Consistency and Propagation with Multiset Constraints: A Formal Viewpoint

Toby Walsh[1]

Cork Constraint Computation Center, University College Cork, Ireland. tw@4c.ucc.ie

Abstract. We study from a formal perspective the consistency and propagation of constraints involving multiset variables. That is, variables whose values are multisets. These help us model problems more naturally and can, for example, prevent introducing unnecessary symmetry into a model. We identify a number of different representations for multiset variables and compare them. We then propose a definition of local consistency for constraints involving multiset, set and integer variables. This definition is a generalization of the notion of bounds consistency for integer variables. We show how this local consistency property can be enforced by means of some simple inference rules which tighten bounds on the variables. We also study a number of global constraints on set and multiset variables. Surprisingly, unlike finite domain variables, the decomposition of global constraints over set or multiset variables often does not hinder constraint propagation.

1 Introduction

Set variables have been incorporated into most of the major constraint solvers (see, for example, [1, 2]). It is therefore surprising that few constraint solvers permit multiset variables. The one exception is ILOG's Configurator. However, little is known from a theoretical perspective about such variables. The aim of this paper is to correct this imbalance, to study formal notions of consistency and propagation for multiset variables, and to discuss how they can be implemented. Many problems naturally involve multisets. Consider the template design problem [3] (prob002 in CSPLib) in which we assign designs to printing templates. As there are a fixed number of slots on each template, we can model this problem with a variable for each slot, whose value is the design in this slot. However, slots on a template are indistinguishable. This model therefore introduces an unnecessary symmetry, namely the permutations of the slots. A "better" model would remove this symmetry by having a variable for each template, whose value is the multiset of designs assigned to that template. It is a multiset, not a set, as the designs on a template can be repeated.

The paper is structured as follows. We start with the formal background (Section 2). We then compare different ways to represent multiset variables (Section

[1] The author is supported by the Science Foundation Ireland. He wishes to thank the members of the 4C lab, the APES research group, and Chris Jefferson.

3) and define a notion of local consistency for multiset variables (Section 4). We identify a number of primitive multiset constraints (Section 5) and show how to enforce this local consistency property on such constraints (Section 6). We also study a number of global multiset constraints (Section 7). We then give some experimental results comparing different representations of multiset variables (Section 8). Finally we describe related work (Section 9) and end with conclusions (Section 10).

2 Formal background

A constraint satisfaction problem consists of a set of variables, each with some domain of values, and a set of constraints specifying allowed values for subsets of variables. A solution is an assignment of values to the variables satisfying the constraints. To find such solutions, we can explore partial assignments enforcing a local consistency like generalized arc-consistency (GAC). A constraint is GAC iff, when a variable in the constraint is assigned a value, compatible values exist for all the other variables in the constraint. GAC reduces to arc-consistency (AC) for binary constraints. A constraint is bounds consistent iff, when a variable in the constraint is assigned its maximum or minimum value, there exist compatible values for all the other variables in the constraint.

We will also need vectors, sets and multisets. A vector is an ordered list of elements, written $\langle m_0, \ldots, m_n \rangle$. A set is an unordered list of elements without repetition, written $\{m_0, \ldots, m_n\}$. A multiset is an unordered list of elements in which repetition is allowed, written $\{\!\{m_0, \ldots, m_n\}\!\}$. We assume that the elements of vectors, sets and multisets are integers drawn from a finite domain. Basic operations on sets generalize to similar operations on multisets. We let $occ(m, X)$ be the number of occurrences of m in the multiset X. Multiset union, addition, intersection, difference, equality and inclusion are defined by the follow identities: $occ(m, X \cup Y) = max(occ(m, X), occ(m, Y))$, $occ(m, X \uplus Y) = occ(m, X) + occ(m, Y)$, $occ(m, X \cap Y) = min(occ(m, X), occ(m, Y))$, $occ(m, X - Y) = max(0, occ(m, X) - occ(m, Y))$, $X = Y$ iff $occ(m, X) = occ(m, Y)$ for all m, and $X \subseteq Y$ iff $occ(m, X) \leq occ(m, Y)$ for all m. Finally, we write $|X|$ for the cardinality of the set or multiset X.

3 Representing multisets

A naive method to represent a multiset variable is a finite domain variable whose values are all the possible multisets. However, this will be computationally intractable as the number of possible multisets is exponential.

3.1 Bounds representation

A better representation for multiset variables is a generalization of the upper and lower bounds used for set variables in [1, 4]. For each multiset variable, we maintain two multisets: a least upper and a greatest lower bound. The least upper bound is the smallest multiset containing all those values that can be in the multiset, whilst the greatest lower bound is the largest multiset containing all those values that must be in the multiset. We write $lub(X)$ and $glb(X)$ for the least upper and greatest lower bound respectively. This representation is compact but is unable to represent all forms of disjunction. Consider, for example, a multiset variable X with two possible multiset values: $\{\!\{0\}\!\}$ or $\{\!\{1\}\!\}$. To represent this, we would need $lub(X) = \{\!\{0, 1\}\!\}$ and $glb(X) = \{\!\{\}\!\}$. However, this representation also permits X to take the multiset values $\{\!\{\}\!\}$ and $\{\!\{0, 1\}\!\}$.

3.2 Occurrence representation

Set variables can be represented by their characteristic function (a vector of Boolean variables, each of which indicates whether a particular value is in the set or not). A straightforward generalization to multisets is the occurrence vector. Each multiset variable X can be represented by a vector $\langle X_0, \ldots, X_n \rangle$ of integer variables with $X_i = occ(i, X)$. This representation is also compact but again cannot represent all forms of disjunction. Consider again the example of a multiset variable X with two possible multiset values: $\{\!\{0\}\!\}$ or $\{\!\{1\}\!\}$. To represent this, we would need an occurrence vector with $X_0 = \{0, 1\}$ (that is, the value 0 can occur zero times or once) and $X_1 = \{0, 1\}$ (that is, the value 1 can occur zero times or once). Like the bounds representation, this also permits X to take the multiset values $\{\!\{\}\!\}$ and $\{\!\{0, 1\}\!\}$.

3.3 Fixed cardinality

Set or multiset variables of a fixed cardinality are common in a number of problems. For example, the template design problem can be modelled as finding a multiset of designs of fixed cardinality for each template. In such situations, we can represent each of the members of the set or multiset with a variable whose values are the possible set or multiset elements. This may appear to introduce symmetry into the problem (via permutations of these variables). This is not the case as we will post constraints on these variables which ignore their permutation. This representation is again compact but again carries the penalty of not being able to represent all forms of disjunction. Consider, for example, a multiset variable X of cardinality 3 with two possible multiset values: $\{\!\{0, 0, 0\}\!\}$ or $\{\!\{1, 1, 1\}\!\}$. To represent this, we would need three variables: $X_1 = \{0, 1\}$, $X_2 = \{0, 1\}$ and $X_3 = \{0, 1\}$. Each finite domain variable represents one of the possible elements of the multiset. However, this representation also permits X to take the multiset values: $\{\!\{0, 0, 1\}\!\}$, and $\{\!\{0, 1, 1\}\!\}$. If the set or multiset variables are not of a fixed cardinality but there are upper bounds on their maximum cardinality, we can use a similar representation but introduce an additional value which represents no value being assigned to a particular variable.

3.4 Nested sets and multisets

We may want to find a set of sets or multisets, or a multiset of sets or multisets. For example, in the template design problem we actually want to find a set of templates, each of which is a multiset of designs. To model such problems, we can introduce set or multiset variables, whose elements themselves are sets or multisets. How do our different representations cope with such variables? The bounds representation handles such cases easily. The least upper and greatest lower bounds are now (multi)sets of (multi)sets. The occurrence representation is more problematic as we have to index over a potential exponential number of sets or multisets. This will require exponential space in general. By comparison, the fixed cardinality representation handles such cases easily. We introduce a variable for each element of the set or multiset, and each of these variables is itself a set or multiset variable.

3.5 Expressivity

We can compare the expressivity of these different representations. We say that one representation is **as expressive as** another if it can represent the same multiset values, **more expressive** if it is as expressive and there are multiset values that it can represent that the other cannot, and **incomparable** if neither representation is as expressive as the other.

Theorem 1 *The occurrence representation is more expressive than the bounds representation. The fixed cardinality representation is incomparable to both the bounds and the occurrence representation.*

Proof: Clearly the occurrence representation is as expressive as the bounds. Consider a multiset variable X with two values: $\{\!\{\}\!\}$ or $\{\!\{0,0\}\!\}$. This can be represented exactly with the occurrence variable $X_0 = \{0,2\}$. By comparison, a bounds representation would need $lub(X) = \{\!\{0,0\}\!\}$ and $glb(X) = \{\!\{\}\!\}$, and this permits the additional value $\{\!\{0\}\!\}$.

Consider a multiset variable X of cardinality 2 with six values: $\{\!\{0,1\}\!\}$, $\{\!\{0,2\}\!\}$, $\{\!\{0,3\}\!\}$, $\{\!\{1,1\}\!\}$, $\{\!\{1,2\}\!\}$, or $\{\!\{1,3\}\!\}$. The fixed cardinality representation can represent this exactly with two finite domain variables $X_1 = \{0,1\}$ and $X_2 = \{1,2,3\}$. Both the bounds and the occurrence representations of this would also permit the additional value $\{\!\{2,3\}\!\}$. On the other hand, consider a multiset variable X of cardinality 2 with three values: $\{\!\{0,1\}\!\}$, $\{\!\{0,2\}\!\}$, or $\{\!\{1,2\}\!\}$. In the bounds representation, we need $lub(X) = \{\!\{0,1,2\}\!\}$ and $glb(X) = \{\!\{\}\!\}$. The only two element multisets between these bounds are exactly $\{\!\{0,1\}\!\}$, $\{\!\{0,2\}\!\}$, or $\{\!\{1,2\}\!\}$ as required. Similarly with an occurrence representation, we need $X_0 = X_1 = X_2 = \{0,1\}$. The only two element multisets between these bounds are again the required ones. A fixed cardinality representation cannot, on the other hand, represent this exactly. We would need two finite domain variables with, say, $X_1 = \{0,1\}$ and $X_2 = \{1,2\}$. These would permit X to take the additional value $\{\!\{1,1\}\!\}$. ♣

Note that if we restrict the occurrence representation to maintain just bounds on the number of occurrences of a value in the multiset then we obtain a representation that is as expressive as the original multiset bounds representation.

4 Local consistency

We now propose a new definition of local consistency that works with constraints involving multiset, set and/or integer variables. We want a definition of local consistency over multiset, set and integer variables since constraints often have a mixture of such variables. For example, channelling between the bounds and occurrence representation of multiset variables uses constraints of the form $X_m = occ(m, X)$, where X_m is an integer variable representing the number of occurrences of the value m, and X is a multiset variable. Cardinality and membership constraints can also involve both multiset, set and integer variables.

Given a constraint C over the variables $X_1, \ldots, X_n$, we write $sol(X_i)$ for the values for X_i which can be extended to the other variables. That is,

$$sol(X_i) = \{d_i \mid C(d_1, \ldots, d_n) \ \wedge \ \forall j \ . \ int(X_j) \rightarrow glb(X_j) \leq d_j \leq lub(X_j) \ \wedge \\ \forall j \ . \ (mset(X_j) \vee set(X_j)) \rightarrow glb(X_j) \subseteq d_j \subseteq lub(X_j)\}$$

Where $mset(X)$, $set(X)$ and $int(X)$ test for multiset, set or integer variables, and $glb(X_j)$ and $lub(X_j)$ are the bounds on X_j (defined below).

We say that a constraint $C(X_1, \ldots, X_n)$ is **BC** iff:

For each multiset or set variable, X_j in the constraint, $sol(X_j) \neq \{\}$ and:

$$lub(X_j) = \bigcup_{m \in sol(X_j)} m \qquad \text{and} \qquad glb(X_j) = \bigcap_{m \in sol(X_j)} m$$

And for each integer variable, X_i in the constraint, $sol(X_i) \neq \{\}$ and:

$$lub(X_i) = max(\{d \in sol(X_i)\}) \qquad \text{and} \qquad glb(X_i) = min(\{d \in sol(X_i)\})$$

This definition of local consistency might look rather expensive, being defined over the set of *all* solutions. However, this set merely identifies *support* for particular values in the set or multiset. When using BC to filter, we will identify values which occur in no solutions and so can be pruned. Thus, we will not be finding all solutions but merely identifying those values that occur in no solutions (i.e. lack support). The following theorem justifies why BC can be called "bounds consistency".

Theorem 2 *BC is equivalent to bounds consistency applied to the occurrence representation.*

Proof: Suppose that a constraint is BC. Consider any integer variable X in this constraint. Then, the value $lub(X)$ for X can be extended to some solution. That is, it has support. Similarly the value $glb(X)$ for X can be extended to some other solution. That is, it also has support. Hence X is bounds consistent. On the other hand, consider any multiset variable X in the constraint. We can construct an equivalent occurrence representation. Suppose $m_{\max} = occ(m, lub(X))$ and $m_{\min} = occ(m, glb(X))$. Then we let the variable X_m in the occurrence vector have a domain $[m_{\min}, m_{\max}]$. Consider $X_m = m_{\max}$. Then, from the definition of BC and the generalized multiset union operator, there must be a satisfying solution to the constraint in which $occ(m, X) = m_{\max}$. If there are several, we choose one non-deterministically. This solution is support for the bounds consistency of this integer variable in the occurrence representation. A similar argument holds for $X_m = m_{\min}$, and for any set variable. Hence, BC implies bounds consistency of the occurrence representation. The proof reverses directly. ♣

This theorem might appear to offer an easy and effective route to prune values from multiset variables: encode the problem into constraints on occurrence variables and use "off the shelf" bounds consistency algorithms. However, the occurrence representation greatly increases the number of variables in the problem. For example, suppose we have a constraint like $X \neq Y$ where X and Y are multiset variables. This maps into a large disjunctive constraint in the occurrence representation over $2d$ integer variables where d is the maximum possible cardinality of the two multisets. It is therefore worth developing specialized propagation algorithms that exploit the semantics of set or multiset constraints. Such algorithms can work on either a bounds or an occurrence representation. In the next two sections, we show how to define such algorithms by means of some simple inference rules. Note that a degenerate version of this last theorem is that BC on a constraint containing just integer variables is equivalent to bounds consistency on these variables. Some other properties also follow immediately from this result.

Theorem 3 *If a set of constraints are satisfiable, there are unique least upper and greatest lower bounds for each variable that makes the constraints BC.*

Proof: Immediate from the last result, and the fact that bounds consistency on integer variables returns an unique answer. ♣

5 Multiset constraints

What sort of constraints can be posted on multiset variables? We assume constraints on multisets and set variables are defined as follows. A constraint is of the form $X \subseteq Y$, $X \subset Y$, $X = Y$, $X \neq Y$, $|X| = N$, $occ(N, X) = m$ or $occ(m, X) = N$ where X and Y are set or multiset expressions, N is an integer variable, and m is an integer. A set of multiset expression is, in turn, either a ground set or multiset, a set or multiset variable, or an expression of the form $X \cup Y$, $X \uplus Y$, $X \cap Y$, or $X - Y$ where X and Y are again set or multiset expressions. To make constraint propagation easier, we decompose constraints into a flattened normal form in which constraints are at most ternary and only of the form: $X \subseteq Y$, $X = Y \cup Z$, $X = Y \uplus Z$, $X = Y \cap Z$, $X = Y - Z$, $X \neq Y$, $|X| = N$, $occ(N, X) = m$ or $occ(m, X) = N$ where X and Y are either set or multiset variables or ground sets or multisets, N is an integer variable, and m is an integer. This decomposition takes any nested set or multiset expression and replaces it by a new equality constraint. For example, $(X \cup Y) \subseteq Z$ is normalized to give

$XY = X \cup Y$ and $XY \subseteq Z$ where XY is a new multiset variable. A similar decomposition of set constraints was used in [4]. In general, such decomposition hinders constraint propagation.

Theorem 4 *BC on a set of constraints is strictly stronger than BC on the equivalent set of constraints decomposed into normal form.*

Proof: Clearly it is as strong. For strictness, we consider each type of multiset or set constraint in turn. For a set not-equals constraint, consider $X \cup (Y \cap Z) \neq (X \cup Y) \cap (X \cup Z)$ with $X = Y = Z = \{\} :: \{0\}$. BC determines that this constraint has no solution. But in the decomposition, with $YZ = Y \cap Z$, $XYZ = X \cup YZ$, $XY = X \cup Y$, $XZ = X \cup Z$, $XYXZ = XY \cap XZ$ and $XYZ \neq XYXZ$, the domains $X = Y = Z = YZ = XYZ = XY = XZ = XYXZ = \{\} :: \{0\}$ make the dec omposed constraints BC. Similar arguments hold for the other types of constraints. ♣

Under the simple restriction that there are no repeated occurrences of variables, decomposition does not hinder constraint propagation.

Theorem 5 *BC on a set of constraints, none of which contains a repeated occurrence of variables, is equivalent to BC on the equivalent set of constraints decomposed into a normal form.*

Proof: The proof uses induction on the number of auxiliary variables introduced and the structure of the multiset expressions which they replace, followed by extensive case analysis. Consider, for example, the multiset constraint $X - Y \subset Z$ and the decomposition: $XY = X - Y$, $XY \subset Z$. Suppose each of the decomposed constraints is BC but the original undecomposed constraint is not BC. There are six possible cases. In the first, $glb(X)$ is too small and we can add at least one value m to it. This is only possible if m is a member of $glb(Y)$ or of $glb(Z)$. In either case, the original pair of decomposed constraints could not be BC. The other five cases are similar. ♣

6 Enforcing local consistency

We now give some simple constraint propagation rules that enforce BC on multiset constraints in normal form. The equivalent inference rules for set constraints can be obtained by treating the operators in the rules as set and not multiset operations. Similarly, for mixed constraints involving both set and multiset variables, we need merely treat operators as appropriate set or multiset operations. Each rule tightens an upper and/or lower bound on a variable. The rules therefore terminate either with domains at a fixed point or by flagging failure. The rules can be applied in any order, though some orders may be quicker than others (especially when the constraints cannot be made BC). Similar rules for set variables are given in [4].

Multiset inclusion rules:

$$\frac{X \subseteq Y}{glb(X) \cup glb(Y) \subseteq Y}$$

$$\frac{X \subseteq Y}{X \subseteq lub(X) \cap lub(Y)}$$

Multiset equality rules:

$$\frac{X = Y \cup Z}{glb(X) \cup (glb(Y) \cup glb(Z)) \subseteq X \subseteq lub(X) \cap (lub(Y) \cup lub(Z))}$$

$$\frac{X = Y \cup Z}{Y \subseteq lub(Y) \cap lub(X)}$$

$$\frac{X = Y \cup Z}{Z \subseteq lub(Z) \cap lub(X)}$$

$$\frac{X = Y \uplus Z}{glb(X) \cup (glb(Y) \uplus glb(Z)) \subseteq X \subseteq lub(X) \cap (lub(Y) \uplus lub(Z))}$$

$$\frac{X = Y \uplus Z}{glb(Y) \cup (glb(X) - lub(Z)) \subseteq Y \subseteq lub(Y) \cap (lub(X) - glb(Z)}$$

$$\frac{X = Y \uplus Z}{glb(Z) \cup (glb(X) - lub(Y)) \subseteq Z \subseteq lub(Z) \cap (lub(X) - glb(Y)}$$

$$\frac{X = Y \cap Z}{glb(X) \cup (glb(Y) \cap glb(Z)) \subseteq X \subseteq lub(X) \cap (lub(Y) \cap lub(Z))}$$

$$\frac{X = Y \cap Z}{glb(Y) \cup glb(X) \subseteq Y}$$

$$\frac{X = Y \cap Z}{glb(Z) \cup glb(X) \subseteq Z}$$

$$\frac{X = Y - Z}{glb(X) \cup (glb(Y) - lub(Z)) \subseteq X \subseteq lub(X) \cap (lub(Y) - glb(Z))}$$

$$\frac{X = Y - Z}{glb(Y) \cup (glb(X) \uplus glb(Z)) \subseteq Y \subseteq lub(Y) \cap (lub(X) \uplus lub(Z))}$$

$$\frac{X = Y - Z}{glb(Z) \cup (glb(Y) - lub(X)) \subseteq Z \subseteq lub(Z) \cap (lub(Y) - glb(X))}$$

Multiset inequality rules:

$$\frac{X \neq Y, glb(Y) = lub(Y) = glb(X), |lub(X)| = |glb(X)| + 1}{X = lub(X)}$$

$$\frac{X \neq Y, glb(Y) = lub(Y) = lub(X), |lub(X)| = |glb(X)| + 1}{X = glb(X)}$$

$$\frac{X \neq Y, glb(X) = lub(X) = glb(Y), |lub(Y)| = |glb(Y)| + 1}{Y = lub(Y)}$$

$$\frac{X \neq Y, glb(X) = lub(X) = lub(Y), |lub(Y)| = |glb(Y)| + 1}{Y = glb(Y)}$$

Multiset cardinality rules:

$$\frac{|X| = N}{max(min(N), |glb(X)|)) \leq N \leq min(max(N), |lub(X)|)}$$

$$\frac{|X| = N, min(N) = max(N) = |glb(X)|}{X = glb(X)}$$

$$\frac{|X| = N, min(N) = max(N) = |lub(X)|}{X = lub(X)}$$

Multiset membership rules:

$$\frac{occ(N, X) = m, occ(min(N), glb(X)) > m}{N > min(N)}$$

$$\frac{occ(N, X) = m, occ(min(N), lub(X)) < m}{N > min(N)}$$

$$\frac{occ(N, X) = m, occ(max(N), glb(X)) > m}{N < max(N)}$$

$$\frac{occ(N, X) = m, occ(max(N), lub(X)) < m}{N < max(N)}$$

$$\frac{occ(N, X) = m, max(N) = min(N)}{glb(X) \cup \{\underbrace{N, \ldots, N}_{m \text{ times}}\} \subseteq X \subseteq lub(X) - \underbrace{\{N, \ldots\ldots, N\}}_{max(0, occ(N, lub(X)) - m)}}$$

$$\frac{occ(m, X) = N}{max(min(N), occ(m, glb(X))) \leq N \leq min(max(N), occ(m, lub(X)))}$$

$$\frac{occ(m, X) = N}{glb(X) \cup \{\underbrace{m, \ldots\ldots, m}_{min(N) \text{ times}}\} \subseteq X \subseteq lub(X) - \underbrace{\{m, \ldots\ldots, m\}}_{max(0, occ(m, lub(X)) - max(N))}}$$

Failure rules: Each of the inference rules given so far tightens the bounds for a variable. We fail whenever this rules out all possible values for the variable. The following additional inference rules also lead to failure:

$$\frac{X \subseteq Y, glb(X) \not\subseteq lub(Y)}{Fail}$$

$$\frac{X = Y \cup Z, glb(X) \not\subseteq lub(Y) \cup lub(Z)}{Fail}$$

$$\frac{X = Y \cup Z, glb(Y) \cup glb(Z) \not\subseteq lub(X)}{Fail}$$

$$\frac{X = Y \uplus Z, glb(X) \not\subseteq lub(Y) \uplus lub(Z)}{Fail}$$

$$\frac{X = Y \uplus Z, glb(Y) \uplus glb(Z) \not\subseteq lub(X)}{Fail}$$

$$\frac{X = Y \cap Z, glb(X) \not\subseteq lub(Y) \cap lub(Z)}{Fail}$$

$$\frac{X = Y \cap Z, glb(Y) \cap glb(Z) \not\subseteq lub(X)}{Fail}$$

$$\frac{X = Y - Z, glb(X) \not\subseteq lub(Y) - glb(Z)}{Fail}$$

$$\frac{X = Y - Z, glb(Y) - lub(Z) \not\subseteq lub(X)}{Fail}$$

$$\frac{X \neq Y, glb(X) = lub(X) = glb(Y) = lub(Y)}{Fail}$$

$$\frac{|X| = N, max(N) < |glb(X)|)}{Fail}$$

$$\frac{|X| = N, |lub(X)| < min(N)}{Fail}$$

$$\frac{occ(N, X) = m, \forall y \ . \ occ(y, lub(X)) < m \ \vee \ m < occ(y, glb(X))}{Fail}$$

$$\frac{occ(m, X) = N, max(N) < occ(m, glb(X))}{Fail}$$

$$\frac{occ(m, X) = N, occ(m, lub(X)) < min(N)}{Fail}$$

Properties It is easy to see that the application of these rules terminates either with domains that are at a fixed point or with failure. Indeed, these rules terminate either with the unique BC domains or, if the problem cannot be made BC, fail, in both cases independent of the order of application of the rules.

Theorem 6 *If a set of constraints in normal form can be made BC, these inference rules reach an unique fixed point in which domains are BC. If the constraints cannot be made BC, the inference rules terminate with failure. Both take at most* $O(enm^2)$ *time where* e *is the number of constraints,* n *is the number of variables and* m *is the maximum cardinality of the multiset variables.*

Proof: (Outline) Each inference rule tightens the upper and lower bounds of a variable or flags failure. The rules must therefore reach a fixed point or fail.

Suppose that we reach some fixed point applying these rules to a set of constraints in normal form. The proof uses case analysis on the type of constraint. Consider, for example, a constraint of the form $X = Y \cup Z$. We consider each of the multiset variables in turn and show that their domains are BC. For the variable X, as the inference rule tightening X's upper and lower bound is at a fixed point, it must be the case that $glb(Y) \cup glb(Z) \subseteq glb(X)$, $lub(X) \subseteq lub(Y) \cup lub(Z)$ and $glb(X) \subseteq lub(X)$. The assignment $X = glb(X)$, $Y = lub(Y) \cap glb(X)$ and $Z = lub(Z) \cap glb(X)$ will satisfy the constraint $X = Y \cup Z$ and the conditions that $glb(Y) \subseteq Y \subseteq lub(Y)$ and $glb(Z) \subseteq Z \subseteq lub(Z)$. Similarly, the assignment $X = lub(X)$, $Y = lub(Y) \cap lub(X)$ and $Z = lub(Z) \cap lub(X)$ will satisfy the constraint $X = Y \cup Z$ and the conditions that $glb(Y) \subseteq Y \subseteq lub(Y)$ and $glb(Z) \subseteq Z \subseteq lub(Z)$. Hence X's domain is BC. Similar arguments hold for the domains of Y and Z, as well as for the other types of constraints. Hence, if the rules terminate at a fixed point, the resulting domains are BC.

We now prove that, if the domains in the problem can be made BC, the rules terminate at this fixed point. Consider a problem that can be made BC, and its unique BC domains. The proof again uses extensive case analysis on the type of constraint. Consider, for instance, the constraint $X = Y \cup Z$ and the BC domains for X, Y and Z. To prove that the rules terminate at this fixed point, we assume that an inference rule can still narrow a domain or flag failure. There are five cases corresponding to the five different inference rules associated with this constraint. In the first, the inference rule narrows the least upper bound of Y by removing one or more values. Suppose one of these removed values is m. Let X_m, Y_m and Z_m be the number of occurrences of m in X, Y and Z respectively. As m is pruned by this inference rule, $max(Y_m) > max(X_m)$. The original multiset variables are not therefore BC (which is a contradiction). Hence, there can be no value m removed and this inference rule is at a fixed point if the domains are BC. Similar arguments hold for the other 4 inference rules

These rules therefore terminate at a fixed point iff the resulting domains are BC. As the rules must terminate either at a fixed point or by flagging failure, it follows that the rules flag failure iff the problem cannot be made BC. As each rule tightens the bounds on a multiset, set or finite domain occurrence variable, the worst case is when the rules tighten each bound by just one element at a time. We may therefore have to apply $O(nm)$ rules. To find which rule applies, we may have to go through each of the e constraints in turn. Associated with each type of constraint, a fixed number of rules can be tried. The cost of applying the inference rules is thus at most $O(enm)$ multiplied by the cost of applying a single inference rule. This last cost is dominated by the $O(m)$ cost to test (dis)equality or inclusion, and the $O(m)$ cost to perform one of the basic operations like union or difference. Hence, the total cost is $O(enm^2)$ in the worst case. ♣

7 Global constraints

An important aspect of constraint programming is global (or non-binary) constraints [5, 6]. Such constraints capture common patterns and often come with efficient and effective propagation algorithms. An important question about such constraints is whether decomposition hurts. Consider a global constraint on finite domain variables like the all-different constraint [5]. This can be decomposed into binary not-equals constraints, but this decomposition hinders constraint propagation. For instance, GAC on an all-different constraint is strictly stronger than arc-consistency (AC) on the decomposed binary not-equals constraints [7]. We therefore have to develop a specialized propagation algorithm to achieve GAC on an all-different constraint. Surprisingly, the decomposition of global constraints involving set or multiset variableos often does *not* hinder constraint propagation. This is good news. We can provide global constraints on set and multiset variables to help users compactly specify models. However, we do not need to develop complex algorithms for reasoning about them as is the case with finite domain variables. We can simply decompose such global constraints into primitive constraints and use the inference rules given in the last section. In the rest of this section, we give results to show that decomposition on the occurrence representation often does not hinder GAC, and that decomposition on the occurrence or bounds representation does not hinder BC.

Disjoint constraint The constraint $disjoint([X_1, \ldots, X_n])$ ensures that the multiset variables are pairwise disjoint. This global constraint can be decomposed into the binary constraints: $X_i \cap X_j = \{\!\{\}\!\}$ for all $i \neq j$. Such decomposition does not hinder constraint propagation.

Theorem 7 *GAC (resp. BC) on a disjoint constraint is equivalent to AC (resp. BC) on the binary decomposition.*

Proof: Clearly GAC (resp. BC) on a disjoint constraint is as strong as AC (resp. BC) on the decomposition. To show the reverse, suppose that the binary decomposition is AC (resp. BC). If the disjoint constraint is not GAC or BC then there must be at least two multiset variables, X_i and X_j with a value m in common. That is, $X_{im} \geq 1$ and $X_{jm} \geq 1$. However, in such a situation, the decomposed constraint $\min(X_{im}, X_{jm}) = 0$ would neither be AC nor BC. ♣

Partition constraint The constraint $partition([X_1, \ldots, X_n], X)$ ensures that the multiset variables, X_i are pairwise disjoint and union together to give X. By introducing new auxiliary variables, it can be decomposed into binary and union constraints of the form: $X_i \cap X_j = \{\!\{\}\!\}$ for all $i \neq j$, and $X_1 \cup \ldots \cup X_n = X$. Decomposition again causes no loss in pruning.

Theorem 8 *GAC (resp. BC) on a partition constraint is equivalent to GAC (resp. BC) on the decomposition.*

Proof: Clearly GAC (resp. BC) on a partition constraint is as strong as GAC (resp. BC) on the decomposition. To show the reverse, by Theorem 7, we need focus just on the union constraints. Suppose that the decomposition is GAC (resp. BC). If the partition constraint is not GAC or BC then there must be one value m that does not occur frequently enough in the upper bounds of the multiset variables. But, in this case, the decomposed constraint (which is equivalent to $\sum_{i=1}^{n} X_{im} = X_m$) would neither be GAC nor BC. ♣

This result continues to hold even if the union constraint is decomposed into the set of ternary union constraints by introducing new auxiliary variables: $X_1 \cup X_2 = X_{12}, X_{12} \cup X_3 = X_{13}, \ldots, X_{1n-1} \cup X_n = X$. We can also consider the non-empty partition constraint which also ensures that each multiset variable is not empty. Decomposition now hinders constraint propagation.

Theorem 9 *GAC (resp. BC) on a non-empty partition constraint is strictly stronger than GAC (resp. BC) on the decomposition.*

Proof. Clearly it is as strong. For strictness, consider 3 multiset variables with $glb(X_1) = glb(X_2) = glb(X_3) = \{\!\{\}\!\}$, $lub(X_1) = lub(X_2) = \{\!\{1,2\}\!\}$ and $lub(X_3) = \{\!\{1,2,3\}\!\}$. The decomposition is both GAC and BC. However, enforcing GAC or BC on the non-empty partition constraint gives $glb(X_3) = lub(X_3) = \{\!\{3\}\!\}$. ♣

Distinct constraint Consider the constraint $distinct([X_1, \ldots, X_n])$ which ensures that all the multisets are distinct from each other. This decomposes into pairwise not equals constraints: $X_i \neq X_j$ for all $i \neq j$. Decomposition in this case hinders constraint propagation.

Theorem 10 *GAC (resp. BC) on a distinct constraint is strictly stronger than AC (resp. BC) on the decomposition.*

Proof: Clearly it is as strong. For strictness, consider a distinct constraint on 3 multiset variables with $glb(X_1) = glb(X_2) = \{\!\{\}\!\}$, $lub(X_1) = lub(X_2) = \{\!\{0\}\!\}$, $glb(X_3) = \{\!\{0\}\!\}$, and $lub(X_3) = \{\!\{0,0\}\!\}$. The decomposition is both AC and BC. But enforcing GAC or BC on the distinct constraint gives $glb(X_3) = lub(X_3) = \{\!\{0,0\}\!\}$. ♣

8 Experimental results

Our preliminary experiments show that the choice of representation for multiset variabls can make a large difference even on relatively easy problems. Table 1 shows results for the template design problem (prob002 in CSPLib). The model

Problem	Number of templates	Objective value	Goal	Occurrence rep fails	Occurrence rep runtime/sec	Fixed card rep fails	Fixed card rep runtime/sec
cat food	1	550	find	8	0.00	371	0.03
			prove	0	0.00	389	0.03
	2	418	find	1173	0.12	3397750	502.40
			prove	5708	0.43	*	*
	3	409	find	48721	5.63	*	*
			prove	*	*	*	*
herbs	1	115	find	142	0.01	*	*
			prove	31	0.00	*	*
	2	96	find	54	0.01	*	*
			prove	132788714	10386.20	*	*

Table 1. Solutions to the template design problem modelled using multiset variables. The objective is the production run length. Multiset variables are represented with either the occurrence or fixed cardinality representations. The objective is the production run length. All solutions are optimal for the given number of templates. Runtimes are for OPL Studio 3.5.1 on a Pentium III 1.2 GHz with 512 MB of RAM running Windows XP. Entries marked "*" are not solved within 3 hours.

is relatively easy to solve when the multiset variables are represented via the occurrence representation. However, despite the fact that the multiset variables in this problem represent the contents of each template and these are of fixed size, the model is difficult to solve when the multiset variables are represented via the fixed cardinality representation.

When constraint programming with multiset variables, a number of issues arise which we are currently exploring. For example, which of the different representations for multiset variables is best? Is it simply enough to find the representation in which the constraints are "easy" to express? When do we go for multiple representations with channelling between them? We also need to develop new variable and value ordering heuristics for multiset variables. The fail first principle for variable ordering translates into: *branch on the multiset variable X in which $|lub(X) - glb(X)|$ is smallest.* However, when we have both set, multiset and integer variables, we need heuristics to choose between them. We must also decide what sort of branching decision to make. For example, do we branch on the number of occurrences or try to split the difference between lower and upper bounds?

9 Related work

ILOG's Configurator has an `IlcBagPort` variable to model the multiset of components connected to a particular component. This uses an occurrence representation for the multisets, as well as integer variables for the cardinality of the multiset and for the number of values in the multiset. The only multiset constraints that appear to be supported are equality, inclusion and their negations. The domain of a set or multiset variable can include the `IlcWildCard` value, representing any possible extension of the set or multiset. It would be interesting to study the theoretical properties of this extension.

Set variables have been integrated into the ECLIPSE constraint logic programming language using a bounds representation [4]. Our definition of bounds consistency generalizes the local consistency property given in [4] for set variables. For example, a subset constraint $S_1 \subseteq S_2$ is *locally consistent* iff $glb(S_1) \subseteq glb(S_2)$ and $lub(S_1) \subseteq lub(S_2)$, whilst a cardinality constraint $l \leq |S_1| \leq u$ is *locally consistent* iff $l \leq |glb(S_1)|$ and $|lub(S)| \leq u$. Another advantage of our definition is that it works with any type of constraint, and is not restricted to those types of constraint considered in [4].

Theorem 11 *A subset constraint $S_1 \subseteq S_2$ is locally consistent iff it is BC. A cardinality constraint $l \leq |S_1| \leq u$ is locally consistent iff it is BC.*

Proof: Suppose $S_1 \subseteq S_2$ is BC and $lub(S_1) \not\subseteq lub(S_2)$. Then there must be $a \in lub(S_1)$ with $a \notin lub(S_2)$. Hence there exists $S \in sol(S_1)$ with $a \in S$, but for all $S \in sol(S_2)$ there is no $a \in S$. The value S for S_1 cannot then have support in the constraint $S_1 \subseteq S_2$. Hence $lub(S_1) \subseteq lub(S_2)$. By an analogous argument, $glb(S_1) \subseteq glb(S_2)$. The proof reverses easily.

Suppose $l \leq |S_1| \leq u$ is BC. Then for $S \in sol(S_1)$, $l \leq |S|$. Hence $l \leq |glb(S)|$. By an analogous argument, $|lub(S_1)| \leq u$. The proof reverses easily. ♣

The constraint logic programming language {Log} provides sets and multisets as basic types [8]. Sets and multisets are axiomatically defined and solved using a mixture of unification and rewriting. However, computational efficiency is not a major goal as {Log} is more concerned with expressivity, e.g. being able to represent and reason about partially specified and nested sets. Our goals, however, are more computational. We wish to augment constraint solving with efficient constraint propagation techniques for dealing with multiset variables. Some other systems like CLPS [9] also build sets into their unification procedure but are again more concerned with expressivity than efficiency.

10 Future work and conclusions

We have formally studied the role of multiset variables in constraint programming. We identified a number of different representations for multiset variables and compared them. We proposed a definition of local consistency for constraints involving multiset, set or integer variables. This definition is a generalization of the notion of bounds consistency for integer variables. We showed how this local consistency property can be enforced by means of some simple inference rules which tighten bounds on the variables. We also studied a number of global constraints on set and multiset variables. Surprisingly, unlike finite domain variables, the decomposition of global constraints over set or multiset variables often does not hinder constraint propagation.

References

1. Gervet, C. Conjunto: constraint logic programming with finite set domains. In Bruynooghe, M., ed.: Proc. of the 1994 Int. Symp. on Logic Programming, MIT Press (1994) 339–358
2. Müller, T., Müller, M. Finite set constraints in Oz. In Bry, F., Freitag, B., Seipel, D., eds.: 13th Logic Programming Workshop, TU München (1997) 104–115
3. Proll, L., Smith, B. Integer linear programming and constraint programming approaches to a template design problem. INFORMS Journal on Computing, **10** (1998) 265–275
4. Gervet, C. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. Constraints **1** (1997) 191–244
5. Régin, J. A filtering algorithm for constraints of difference in CSPs. In Proc. of the 12th National Conference on AI, American Association for AI (1994) 362–367
6. Beldiceanu, N. Global constraints as graph properties on a structured network of elementary constraints of the same type. In Proc. of 6th Int. Conf. on Principles and Practice of Constraint Programming (CP2000), Springer (2000) 52–66
7. Gent, I., Stergiou, K., Walsh, T. Decomposable constraints. Artificial Intelligence **123** (2000) 133–156
8. Dovier, A., Piazza, C., Pontelli, E., Rossi, G. Set and constraint logic programming. ACM Trans. on Programming Languages and Systems **22** (2000) 861–931
9. Legeard, B., Legros, E. Short overview of the CLPS system. In Proc. of 3rd Int. Symp. on Programming Language Implementation and Logic Programming, Springer-Verlag (1991) 431–433 LNCS 528.