the first constraint is forward-checked at that node. This occurs at each of the m^{A-1} nodes at level $k = A-1$ since (as mentioned for the $p = 1$ case) level $k = A-1$ is the first level where A-ary constraints can be forward-checked. At each of these nodes at level $k = A-1$ the first constraint is checked once for each of the m values of its corresponding future variable, resulting in a domain wipe-out for that variable and termination of processing at that node. Thus $m^A = m^{A-1} \times m$ constraint-checks are performed in all when $p = 0$. Note that like (198), (199) is valid independent of the earlier assumption of f-exhaustive constraint orderings. For $p = 0$ all orderings are equivalent.

(200) is obtained from (197) by analytically summing the geometric series given by the sum over j and the one obtained from the sum over t after dividing its summand by $(1 - [1 - p^{\binom{k-1}{A-1}}]^m)^{n-k-1}$ and taking this factor out the front. Since $p \neq 0$ is assumed in (200), this latter division is legal for all $A-1 \leq k \leq n-1$. Moreover, since (200) also assumes $p \neq 1$, in both resulting geometric series the ratio of successive terms does not equal 1 so that the usual summation formula can be applied.

Note that the results of this sub-section subsume those published earlier in [22], providing the completion anticipated in that paper and agreeing of course with the results that appeared there.

3.2.4.2. Uniform Degree-m, Pure & Simple Binary Big-Classes

The results of the previous section may be specialized to the binary case by setting the arity to $A = 2$, in which case (188), (189), (190) and (197) to (200) become

$$\overline{N}_\beta^0(\text{gFC} ; p) = \sum_{k=1}^{n} m^k \, p^{\binom{k}{2}} (1 - [1 - p^{k-1}]^m)^{n-k} \tag{201}$$

$$\overline{C}_\beta^0(\text{gFC} ; p) = \sum_{k=1}^{n-1} m^{k+1} \, p^{(k-1)(k+2)/2} \sum_{t=1}^{n-k} (1 - [1 - p^k]^m)^{t-1} (1 - [1 - p^{k-1}]^m)^{n-k-t} \tag{202}$$

Special cases:

$p = 1$: $$\overline{N}_\beta^0(\text{gFC} ; p) = \sum_{k=1}^{n} m^k \tag{203}$$

$p = 0$: $$\overline{N}_\beta^0(\text{gFC} ; p) = m \tag{204}$$

$p = 1$: $$\overline{C}_\beta^0(\text{gFC} ; p) = \sum_{k=1}^{n-1} m^{k+1} (n-k) \tag{205}$$

$p = 0$: $$\overline{C}_\beta^0(\text{gFC} ; p) = m^2 \tag{206}$$

$p \neq 0, 1$: $$\overline{C}_\beta^0(\text{gFC} ; p) =$$

$$\sum_{k=1}^{n-1} m^{k+1} \, p^{(k-1)(k+2)/2} (1 - [1 - p^{k-1}]^m)^{n-k-1} \, \frac{1 - \left[\dfrac{1 - [1 - p^k]^m}{1 - [1 - p^{k-1}]^m}\right]^{n-k}}{1 - \left[\dfrac{1 - [1 - p^k]^m}{1 - [1 - p^{k-1}]^m}\right]} \tag{207}$$

The expected number of nodes generated by gBT or gFC is never a function of the constraint-check order used, but this is generally not the case for the expected number of constraint-checks performed. In fact (197) is valid only when gFC uses f-exhaustive orderings at each level. In spite of this, its binary version (and corresponding binary special cases) of this section are nevertheless valid for *arbitrary* constraint-check orders at each level. This is so because for the pure and simple binary case we have at each level

that $|\phi_{kf}| = \binom{k-1}{A-2} = \binom{k-1}{0} = 1$ for each future variable $f \in F_k$. This means that not only does the distinction between f-exhaustive and f-cyclic orderings disappear but in fact, every one of the possible $|\phi_k|!$ permutations of ϕ_k can then be considered a degenerate f-exhaustive order[24] and (202) and its special cases (205) to (207) are therefore applicable to any family of orderings ϕ_k whatsoever. For gBT, we saw that independence from the constraint-check orders ψ_k was achieved under model-0 for *arbitrary* big-class (see (133)) rather than requiring the highly specialized type of big-class of this section.

Note that Haralick in [11] analyzed the same situation as treated here, as well as its gBT analogue treated above in section 3.1.4.2. His results agree with ours, apart from the expression for $\overline{C}^0_\beta(\text{gFC}\ ;\ p)$. The version in [11] incorrectly leaves out the $1 - [1 - p^{k-1}]^m$ term in the numerator and denominator of the fraction in (207) above.

3.3. Expected-Complexity of gwFC

This section analyses the complexity of solving CLP instances using algorithm gwFC, described in section 2.4. As mentioned there, this algorithm is a form of gFC, modified to take advantage of the computer's parallel bit-handling capabilities. The modification involved does not change the node structure — nor, in particular, the number of nodes generated — in the search tree for any given instance. Hence the various expectations $\overline{N}^1_\beta(\text{gFC})$ and $\overline{N}^2_\sigma(\text{gFC})$ derived in section 3.2 remain valid for $\overline{N}^1_\beta(\text{gwFC})$ and $\overline{N}^2_\sigma(\text{gwFC})$ under the corresponding situations.

Rather than effect the node structure of a search tree, the modification of gFC to obtain gwFC effects the amount of work performed *at* the various nodes. Machine-*anding* is used in gwFC to perform, in parallel, multiple individual constraint-checks of gFC. Corresponding to gFC's complexity measure $C(\text{gFC } clp)$, the number of constraint-checks performed in solving an instance clp, we now have the measure $A(\text{gwFC } clp)$, the number of machine *ands* performed by gwFC in solving clp.

As mentioned in section 2.4, we do not restrict ourselves to the case where the domain of each variable can be represented by a single machine word. We allow an arbitrary domain size m_{z_i} for each variable z_i and an arbitrary number b of bits in a machine word. As given in (107), domain d_{z_i} then requires $w_{z_i} = \lceil m_{z_i}/b \rceil$ machine words for its representation.

3.3.1. Class- and Model-Independent Expressions

Paralleling (154), the number of machine *ands* performed by wgFC in solving instance clp is given by

$$A(\text{gwFC } clp) = \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} w_{f_k^i} \sum_{t=1}^{m_{f_k^i}} \delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp) \tag{208}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} w_{f_k^i}\, \delta(\text{gFC } k\ i \geq 1\ \bar{X}_k\ clp) \tag{209}$$

These are in terms of the indicator functions $\delta(\text{gFC } k\ i\ t\ \bar{X}_k\ clp)$ and $\delta(\text{gFC } k\ i \geq 1\ \bar{X}_k\ clp)$ of (83) and (86). Although the latter functions are for algorithm gFC, their use is appropriate here for wgFC. The above expressions for $A(\text{gwFC } clp)$ say simply that, in forward checking a constraint $C_{\phi_k^i}$ at a node, so long as at the corresponding stage in gFC there would be $t \geq 1$ values still viable from the domain of the corresponding future variable f_k^i, then gwFC will perform $w_{f_k^i}$ *ands* in filtering that domain (rather than the t individual checks that would be performed by gFC). Paralleling the sequence (166) to (170) we have now

[24] Note that when all $|\phi_{kf}| = 1$ for a given k, then there is a one-to-one correspondence between orderings ϕ_k and permutations of F_k. This is why in [21], where only pure and simple binary instances were treated, it was sufficient to use permutations of F_k to characterise the constraint-check orderings of gFC. Similarly for gBT, it was sufficient to use permutations of the past variables P_k of table 2-1.

$$\bar{A}(\text{gwFC}) = \sum_{clp \in \Omega} A(\text{gwFC } clp)\, P(clp) \quad \text{by (11)} \tag{210}$$

$$= \sum_{clp \in \Omega} \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} w_{f_k^i}\, \delta(\text{gFC } k\ i \geq 1\ \bar{X}_k\ clp)\, P(clp) \quad \text{by (209)} \tag{211}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} w_{f_k^i} \sum_{clp \in \Omega} \delta(\text{gFC } k\ i \geq 1\ \bar{X}_k\ clp)\, P(clp) \tag{212}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \sum_{i=1}^{|\phi_k|} w_{f_k^i}\, P(\text{gFC } k\ i \geq 1\ \bar{X}_k) \quad \text{by (13)} \tag{213}$$

$$= \sum_{k=1}^{n-1} \sum_{\bar{X}_k \in D_{X_k}} \left[\prod_{j \in \Psi_k} P(j\ \bar{X}_k) \right] \left[\sum_{i=1}^{|\phi_k|} w_{f_k^i} \prod_{g \in F_k} P(\Phi_{kg}(<i)\ g \geq 1\ \bar{X}_k) \right] \tag{214}$$

by (157) and T_j independence

3.3.2. Small-Classes Under Model-2

For the case of a CLP small-class $\text{CLP}_\sigma(n\ \mathbf{m}\ c\ \mathbf{Z}\ \mathbf{S})$ under model-2, the expected number of machine *ands* in solving an instance is given by the expression below — obtained from the model-independent expression (214) in the same way that (173) was obtained from (170) for gFC.

$$\bar{A}_\sigma^2(\text{gwFC}) = \sum_{k=1}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} R_j \right] \left[\sum_{i=1}^{|\phi_k|} w_{f_k^i} \prod_{g \in F_k} P_\sigma^2(\Phi_{kg}(<i)\ g \geq 1) \right] \tag{215}$$

As mentioned above, the expected number of nodes generated by gwFC is the same as that for gFC, and hence for the present situation is as given in (172).

3.3.3. Big-Classes Under Model-1

For the case of a CLP big-class $\text{CLP}_\beta(n\ \mathbf{m}\ c\ \mathbf{Z})$ under model-1, the expected number of machine *ands* in solving an instance is given by the expression below — obtained from the model-independent expression (214) in the same way that (176) was obtained from (170) for gFC.

$$\bar{A}_\beta^1(\text{gwFC}) = \sum_{k=1}^{n-1} \left[\prod_{x \in X_k} m_x \right] \left[\prod_{j \in \Psi_k} p_j \right] \left[\sum_{i=1}^{|\phi_k|} w_{f_k^i} \prod_{g \in F_k} P_\beta^1(\Phi_{kg}(<i)\ g \geq 1) \right] \tag{216}$$

Again, since the expected number of nodes generated by gwFC is the same as that for gFC, for the present situation this quantity is as given in (175). Of course, the various expressions for expected number of nodes generated by gFC under model-0, appearing in section 3.2.4, also apply for gwFC in the corresponding situations. Moreover, the expected number of *ands* performed by gwFC in those model-0 situations is easily obtained by specialization of (216) in the same way as the various $\bar{C}_\sigma^2(\text{gFC})$ of section 3.2.4 were obtained from (176).

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] Bitner J. R., "Backtrack programming techniques," *Communications ACM,* vol. 18, pp. 651-656, 1975.

[3] Burstall R. M., "A program for solving word sum puzzles," *Computer Journal,* vol. 12, pp. 48-51, 1969.

[4] Feller W., *An Introduction to Probability Theory and Its Applications,* Wiley, New York, 1968.

[5] Freuder E. C., "Synthesizing constraint expressions," *Comm. ACM,* vol. 21, pp. 958-966, 1978.

[6] Gaschnig J., "A constraint satisfaction method for inference making," *Proc. 12-th Annual Allerton Conf. on Circuit System Theory,* pp. 866-874, U. Illinois, 1974.

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

[8] Golomb S. W. and Baumert L. D., "Backtrack programming," *J. ACM,* vol. 12, pp. 516-524, 1965.

[9] Haralick R. M., Davis L. S., and Rosenfeld A., "Reduction Operations for Constraint Satisfaction," *Information Sciences,* vol. 14, pp. 199-219, 1978.

[10] Haralick R. M. and Shapiro. L. G., "The consistent labeling problem: part I," *I.E.E.E. Trans. Pattern Analysis and Machine Intelligence,* vol. PAMI-1, no. 2, pp. 173-184, 1979.

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

[12] Haralick R. M. and Shapiro. L. G., "The consistent labeling problem: part II," *I.E.E.E. Trans. Pattern Analysis and Machine Intelligence,* vol. PAMI-2, no. 3, pp. 193-203, 1980.

[13] Johnson N. L. and Kotz S., *Urn Models and Their Applications: An Approach to Modern Discrete Probability Theory,* Wiley, New York, 1977.

[14] Knuth D. E., "Estimating the efficiency of Backtrack programs," *Mathematics of Computation,* vol. 29, no. 129, pp. 121-136, January, 1975.

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

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

[17] Montanari U., "Networks of constraints: Fundamental properties and applications to picture processing," *Information Sciences*, vol. 7, pp. 95-132, 1974.

[18] Nadel B. A., *The Consistent Labeling Problem and its Algorithms: Towards Exact-Complexities and Theory-Based Heuristics*, Computer Science Dept., Rutgers University, New Brunswick, N.J., 1986, Ph.D. dissertation, to appear.

[19] Nadel B. A., "The general Consistent Labeling (or Constraint Satisfaction) Problem," Report DCS-TR-170, Computer Science Dept., Rutgers University, New Brunswick, N.J., 1986, Also appears as Report CRL-TR-2-86, Dept. Electrical Engineering and Computer Science, U. of Michigan, Ann Arbor, MI, 1986.

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

[21] 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.

[22] 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.

[23] Papoulis A., *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, New York, 1965.

[24] Stallman R. M. and Sussman G. J., "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis," *Artificial Intelligence*, vol. 9, pp. 135-196, 1977.

[25] Waltz D., "Understanding line drawings of scenes with shadows," in *The Psychology of Computer Vision*, Winston P. H., Ed McGraw-Hill, New York, 1975.

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	I	B

A4 Page 6543210

MESH HALFTONE WEDGES

65

85

100

110

133

150

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