

Representations of Sets and Multisets in Constraint Programming

Christopher Jefferson and Alan M. Frisch

AI Group, Department of Computer Science
University of York, UK
[caj,frisch]@cs.york.ac.uk

Abstract. Constraint programming is a powerful and general purpose tool, but its use is limited, as the process of refining a specification of a problem into an efficient constraint program (known as modelling) is more of an art than a science at present and must be learned by years of experience. This paper theoretically analyses one frequently occurring pattern in modelling, how to choose between different representations of high-level structures, in particular sets and multisets. It differs from previous work by providing methods of comparing very different representations, and by abstracting away from a particular implementation of a representation. It demonstrates useful theoretical dominance results between representations in both a problem dependant and independent context.¹

1 Introduction

Constraint programming has proved to be an efficient method of solving combinatorial problems, and there are now a number of powerful and efficient constraint solvers available. Unfortunately, constraint programs which implement the same problem in different ways can have hugely differing runtimes and search size.

The first, and arguably most important part of the modelling process is deciding how to represent each variable in a problem specification, either by creating a CSP where some variables of the original specification are represented as multiple CSP variables or choosing how the constraint solver will internally represent each variable. This decision is affected by a number of issues, including the constraints, the domain size of the variables and if the solver being used offers special support for some choices of representation.

Many previous papers have looked at the most efficient methods of implementing one or more representations of high-level structures, including functions [6], permutations [7] and sets and multisets [1]. These papers compared the search size and runtime required for the implementation of constraints on particular representations, making use of the specialised constraints available in specific solvers, and how logically equivalent constraints on the same variables achieve varying levels of propagation. The purpose of this paper is to avoid

¹ Thanks to both Ian Miguel and Bernadette Martinez-Hernandez for helpful discussions on this work

studying a particular implementation of a representation, and instead given the CSP variables which will represent a particular variable in an abstract specification, investigate how well the best implementation which used those variables would perform in terms of search size. The closest previous work to this paper, by Walsh [8], compared two representations of multisets, and this paper extends some of the results in that paper. This paper begins by giving a more general definition of representation than the one considered in [8], and then goes on to prove a number of useful results, including when representations dominate each other, where one representation always produces smaller searches than another, and also how representations can be shown to produce as small a search as the best possible representation (which is the representation which can represent all possible sub-domains of a variable). The aim is that this theory will be useful to both modellers and automated modelling tools (such as CONJURE [3]), to provide guidance on the best representation to use for a particular problem. The examples used throughout this paper involve sets and multisets, as these are very frequently occurring constructs in constraint programming. The theory however applies to representing any type of variable, and has been usefully applied to a large range of constructs including graphs, relations and sparse matrices.

1.1 Background

An instance of a finite domain CSP consists of a triple $\langle \mathbf{V}, Dom, C \rangle$, where \mathbf{V} is a finite set of variables, Dom is a function which maps each variable $v \in \mathbf{V}$ to the finite set of assignments of that variable and C is a finite set of constraints. Each constraint $c \in C$ specifies a set of assignments to some subset of \mathbf{V} , denoted by $vars(C)$, which the constraint allows.

Given a CSP variable X with domain D , a sub-domain of X is defined as a subset of D . For a vector of CSP variables \mathbf{V} , a sub-domain of \mathbf{V} is defined as a vector \mathbf{VD} , where $VD[i]$ is a sub-domain of $V[i]$. Given a CSP instance $P = \langle \mathbf{V}, D, C \rangle$, a sub-domain of P is a sub-domain of \mathbf{V} . Given a sub-domain of either a variable, vector of variables or a CSP, the assignments allowed by a sub-domain (denoted $assign(S)$) are defined only on the variables in the sub-domain, and consists of all possible assignments to the variables allowed by the given sub-domains. One sub-domain is (strictly) contained in another if the set of assignments allowed by the first is (strictly) contained in the assignments allowed by the second. A solution of a CSP $\langle \mathbf{V}, Dom, C \rangle$ is an assignment to \mathbf{V} which satisfies all the constraints in C .

Example 1. A finite domain CSP is given by $\langle \{X, Y\}, \{X \in \{1, 2, 3\} Y \in \{1, 2, 3\}\}, \{X + Y > 1, X < 3\} \rangle$. The domain of X is $\{1, 2, 3\}$. One sub-domain of the CSP is $X \in \{1, 2\}, Y \in \{2, 3\}$, which allows the assignments $\langle X, Y \rangle \in \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 2 \rangle, \langle 2, 3 \rangle\}$. Of these assignments, $\langle 1, 2 \rangle$ is the only solution, as it is the only one which satisfies both constraints.

The method for solving CSPs considered in the major of this paper will be branch-and-propagate search. As can be implied from the title, there are two major operations which must be performed while using this kind of search, branching and propagating. Both act on sub-domains of the CSP.

Definition 1. A propagation algorithm is defined by a function F which maps between sub-domains of a CSP with satisfies the following two conditions.

- $f(S) = T \rightarrow \text{assign}(T) \subseteq \text{assign}(S)$ and any solutions in $\text{assign}(S)$ are also in $\text{assign}(T)$
- $\text{assign}(S) \subseteq \text{assign}(T) \rightarrow \text{assign}(f(S)) \subseteq \text{assign}(f(T))$

One common family of propagation algorithms are constraint-based propagation algorithms, which given a sub-domain of a CSP P and a constraint $c \in P$, only remove values which do not satisfy c , and therefore clearly can't be part of any assignment which satisfies P . *MAC propagation* [2] removes all such values, and when no such values can be removed the sub-domain is defined to satisfy GAC with respect to c .

The second operation used during search is branching, which simply takes a sub-domain of a CSP P and generates n new CSPs P_1, \dots, P_n where P_i is identical to P except with an added constraint C_i , such that all assignments in the sub-domain are allowed by at least one of the C_i . Commonly the C_i are unary constraints. If any constraints can be used for branching it is possible for search to not be finite. The simplest way to prevent this is to ensure either at each propagation at least one value is removed from the sub-domain of some variable in each branch, or alternatively one finite CSPs, to ensure that no constraint is imposed more than once.

Definition 2. N-way branch-and-propagate search of a CSP P begins from the complete sub-domain of P and is defined recursively. Given a sub-domain of a CSP, branch-and-propagate search first performs propagation algorithms on the sub-domain. If the sub-domain of each variable consists of exactly one value, then this is an assignment node. If this assignment satisfies all the constraints then a solution has been found, else this is a failure node. Failure nodes also arise if the sub-domain of any variable becomes empty. If neither of these conditions hold then the current sub-domain is branched, generating a set of new CSPs to solve.

2 Representations

Given a specification of a problem, it is frequently not possible to map it directly into a CSP as it contains variables of types which are not supported by the constraint solver being used.

Example 2. The Social Golfer Problem involves trying to schedule w rounds of golf, where in each round the $g \times s$ players are arranged into g groups of size s such that no pair of players appear in a group together more than once.

The most obvious formulation of Example 2 would consist of a single variable to represent the schedule. This leads to variables with huge domain, for example 16 golfers split into 4 games on each of 6 weeks would lead to a schedule variable with over $6 * 10^{45}$ possible assignments. For this, and a number of other reasons, it often not efficient or even feasible to use an initial choice of representation. There are a number of ways of reducing memory usage to feasible amount, each

of which has its own trade-offs. Studying these trade-offs is the major purpose of this paper.

Example 3. Consider a CSP variable X with domain $D = \{1, 2, \dots, 10\}$. The “bounds representation” of X allows only the sub-domains of X (and therefore subsets of D) represented by the set $\{\langle l, u \rangle \mid 1 \leq l \leq u \leq n\} \cup \{\emptyset\}$, where $\langle l, u \rangle$ represents the sub-domain $\{x \mid l \leq x \leq u\}$ and \emptyset represents the empty sub-domain.

Propagating the constraint $X > 5$ from the complete sub-domain of X as much as possible leads to the sub-domain $\{6, 7, 8, 9, 10\}$, which the “bounds representation” represents exactly with $\langle 6, 10 \rangle$. Consider now the constraint “ X is even”. The assignments to X allowed by this constraint are $\{2, 4, 6, 8, 10\}$. Any element of the “bounds representation” which allows all these assignments (for example $\langle 2, 10 \rangle$) would also allow the set of assignments $\{3, 5, 7, 9\}$.

Example 4. Consider a CSP variable X with the domain “all subsets of $\{0, 1, 2, 3\}$ of size 2”. This is represented under the “explicit representation” by 2 variables, V_1 and V_2 , each of domain $\{0, 1, 2, 3\}$, and the constraint $V_1 \neq V_2$. Assignments to V_1 and V_2 represent assignments to X by the mapping $X = \{V_1, V_2\}$.

Consider the problem “find all assignments to X such that $1 \notin X$ ”. This would be represented with the explicit representation by the CSP “find all assignments to V_1 and V_2 such that $V_1 \neq 1, V_2 \neq 1$ and $V_1 \neq V_2$ ”. At the first node of search, MAC would remove 1 from the sub-domain of V_1 and V_2 , and the remaining assignments to V_1 and V_2 which satisfy $V_1 \neq V_2$ all represent assignments to X which satisfy $1 \notin X$.

Consider now the problem “find all assignments to X such that the sum of the members of X is 3”. This would be represented by the CSP “find all assignments to V_1 and V_2 such that $V_1 \neq V_2$ and $V_1 + V_2 = 3$ ”. At the first node of search, no values can be removed from the domains of either V_1 or V_2 , as they are all part of an assignment to the variables which satisfies all the constraints. There are however many assignments which represent assignments to X whose sum is not 3, for example $V_1 = 2, V_2 = 3$.

Example 5. Given a natural number n , consider the CSP containing a single set variable A with domain $MS(\{1, \dots, 2n\}, 1, [0 \dots 2n])$ and the two constraints $|A| = n$ and $|A| = n + 1$. If a solver can achieve GAC propagation for these two constraints, propagating both constraints will empty the sub-domain of A without search. Now consider A being represented by a Boolean vector \mathbf{V} of length $2n$, where $V[i]$ is true is $i \in A$. On any sub-domain of V where less than $n - 1$ variables are instantiated, neither the constraint $sum(\mathbf{V}) = n$ or $sum(\mathbf{V}) = n + 1$ would remove any values from the domains of any variables if propagated, so the size of the search tree will be at least 2^{n-2} nodes.

Examples 3 and 4 give two examples of representing a variable in a way which limits the number of possible sub-domains which are allowed. These aim to show the problems involved with choosing how to represent a variable in a CSP. In both examples, the first constraint considered is unaffected by the choice of representation, as it is possible to represent the sub-domain of the original

variable which contains only those assignments which satisfy the constraint. In both examples, the second constraint considered suffers by the choice of representation, as the smallest sub-domain which contains all assignments which satisfy the constraint also contains a number of assignments which do not. As the sub-domain of the CSP is the only method by which propagation algorithms can pass information to each other, this can impair the performance of search, and lead to greatly increased search size. This is shown in Example 5, where a poor choice of representation of a set of size n causes search to increase in size from a single node to a search tree of size exponential in n . This paper aims to formalise the idea of a representation and investigate how the states which can be represented limit the effectiveness of a particular representation.

Example 3 shows an example of where the representation can be considered as integral to the solver. Example 4 on the other hand, generates a new CSP whose solutions can be mapped to the solutions of the original. These should not be considered as disjoint families of representations, as the second can be considered as a subset of the first, and many solvers implement nested types internally as directly as a vector of CSP variables. For example, Conjunto[4] represents sets using a representation known as the occurrence representation (Definition 8). Both of these kinds of representations will be considered, and much of the theory of representations which will be presented applies equally to all representations, although some results can be extended or simplified by considering only representations which map one CSP to another.

Definition 3. *A representation of a CSP domain D is defined as a pair $\langle R, f \rangle$, where R is a partially ordered set and $f : R \rightarrow \mathbb{P}(D)$. R is the set of states which the representation can take, and for each $r \in R$, $f(r)$ is the sub-domain of X this state represents. A state $r_1 \in R$ is defined to be reachable from a state $r_2 \in R$ if $r_2 < r_1$. To be a valid representation, $\langle R, f \rangle$ must satisfy the following conditions:*

1. $r_1 \leq r_2 \rightarrow f(r_1) \subseteq f(r_2)$
If r_1 is reachable from r_2 , then r_1 must represent a subset of the assignments allowed by r_2 .
2. $\exists! r_D \in R. (f(r_D) = D \wedge \forall r \in R. r \neq r_D \rightarrow r < r_D)$
There must be some initial representational state from which all other states can be reached, to begin search.
3. $\forall r \in R. \exists r_\emptyset \in R. (f(r_\emptyset) = \emptyset \wedge r_\emptyset \leq r)$
From all representational states it must be possible to reach the state which represents nothing, else it would not be possible to fail.
4. $\forall r \in R, \forall d \in f(r). \exists r_d \in R. f(r_d) = \{d\} \wedge r_d \leq r$.
If a representational state represents an assignment d , it must be possible to reach a state which represents just the assignment d , else it would not be possible to reach all assignments a representation state represents.

Definition 4. *A simple representation of a CSP domain D is a representation $\langle R, f \rangle$ where the extra condition $\forall r_1, r_2 \in R. f(r_1) \subseteq f(r_2) \rightarrow r_1 \leq r_2$ holds. Combining this with Condition 1 of the definition of representation gives $\forall r_1, r_2 \in R. f(r_1) \subseteq f(r_2) \leftrightarrow r_1 \leq r_2$.*

The definition of *n-way branch-and-propagate search* in Section 1.1 did not refer to representations. The addition of representations limits the search, so that only certain sub-domains are allowed during search, and also which sub-domains may be reached from a given sub-domains is limited by the representation. The most important change to the definition of propagation algorithms is that instead of the result of applying the propagation algorithm being a subset of the assignments, instead the state of search achieved must be less in the ordering on the representation than the state which before. A proof ensuring that the limitations places on representations in Definition 3 are sufficient to ensure search is still correct is outlined in Theorem 1 due to lack of space. We are now in a position to justify the definition of representation given in Definition 3, and also prove that this definition is sufficient for search to be correct.

Theorem 1. *The four conditions in Definition 3 are necessary and sufficient for a representation to be used to represent sub-domains in an n-way branch and propagate search and any branching strategy still leads to all solutions, finite search, and all nodes being either solutions nodes, failure nodes, or branched on.*

Proof. If all these conditions are satisfied, then by condition 2, the first node can allow all assignments to the CSP. Condition 1 ensures search will be finite, condition 3 ensures from any node failure can be reached and condition 4 ensures from any node it possible (although not necessary) to choose a variable whose sub-domain allows more than one assignment and generate a new node for every allowed assignment. \square

2.1 Variable representations

As discussed by example at the beginning of Section 2, an important family of representations are those which represent a CSP variable by replacing it with a vector of CSP variables and mapping constraints involving the original variable to new constraints involving the new variables. These are known as *variable representations* (Definition 5). One important feature of variable representations is that as the result of applying them is another CSP and the resulting variables can be replaced with representations themselves.

Definition 5. *A variable representation of a variable X with domain D is a pair $\langle \mathbf{V}, f \rangle$ where \mathbf{V} is a vector of CSP variables and f is a partial surjective function from assignments of \mathbf{V} to D . An assignment to \mathbf{V} which is in the domain of f is said to represent its image under f . The representation constraint is defined as the constraint “ \mathbf{V} is in the domain of f ”. For a sub-domain \mathbf{V}' of \mathbf{V} , $f(\mathbf{V}')$ is defined as the sub-domain of X generated by applying f to all assignments in \mathbf{V}' .*

The induced representation of a variable representation $\langle \mathbf{V}, f \rangle$ is defined to be the representation $\langle R, g \rangle$ where R is defined as all sub-domains of \mathbf{V} , R is ordered by $r_1 \leq r_2 \leftrightarrow \forall v \in V. r_1[v] \subseteq r_2[v]$ and $g(r) = \{f(x) | x \text{ is allowed by } r\}$.

Lemma 1. *The representation $\langle R, g \rangle$ induced from a variable representation $\langle \mathbf{V}, f \rangle$ is simple if and only if f is injective.*

Proof. If the function f is not injective, then there must exist two distinct assignments v and w of \mathbf{V} such that $f(v) = f(w)$. The sub-domains containing only these assignments will clearly be incomparable but represent the same set of assignments. Consider now the case where f is injective. If $v \leq w$ then w allows all assignments v does, and therefore $f(v) \subset f(w)$. Similarly if $f(v) \subseteq f(w)$ then as f is injective $v \leq w$. \square

Definition 6. Given a constraint C and a variable representation $\langle \mathbf{V}, f \rangle$ of a variable $X \in \text{vars}(C)$, then X is defined to be replaced in C by $\langle \mathbf{V}, f \rangle$ as follows:

Given $\text{vars}(C) = \{X, Y_1, \dots, Y_n\}$, then C can be expressed as the subset S of $\langle D(X) \times D(Y_1) \times \dots \times D(Y_n) \rangle$ which contains the assignments $\text{vars}(C)$ which are allowed by C . C is the replaced by a new constraint over the V_i and Y_i which allows the assignments $\{\langle v_1, \dots, v_m, y_1, \dots, y_n \rangle \mid \langle f(\langle v_1, \dots, v_m \rangle), y_1, \dots, y_n \rangle \in S\}$. \square

Example 6. Consider a set variable X of size 2 drawn from $\{0, 1, 2, 3\}$. This is represented under the explicit representation by $\langle \mathbf{V}, f \rangle$, where $\mathbf{V} = \langle V_1, V_2 \rangle$ and the V_i have domain $\{0, 1, 2, 3\}$, f is defined as mapping each pair of values $\langle V_1, V_2 \rangle$ to the set $\{V_1, V_2\}$ where X and Y are distinct. Consider the constraint $\sum_{i \in X} = 3$, expressed as a set of tuples with $\{\langle \{0, 3\} \rangle, \langle \{1, 2\} \rangle\}$. When this constraint is replaced by the representation $\langle \mathbf{V}, f \rangle$, then the original constraint is replaced by the constraint $\langle V[1], V[2] \rangle \in \{\langle 0, 3 \rangle, \langle 3, 0 \rangle, \langle 1, 2 \rangle, \langle 2, 1 \rangle\}$.

Theorem 2. a) Given a CSP P , a variable X from P and a variable representation $\langle \mathbf{V}, f \rangle$ of X , a new CSP P' can be generated by adding the variables \mathbf{V} and the constraint $f(\mathbf{V}) = X$ to P . All solutions of P' can be generated by taking a solution of P and then adding an assignment to \mathbf{V} which satisfies the constraint $f(\mathbf{V}) = X$, and there will be at least one such assignment to \mathbf{V} for each assignment to X , and exactly one when the representation is simple.

b) Given a CSP P which contains a variable X , a vector of variables \mathbf{V} and the constraint $c_R = f(\mathbf{V}) = X$ where $\langle \mathbf{V}, f \rangle$ is a representation of X , a new CSP P' can be generated by taking any constraint C in P (except c_R) where $X \in \text{vars}(C)$ and replacing X with \mathbf{V} by $\langle \mathbf{V}, f \rangle$ in C . The solutions of P and P' will be identical.

c) Given a CSP P which contains a vector of variables \mathbf{V} and a variable X , which is in only the constraint $f(\mathbf{V}) = X$ where $\langle \mathbf{V}, f \rangle$, a new CSP P' can be generated by removing the variable X and the constraint $f(\mathbf{V}) = X$. The solutions to P' are exactly the solutions to P without the assignment to X .

Proof. a) Trivial, as for each assignment to X there exists at least one (and exactly one if the representation is perfect) assignment to \mathbf{V} such that $f(\mathbf{V}) = X$ is satisfied.

b) Consider an assignment s to $\text{vars}(P)$ which is a solution to P . This will be a solution to P' if and only if s satisfies C with X replaced by \mathbf{V} , as described in Definition 6. However we know that $\text{vars}(P)$ satisfies C , and therefore in particular this assignment satisfies $f(\mathbf{V}) = X$. By the definition of replacing a variable in a constraint with a representation, the original constraint will be satisfied if and only if the new one is. The reverse argument follows identically.

c) Clearly any solution to P when limited to $vars(P')$ will be a solution to P' , as the constraints on P' are a subset of the constraints on P . Given any solution to P' , adding the assignment to X which satisfies $f(\mathbf{V}) = X$ which also clearly be a solution to P , as it is only P' with one extra variable and constraint, and this new constraint is the only one which involves X . \square

Theorem 2 proves that the Definition 6 can be used to apply representations to a CSP and get a new CSP, whose solutions can be mapped to the solutions of the original CSP. One minor problem with Definition 6 is that it assumes constraints are expressed explicitly. Most constraint solvers operate most efficiently on constraint expressed implicitly using a small language of constraints. How an implicit representation of a constraint can be refined to an implicit representation on a representation is a complex issue in its own right, see [3]. In this paper refined constraints will be given implicitly where possible, but how these implicit constraints could be generated will not be discussed.

2.2 Representations of Sets and Multisets

One of the most common types for variables which occur in specifications of combinatorial problems is sets and multisets, and these will be used as the primary example throughout this paper. There are a number of representations of sets and multisets in common usage in constraint programming, and some of these will be explored in this paper. Definition 7 gives a formal way of referring to the various families of sets and multisets which will arise.

Definition 7. *Given set X , natural number occ and integer range $Size$, then $S = MS(X, occ, Size)$ denotes the set of all (multi)sets drawn from X with size in the range $Size$ and where each value occurs at most occ times. Three common special cases of this definition are where occ is 1 (so S represents only sets), $Size$ allows a single value (so S represents a single fixed size of (multi)sets), and $Size$ is a range at least as large as $[0, |X| * occ]$, so it places no limit on the (multi)sets in S , so S is unsized.*

Definition 8. *Given a (multi)set variable X with domain $MS(S, occ, Size)$, the occurrence representation of X is defined to be $\langle \mathbf{V}, f \rangle$, where \mathbf{V} is a vector indexed by S of variables with domain $\{0, \dots, occ\}$. The function f maps assignments of \mathbf{V} to X by $f(\mathbf{v}) = \{v[s] \times s \mid s \in S\}_m$, under the condition the resulting (multi)set is in the domain of X . The representation constraint is therefore given by $sum(\mathbf{V}) \in Size$. \mathbf{V} is called the occurrence vector.*

Definition 9. *Given a (multi)set variable X with domain $MS(S, occ, Size)$, the explicit representation of X is defined to be $\langle \mathbf{E}, f \rangle$, where \mathbf{E} is a vector of variables of domain $S \cup \{\emptyset\}$ (where \emptyset represents a value not in S) indexed by the range $1, \dots, max(size)$. f maps assignments of \mathbf{E} by $f(\mathbf{v}) = \{E[i] \mid 1 \leq i \leq max(size) \wedge E[i] \neq \emptyset\}_m$, where this (multi)set is in $MS(S, occ, Size)$. \mathbf{E} is known as the element vector.*

Definition 10. *Given an abstract variable X with domain $MS(S, occ, Size)$, it is represented under the explicit with check representation by $\langle \mathbf{E.C}, f \rangle$, where*

\mathbf{E} and \mathbf{C} are both of length $\max(\text{size})$, the elements of \mathbf{C} are Boolean variables and the elements of \mathbf{E} have domain S . The function f maps assignments of \mathbf{E} and \mathbf{C} to the (multi)set which contains each $E[i]$ for which $C[i]$ is TRUE, where this (multi)set is in $MS(S, \text{occ}, \text{Size})$. \mathbf{E} is called the element vector, and \mathbf{C} is called the check vector.

Lemma 2. For (multi)sets of domain and size greater than 1 the explicit representation is not simple.

Proof. It is possible to freely permute the vector of variables in the explicit representation and they will represent the same (multi)set, so as long as this vector has length greater than 1 and more than one possible assignment, the representation cannot be simple. \square

The *explicit with check* can be simpler to implement than the *explicit* representation, as it does not require constraints which can cope with a special domain element which represents that variable does not represent a value in the (multi)set. Both these representation have the problem of variable symmetry, as permuting an assignment to the element vector (and identically permuting the check vector in the case of the *explicit with check* representation) generates an assignment which represents the same (multi)set. Only one method of breaking this symmetry shall be considered in this paper, which is to impose that an assignment to the *element* vector only represents a (multi)set when it is ordered according to some ordering of the elements the (multi)set is drawn from. After performing symmetry breaking, both these representations are simple.

Definition 11. The Gent representation² was designed to try to combine the strengths of the occurrence and explicit with symmetry breaking representations. Given a (multi)set variables X with domain $MS(S, \text{occ}, \text{size})$, it is represented with the Gent representation by $\langle \mathbf{V}, f \rangle$, where \mathbf{V} is a vector of length $|S|$ with variables of domain $\{0, 1, \dots, \max(\text{size})\}$. f maps assignments of \mathbf{V} to assignments of X by the following rules, assuming a total ordering on the elements of S . 1) If a is the smallest element in S , $f(\mathbf{v})$ contains $v[a]$ occurrences of a . 2) If $v[b] \neq 0$, then $f(\mathbf{v})$ contains $v[b] - \max\{v[i] \mid i < b\}$ occurrences of b . 3) $f(\mathbf{v})$ is only defined if $a < b \rightarrow (v[b] = 0 \cup v[a] < v[b])$ and the resulting (multi)set is valid assignment to X .

Example 7. If $X = MS(\{1, 2, 3, 4, 5\}, 5, [0, 5])$, then if X is represented under the Gent representation then $\langle 0, 2, 0, 4, 0 \rangle$ represents $\{2, 2, 4, 4\}_M$ (where $\{\}_M$ denotes a multiset) and $\langle 1, 2, 4, 0, 0 \rangle$ represents $\{1, 2, 3, 3\}_M$. The representation of both $\langle 0, 1, 0, 1, 0 \rangle$ and $\langle 0, 2, 0, 1, 0 \rangle$ is undefined, as the non-zero elements are not in strictly increasing order.

There are a number of variants of these representations, some of which modify them in a minor manner, and others which perform large additions. One very common extension, and the only discussed in this paper, is adding a variable

² Discovered by Ian Gent, unpublished

whose value is size of the (multi)set. It is simple to add this variable to each of the representations considered so far. These extended representations will be denoted by adding “+ size” to their title, for example the “occurrence + size representation”.

3 Comparing representations

The definition of representations given in Section 2 provides a method of specifying representations and in the case of variable representations using them to map one CSP to another. In order to become useful to modellers, these definitions must be used to aid choosing the “best representation” to use to solve a particular CSP.

The first obvious problem is to decide on a definition of “best representation”. In particular, solver-specific implementation issues and optimisations mean that even on different versions of the same solver the representation which will solve a problem fastest can change drastically. The next most obvious method of comparing representations, and the one considered here, is to compare the size of search, ignoring how long a particular implementation may take to implement a given implementation. Of course it is not possible to entirely ignore time and memory usage, in particular because except in extreme cases, using a representation should always decrease memory usage and time taken per node, but may increase (and should never decrease) search size.

The strongest possible relationship between two representations A and B of the same variable would be if it could be shown that using A always produced smaller searches than B . Definition 12 defines exactly this property. It is obvious that given this definition, if A dominates B , then given any search which was performed where a variable is represented using B , B could be replaced with A .

Definition 12. $Rep_1 = \langle R_1, f_1 \rangle$ dominates $Rep_2 = \langle R_2, f_2 \rangle$ (or Rep_2 is embedded in Rep_1) if there is a function $M : R_2 \rightarrow R_1$ such that $\forall r \in R_2. f_1(M(r)) = f_2(r)$ and $\forall r_1, r_2 \in R_2. r_1 \leq r_2 \rightarrow M(r_1) \leq M(r_2)$. Rep_1 and Rep_2 are equivalent if Rep_1 dominates Rep_2 and Rep_2 dominates Rep_1 . Rep_1 strictly dominates Rep_2 if Rep_1 dominates Rep_2 and Rep_2 does not dominate Rep_1 .

Dominance is very similar to the idea of expressivity from [8], and for simple representations Lemma 3 shows that they are identical. For non-simple representations however it is necessary to also consider the ordering relation between states, as well as the sub-domains those states represent. Considering only the states which can be represented can suggest a representation is more useful than it actually is during search, as Example 8 demonstrates.

Lemma 3. Given two simple representations $Rep_1 = \langle R_1, f_1 \rangle$ and $Rep_2 = \langle R_2, f_2 \rangle$ then Rep_1 dominates Rep_2 if and only if $\{f_2(r_2) | r_2 \in R_2\} \subseteq \{f_1(r_1) | r_1 \in R_1\}$.

Proof. If Rep_1 dominates Rep_2 , then there exists a function $M : R_2 \rightarrow R_1$ such that $\forall r \in R_2. f_1(M(r)) = f_2(r)$ and therefore $\forall r_2 \in R_2. \exists r_1 \in R_1. f_1(r_1) = f_2(r_2)$.

In the opposite direction, as the set of sub-domains represented Rep_2 is a subset of those represented by Rep_1 and as both representations are simple (and so each sub-domain is represented at most once), then it is simple to define a function M from R_2 to R_1 by $f(r_2) = r_1 \leftrightarrow f_1(r_1) = f_2(r_2)$. As the ordering on simple representations is entirely determined by the domains they represent, this function defines the dominance between Rep_1 and Rep_2 . \square

Example 8. Consider representing two element multisets drawn from $\{1, 2, 3\}$ using the explicit representation with and without symmetry breaking.

The sub-domain $\{\{1, 1\}_M, \{1, 2\}_M, \{2, 3\}_M\}$ can be represented by the sub-domains $\langle\{1, 2\}, \{1, 3\}\rangle$ if symmetry breaking is not imposed, but if symmetry breaking is imposed the smallest sub-domains which contain these assignments are $\langle\{1, 2\}, \{1, 2, 3\}\rangle$, which also allows $\{2, 2\}_M$.

Consider now representing the sub-domain $\{\{1, 2\}_M, \{2, 3\}_M\}$. These can be represented exactly by the sub-domains $\langle\{2\}, \{1, 3\}\rangle$ without symmetry breaking, but require at least the sub-domains $\langle\{1, 2\}, \{2, 3\}\rangle$ with symmetry breaking, which also allow the multiset $\{2, 2\}_M$.

Without symmetry breaking, there are more possible sets of sub-domains which can be represented. This comes at the cost however that the domains which were shown second are not reachable from those which were generated first. It is therefore misleading to simply list the states which are reachable during search in the case of non-simple representations.

3.1 Dominance in variable representations

While dominance is a useful property, proving if one representation dominates another is non-trivial directly from the definition, and therefore finding simpler methods of proving dominance is useful. For simple variable representations, Theorem 3 gives a sufficient, although not necessary, to prove one simple representation dominates another.

Definition 13. *A set of channelling constraints between two variable representations $R_1 = \langle \mathbf{V}_1, f_1 \rangle$ and $R_2 = \langle \mathbf{V}_2, f_2 \rangle$ of a variable X is a set of constraints over \mathbf{V}_1 and \mathbf{V}_2 such that an instantiation \mathbf{v}_1 of \mathbf{V}_1 and \mathbf{v}_2 of \mathbf{V}_2 satisfies all the constraints if and only if either $f_1(\mathbf{v}_1)$ is equal to $f_2(\mathbf{v}_2)$, or both are undefined.*

Theorem 3. *Given two simple variable representations $R_1 = \langle \mathbf{V}, f \rangle$ and $R_2 = \langle \mathbf{W}, g \rangle$ of a variable X and a set C of channelling constraints between \mathbf{V} and \mathbf{W} , where each $c \in C$ can be expressed in the form $v = x \leftrightarrow (w_1 = y_1 \vee \dots \vee w_n = y_n)$ for constants x, y_1, \dots, y_n and variables $v \in \mathbf{V}$ and $w_i \in \mathbf{W}$, then R_2 dominates R_1 .*

Proof. Consider a sub-domain \mathbf{V}' of \mathbf{V} which is GAC with respect to the representation constraint, and construct the maximal sub-domain \mathbf{W}' of \mathbf{W} which is GAC with respect to the representation constraint and \mathbf{W}' and \mathbf{V}' together are GAC with respect to C . By construction, for each assignment $v' \in \mathbf{V}'$ where $f(v')$ is defined, there will exist a $w' \in \mathbf{W}'$ such that $g(w') = f(v')$.

Consider an assignment $w' \in \mathbf{W}'$ such that $g(w')$ is defined. There must exist a unique assignment $v \in \mathbf{V}$ such that $g(w') = f(v)$ and $c(w', v)$ holds for all $c \in C$. It remains to show this assignment is allowed by \mathbf{V}' .

Consider a single channelling constraint in C . if the constraint is true, either both the left and right hand side of the constraint are false, or the comparison on the left hand side and at least one comparison on the right hand side are true. Given a sub-domain of \mathbf{W} therefore, by looking at each channelling constraint in turn it is possible to construct a list of assignments to variables in \mathbf{V} which must be allowed, and which must be forbidden. Considering a larger sub-domain of \mathbf{W} can only increase the number of assignments to variables in \mathbf{V} which must be allowed. Therefore as $C(\mathbf{v}, \mathbf{w})$ is true, then as \mathbf{W}' contains \mathbf{w}' , \mathbf{V}' must contain \mathbf{v}' , because GAC $C(\mathbf{V}', \mathbf{W}')$ holds. \square

- Theorem 4.** 1. *The Gent representation dominates the occurrence representation for representing sets.*
2. *The Gent + size representation dominates the occurrence + size representation for representing sets.*
3. *The Gent representation dominates the explicit with symmetry breaking representation for fixed sized sets and multisets.*
4. *The Gent + size representation dominates the explicit with symmetry breaking representation for variable sized sets and multisets.*
5. *The explicit and explicit with check representations with symmetry breaking are equivalent on variable and fixed sized sets and multisets.*

Proof. From Theorem 3, we only need to find a set of channelling constraints of the appropriate form to prove dominance. The required channelling constraints are given below. In each case, a CSP variable $X = MS(S, occ, size)$ is represented in multiple ways.

1. Represent X as $\langle \mathbf{V}, f \rangle$ under the *ordered occurrence* representation and $\langle \mathbf{W}, g \rangle$ under the *occurrence* representation. Consider the set of constraints $\{W[i] = 0 \leftrightarrow V[i] = 0 | i \in X\} \cup \{W[i] = 1 \leftrightarrow V[i] = 1 \vee \dots \vee V[i] = size | i \in X\}$.
2. Represent X as $\langle \mathbf{V}.S_1, f \rangle$ under the *ordered occurrence + size* representation (where S_1 is a variable which represents the size of X), and $\langle \mathbf{W}.S_2, g \rangle$ under the *occurrence + size* representation. Consider the sets of constraints $\{W[i] = 0 \leftrightarrow V[i] = 0 | i \in X\} \cup \{W[i] = 1 \leftrightarrow V[i] = 1 \vee \dots \vee V[i] = size | i \in X\}$ and $\{S_1 = i \leftrightarrow S_2 = i | i \in size\}$.
3. Represent X as $\langle \mathbf{V}, f \rangle$ under the *ordered occurrence* representation and $\langle \mathbf{W}, g \rangle$ under the *explicit* representation. Consider the set of constraints $\{W[i] = x \leftrightarrow V[x] = \{i | x \in X, i \in \{1, \dots, size\}\} \cup \{W[i] = \emptyset \leftrightarrow FALSE | i \in \{1, \dots, size\}\}$.
4. Represent X as $\langle \mathbf{V}.S, f \rangle$ under the *ordered occurrence + size* representation and as $\langle \mathbf{W}, f \rangle$ under the *explicit* representation. Consider the set of constraints $\{W[i] = x \leftrightarrow V[x] = i | x \in X\} \cup \{W[i] = \emptyset \leftrightarrow S = 0 \vee \dots \vee C = i | i \in size\}$.
5. Represent X as $\langle \mathbf{V}, f \rangle$ under the *explicit* representation and as $\langle \mathbf{W}.C, g \rangle$ under the *explicit with check* representation. The set of constraints $\{V[i] =$

$x \leftrightarrow W[i] = x | x \in X, i \in size \} \cup \{V[i] = \emptyset \leftrightarrow C[i] = 1 | x \in X$ show that the *explicit with check* representation dominates the *explicit* representation and the set of constraints $\{C[i] = 0 \leftrightarrow V[i] = \emptyset | i \in size \} \cup \{C[i] = 1 \leftrightarrow V[i] \in X | i \in size \} \cup \{W[i] = x \leftrightarrow V[i] = x \vee V[i] = \emptyset | i \in size, x \in X \}$ demonstrates the converse.

□

4 Perfect representations

While dominance provides a powerful method of comparing representations, it is often too coarse. A more precise system would compare representations with respect to a specific problem. Unfortunately, performing this usefully in general has proved difficult, but this section studies a useful special case of this problem, showing when a representation of some variable is equivalent to the complete representation with respect to one or more constraints.

Definition 14. *Given a constraint C , a set $T = \{\langle R_v, f_v \rangle\}$ of representations for some subset of $vars(C)$, some $\langle R_t, f_t \rangle \in T$ which represents a variable X , and the constraint C' obtained by applying each representation in T to variables in $vars(C)$, then “ $\langle R_t, f_t \rangle$ is perfect with respect to C and T ” if given a sub-domain A of $vars(C')$ which is GAC with respect to C' , then any assignment to R_t which is allowed by A and has an image under f_t can be extended to an assignment to A which satisfies C' .*

Note that the definition of perfect considers a set of representations. This is because replacing many variables with representations may be perfect with respect to each of these representations, while replacing any single variable with a representation is not.

It follows from the definition of perfect that using a perfect representation instead of the original variable will not lead to an increase in search space assuming GAC propagation as whenever GAC propagation occurs, all assignments which represent an assignment to the original variable and which do not satisfy the constraint will be removed.

Example 9. Consider a variable A with domain $MS(\{1, \dots, n\}, 1, [0 \dots n])$ represented with the *occurrence* representation with *occurrence* vector \mathbf{V} . The constraint $a \in A$ for a constant value a is mapped via the representation to the constraint $V[a] = 1$. Any sub-domain of \mathbf{V} which is GAC with respect to this constraint will clearly have $V[a]$ with only 1 in its domain and therefore any assignment to \mathbf{V} which has represents an element of A will satisfy the original constraint. The *occurrence* representation is therefore perfect with respect to the constraint $a \in A$.

Example 9 gives an example of a perfect representation with respect to a specific constraint, while Example 5 gave an example of a non-perfect one, which leads to an exponential increase in search size compared to using the complete representation. In a similar fashion to Theorem 3, which showed a useful sufficient condition for one variable representation to dominate another, Theorem 5

shows a necessary and sufficient (although only sufficient is proved due to space restrictions) condition for a variable representation to be perfect with respect to a given constraint.

Definition 15. *Given a constraint C and a set of constraints K , then a set of constraints S is defined to be a split of C in the context of K if $\text{vars}(C) = \cup\{\text{vars}(s) \mid s \in S\}$, $(s, t \in S \wedge s \neq t) \rightarrow \text{vars}(s) \cap \text{vars}(t) = \emptyset$ and $C \wedge K$ is logically equivalent to $\wedge\{s \mid s \in S\} \wedge K$.*

Lemma 4. *Given a split S of a constraint C in the context of K , then the set of constraints S' generated by creating a new constraint s' for each $s \in S$ where $\text{vars}(s')$ is the same as $\text{vars}(s)$ and an assignment is allowed by s' if it can be extended to a valid assignment of C is also a split of C in the context of K .*

Proof. There cannot be an assignment to $\text{vars}(C)$ which is allowed by $C \wedge K$ and not by $(\wedge S') \wedge K$ as each $s' \in S'$ accepts any assignment which can be extended to a complete assignment of $\text{vars}(C)$ which satisfies C . Furthermore, all assignments to $(\wedge S') \wedge K$ must be allowed by $(\wedge S) \wedge K$ as the constraints in S' allow the minimal possible set of assignments required to satisfy that they are a split. Therefore $C \wedge K \rightarrow (\wedge S') \wedge K \rightarrow (\wedge S) \wedge K \leftrightarrow C \wedge K$, and therefore all 3 must be equivalent. \square

Theorem 5. *Given a constraint C and the constraint C' generated by applying a set of variable representations $R = \{\mathbf{V}_i, f_i\}$ to C , a distinguished representation $\langle \mathbf{V}_r, f_r \rangle \in R$, and the set of constraints K generated by taking the representation constraints of each element of R , then r is perfect with respect to C and R if C' can be split into a set of constraints S in the context of K , each of which contains at most one variable from each \mathbf{V}_r .*

Proof. Without loss of generality, we will assume any splits are in the form discussed in Lemma 4. Consider the case where a split exists. We must show that this implies that given any sub-domain of $\text{vars}(C')$, if it is GAC with respect to C' , any assignment to \mathbf{V}_r can be extended to a valid assignment of $\text{vars}(C')$ with respect to both C' and the representation constraints of all the elements of R . If this was not the case, this would mean there was a sub-domain of $\text{vars}(C')$ which is GAC with respect to C' but there was an assignment to one of the \mathbf{V}_r which could not be extended to a valid assignment of $\text{vars}(C')$. This would mean also it could not be extended to a valid assignment which satisfied all the constraints in a split of C' . As no variable is in more than one constraint of the split, this would mean that at least one constraint in the split had an assignment which was not satisfiable. However each constraint in the split contains at most one variable from \mathbf{V}_i , so that variable can be pruned, and therefore one constraint in the split of C' was not GAC, and therefore neither was C' . \square

Corollary 1. *The occurrence representation is perfect when all variables in the constraints $A \cup B = C$, $A \cap B = C$ and $A \subseteq B$ are represented by the occurrence representation, and the (multi)set variables A, B and C are unsized.*

Proof. If A , B and C are represented by *occurrence* representation with vectors A' , B' and C' , then each of the constraints trivially splits to satisfy Theorem 5, for example for $A, B, C = MS(\{1, \dots, n\}, 1, [0, n])$, $A \cup B = C$ becomes the set of constraints $\{A'[i] \wedge B'[i] = C'[i] | i \in \{1, \dots, n\}\}$. \square

Although we shall not prove it here, the explicit representation is not perfect for any of these constraints. This can be seen intuitively, as all these constraints involve examining every variable in representation to know if a certain element is in the set or not.

5 Conclusion

This paper has extended the study of representations by giving an implementation independent definition of representations and shown how this can be used to usefully compare representations both to each other in both a problem-dependent and problem-independent manner. Of particular interest is that a previously unstudied representations of sets and multisets has been proved to perform better than two frequently used representations. Further, we have shown that for problems which involve most common set constraints excluding cardinality, the occurrence representation performs as well as the theoretically best representation. The work in this paper is being further expanded to cover other families of constraints, in particular quantified constraints, and also other types of variables, such as graphs, are being further investigated.

References

1. C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. Disjoint, partition and intersection constraints for set and multiset variable. In *Proceedings of CP04*, 2004.
2. C. Bessière and J.-C. Régin. Arc consistency for general constraint networks: Preliminary results. In *IJCAI (1)*, pages 398–404, 1997.
3. A. M. Frisch, C. Jefferson, B. Martínez Hernández, and I. Miguel. The rules of constraint modelling. In *Proceedings of the 19th International Joint Conferences on Artificial Intelligence*, 2005.
4. C. Gervet. Conjunto: constraint logic programming with finite set domains. In M. Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, pages 339–358, Massachusetts Institute of Technology, 1994. The MIT Press.
5. W. Harvey and P. Stuckey. Improving linear constraint propagation by changing constraint representation. *Constraints*, 8[2]:173–207, 2003.
6. B. Hnich. *Function Variables for Constraint Programming*. PhD thesis, Uppsala, 2004.
7. B. Hnich, B. Smith, and T. Walsh. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research, Volume 21*.
8. T. Walsh. Consistency and propagation with multiset constraints: A formal viewpoint. In *Proceedings of CP*, 2003.