TLP **17** (4): 365–407, 2017. © Cambridge University Press 2017
doi:10.1017/S1471068417000126 First published online 27 June 2017

Contractibility for open global constraints

MICHAEL J. MAHER
Reasoning Research Institute, Canberra, Australia
(*e-mail:* michael.maher@reasoning.org.au)

submitted 20 December 2016; revised 20 December 2016; accepted 28 February 2017

Abstract

Open forms of global constraints allow the addition of new variables to an argument during the execution of a constraint program. Such forms are needed for difficult constraint programming problems, where problem construction and problem solving are interleaved, and fit naturally within constraint logic programming. However, in general, filtering that is sound for a global constraint can be unsound when the constraint is open. This paper provides a simple characterization, called contractibility, of the constraints, where filtering remains sound when the constraint is open. With this characterization, we can easily determine whether a constraint has this property or not. In the latter case, we can use it to derive a contractible approximation to the constraint. We demonstrate this work on both hard and soft constraints. In the process, we formulate two general classes of soft constraints.

KEYWORDS: global constraints, open constraints, soft constraints.

1 Introduction

Constraint Logic Programming (CLP) (Jaffar and Maher 1994) provides the ability to add variables and constraints to a constraint store during the course of an execution. In this it is not alone: linear and integer programming solvers and solvers presented as libraries for an underlying programming language also allow the introduction of new variables and constraints in an incremental way. In some problems, it is natural for the presence of some variables to be contingent on the value of other variables. This is true of configuration problems and scheduling problems that involve process-dependent activities (Mittal and Falkenhainer 1990; Barták 2003). More generally, for difficult problems the intertwining of problem construction and problem solving provides a way to manage the complexity of a problem, and thus new variables and constraints may arise after solving has begun. Thus, CLP is particularly well suited for such problems, in contrast to compilation-based modelling languages, such as MiniZinc (Nethercote *et al.* 2007), where all variables and constraints must be fixed at compilation time.

CLP also supports global constraints, which have been an important part of the success of constraint programming. However, most implementations of global constraints adopt a non-incremental approach: the variables constrained by a global constraint are fixed when the constraint is imposed. Thus, the collection

of variables they constrain is *closed*, rather than *open*. This restricts the exploitation of incrementality that is available in CLP languages. Delaying the imposition of a global constraint until all variables it might involve have been generated can leave the filtering effect of the global constraint until too late in the execution, resulting in a large search space. Open global constraints remove this limitation by allowing variables to be added dynamically.

A major difficulty in implementing open constraints is that a propagator for a closed constraint may be unsound for the corresponding open constraint. That is, the propagator may make an inference that turns out to be unjustified, once the sequence of variables is extended. In this paper, we focus on the issue of identifying constraints for which a closed propagator is sound as an open propagator. These constraints have a simple characterization, which we call *contractibility*, and which allows us to easily determine whether a given constraint has this property. This characterization is also convenient for finding the tightest contractible approximation of an uncontractible constraint, which can be the basis for an open propagator of the constraint. We illustrate our results with a wide variety of global constraints, including both hard and soft constraints.

As part of our treatment of soft constraints, we formulate two very general classes of soft constraints based, respectively, on constraint decomposition and edit distance. These classes unify and generalize several different proposals in the literature. Using these formulations, we introduce general results and techniques for establishing that a constraint is contractible. It turns out that finding a tightest contractible approximation is more difficult for soft constraints than for hard constraints. In particular, while we can mathematically characterize the tightest approximation, and define some pragmatic generic non-tight approximations, we show that the tightest contractible approximation cannot always be represented in the edit-distance framework.

This paper is arranged as follows. After some preliminaries in Section 2 and a discussion of open constraints in Section 3, we introduce contractibility in Section 4. We show that it characterizes those constraints for which closed propagators remain sound when the constraint is open, and develop an algebra for constructing contractible constraints. We conclude Section 4 by characterizing contractibility in language-theoretic terms, and use that characterization to identify contractible constraints (Section 5) and tight approximations of uncontractible constraints (Section 6). We show that, with a tight approximation, a proposal of Barták for implementing open uncontractible constraints achieves an appropriate consistency. We then address the same issues for soft constraints (Sections 7 and 8).

This paper incorporates results announced in Maher (2009b,c,d, 2010). It includes unpublished proofs, strengthened results, new results, and some additional discussion.

2 Background

The reader is assumed to have a basic knowledge of constraint programming, constraint satisfaction problems (CSPs), global constraints, and filtering, as might be found in Dechter (2003), Rossi *et al.* (2006), and Beldiceanu *et al.* (2005).

For the purposes of this paper, a global constraint is a relation over a single sequence of variables. Other arguments of a constraint are considered parameters and are assumed to be fixed before execution. Throughout this paper, a sequence of variables will be denoted, interchangeably, by $\vec{X}$ or $[X_1, \ldots, X_n]$. We make no *a priori* restriction on the variables that may participate in the sequence except that, in common with most work on global constraints, we assume that no variable appears more than once in a single constraint.

There are some specific global constraints that we define for completeness. These and other global constraints are discussed more completely in Beldiceanu *et al.* (2005) and the references therein. As with variables, a sequence of values v_i is expressed by $\vec{v}$. The constraint AllDifferent($[X_1, \ldots, X_n]$) (Régin 1994) states that the variables $X_1, \ldots, X_n$ take distinct values. The global cardinality constraint GCC($\vec{v}, \vec{l}, \vec{u}, [X_1, \ldots, X_n]$) (Régin 1996) states that, for every i, the value v_i occurs between l_i and u_i times in the list of variables. The constraint NValue($[X_1, \ldots, X_n], N$) (Pachet and Roy 1999) states that there are exactly N distinct values in $X_1, \ldots, X_n$. The constraint Regular($\mathscr{A}, [X_1, \ldots, X_n]$) (Pesant 2004) states that the value of the list of variables, when considered as a word, is accepted by the automaton $\mathscr{A}$. Similarly, the constraint CFG($\mathscr{G}, [X_1, \ldots, X_n]$) (Quimper and Walsh 2006; Sellmann 2006) (called Grammar in Quimper and Walsh (2006)) states that the value of the list of variables, when considered as a word, is generated by the context-free grammar $\mathscr{G}$.

The constraint Sequence($l, u, k, [X_1, \ldots, X_n], \vec{v}$) (Beldiceanu and Contejean 1994) states that any consecutive sequence of k variables $X_j, \ldots, X_{j+k-1}$ contains between l and u occurrences of values from $\vec{v}$. The constraint SlidingSum($l, u, k, [X_1, \ldots, X_n]$) (Beldiceanu and Carlsson 2001) states that the sum of any consecutive sequence of k variables lies between l and u. The constraint Contiguity($[X_1, \ldots, X_n]$) (Maher 2002) states that the variables X_i take values from $\{0, 1\}$ and the variables taking the value 1 are consecutive. The lexicographical ordering constraint $[X_1, \ldots, X_n] \leqslant_{lex} [Z_1, \ldots, Z_n]$ (Frisch *et al.* 2002) states that the sequence of X variables is lexicographically less than or equal to the sequence of Z variables, where we assume some ordering on the underlying values. The precedence constraint $s \prec_{\vec{X}} t$ (Law and Lee 2004) states that if t appears in the sequence $\vec{X}$, then s appears at a lower index.

For some constraints, like AllDifferent, GCC, and NValue, the order of variables is immaterial to the semantics of the constraint. We say a constraint C is *order-free* if

$$C([X_1, \ldots, X_n]) \leftrightarrow C([X_{\pi(1)}, \ldots, X_{\pi(n)}])$$

for every permutation π of $1..n$. The other constraints mentioned above are not order free.

We assume that the argument $\vec{X}$ of a use of a global constraint has a static type T that assigns, for every position i, a set of values. Thus, every variable X in $\vec{X}$ has a static type $T(X)$ of values that it may take. We will also view T as a unary predicate on the variables of $\vec{X}$, where $T(X)$ is true iff X takes a value from its static type. In addition, generally, each variable has an associated set $S \subseteq T(X)$ of values, called its domain. We will view this simultaneously as: a function $D : \vec{X} \to 2^{Values}$, where

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

$D(X) = S$ and $Values = \bigcup_{X in \vec{X}} T(X)$, a unary relation $D(X)$, which is satisfied only when the value of X is some $s \in S$, and the pointwise extension of D to sequences of variables.

We formalize the semantics of a global constraint C as a formal language L_C. A word $d_1 d_2 \ldots d_n$ appears in L_C iff the constraint $C([X_1, X_2, \ldots, X_n])$ has a solution $X_1 = d_1, \ldots X_n = d_n$. Thus, for example, the semantics of AllDifferent is $\{a_1 \ldots a_n \mid \forall i, j \; i \neq j \rightarrow a_i \neq a_j, n \in \mathbb{N}\}$ and the semantics of Regular$(\mathscr{A}, \vec{X})$ is $L(\mathscr{A})$, the language accepted by $\mathscr{A}$. When it is convenient, we will describe languages with Kleene regular expressions (Hopcroft and Ullman 1979). For a given use of a constraint $C(\vec{X})$, we write $T(\vec{X})$ for the language defined by the static type of $C(\vec{X})$.

The following definitions will be important later. Let $P(L) = \{w \mid \exists u \; wu \in L\}$ denote the set of prefixes of a language L, called the *prefix-closure* of L. We say L is *prefix-closed*, if $P(L) = L$. We say two languages L and L' are *prefix-equivalent* if $P(L) = P(L')$.

3 Open constraints

There are many problems that are dynamic in nature but to which we would like to apply constraint techniques. Barták (1999) describes a class of complex processing environments, where there may be alternative processing routes, different production formulas, and alternative raw materials. In addition to the core products of the processes, there may be by-products and co-products, which require additional processing. Some instances of products may be re-processed or recycled. Because of storage limits and/or a necessity to work with the instances while they are still in an amenable state, such instances might need to be re-processed or recycled promptly. In such environments, process scheduling must be dynamic: additional tasks may arise from re-processing, and additional raw materials may arise.

Many production processing environments have these characteristics. Consider, for example, sugar cane processing. Juice is extracted from the sugar cane and clarified before it is refined. Refining involves repeated crystallization and centrifuging processes, with molasses produced as a by-product. Usually, three repetitions of these processes are performed but, through natural variation of the raw materials, an additional repetition may be needed. Such a need can be identified through monitoring the refinement process.

Now, consider a constraint-based approach to the problem of the on-going scheduling of these processes. We might use a Cumulative constraint to express the limited availability of centrifuges. When a batch requires an additional repetition, a new task must be added to that constraint and additional constraints concerning the task must be added to the problem. Thus, we require that Cumulative be an *open* constraint – able to accept additional tasks.

Open constraints pre-suppose the existence of a meta-program that can impose constraints, close an open constraint, add variables to an open constraint, (possibly) create new variables, and interact with the execution of the constraint system, possibly controlling it. In this paper, we will abstract away the details of the meta-program so that we can focus on the open constraints. We assume that the collection

of variables forms a sequence, to which variables may be added at the right-hand end only[1]. The scope of constraints changes during the execution, and we refer to the state of the constraint at some point in the execution as an *occurrence* of the constraint. In open global constraints C, the length of the sequence of variables varies and consequently the semantics in terms of the language L_C is particularly appropriate.

There are three models of open constraint that have been proposed[2]. Barták (2003) first formulated this issue and described a straightforward model: the constraint involves a sequence of variables to which variables may be added. Thus, the arity and type of the constraint are unchanged, whether the constraint is open or closed. Barták (2003) outlined a generic implementation technique to make open versions for the class of *monotonic* global constraints. Barták focussed on a specific implementation of the open AllDifferent constraint. This is an order-free constraint, and details of the model, such as where variables are added to the sequence, are left unspecified. The remaining models extend this model by incorporating more details about the possible extension of the sequence; for these models the constraint has a different arity or type.

The model of van Hoeve and Régin (2006) only applies to order-free constraints expressed in the form $C(S)$. It uses a set variable S describing a *set* of object variables, rather than a sequence, to represent the collection of variables in the constraint[3]. The lower bound of S is the set of variables that are committed to appear in the constraint; the upper bound is the set of variables that are permitted to appear in the constraint. Thus, there is a finite set of variables that might appear in the constraint, and these are fixed in advance. The authors refer to the constraint as open "in a closed world", since the set of variables that might be added to the constraint is closed. The model makes elegant use of existing implementations of set variables and their associated bounds. However, the use of a constraint in this model requires knowing all the variables that might appear before imposing the constraint. As a result, it cannot deal well with contingent variables. They create a similar problem to the one faced by closed constraints: the constraint may be imposed late in the execution, creating a larger search space.

The third model (Maher 2009b) is, in some ways, intermediate between that of Barták (2003) and van Hoeve and Régin (2006). Under this model, a constraint $C(\vec{X}, N)$ acts on both a sequence of variables $\vec{X}$ and an integer variable N, representing the length of the sequence once it is closed. Variables can only be added at the right-hand end of the sequence. This is a more detailed model than Barták's. In one sense, this model is an abstraction of the model of van Hoeve and

[1] There is a brief discussion of the effect of alternatives in Section 9.

[2] The terminology "open constraint satisfaction problem" was introduced by Faltings and Macho-Gonzalez (2002, 2005). However, that use refers to problems in which the set of variables is closed but the domains are open, that is, extra values can be added to variable domains. That work is not technically related to "open constraints" as used in this paper, but it shares with this paper an interest in constraint problems that may change over time.

[3] A set variable S ranges over sets and is constrained by two fixed finite sets L and U, which are a lower and upper bound on the value of the variable: $L \subseteq S \subseteq U$. See Gervet (1997).

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Régin (2006): if N is subject only to lower and upper bounds, then the bounds on N correspond to the cardinalities of the bounds of S. It does not have the weakness of that model that the variables that might appear are fixed in advance. On the other hand, the van Hoeve-Regin model has more information about how $\vec{X}$ might be extended, and so might be able to perform stronger propagation.

The model, we employ here is Barták's model, where we specify that variables may be added only at the right-hand end of the sequence. It is equivalent to a weak form of the model of Maher (2009b), where there are no restrictions on N. However, the notion of contractibility, to be introduced in the next section, is relevant for other models of open constraints. Some results are given in Maher (2009b) for the model treated there. We will assume that the only operations that can be applied to an open constraint are adding a variable and designating the constraint closed, so that no more variables may be added.

Constraint programming with open constraints is a special case of dynamic CSPs in the broad sense described in Dechter and Dechter (1988). Work on dynamic CSPs has focussed on the addition and retraction of constraints (Bessière 1991; Hentenryck and Provost 1991; Georget *et al.* 1999; Debruyne *et al.* 2003). It does not directly address the addition of variables to a constraint, although that can be viewed as a combined retraction and addition of constraints. See Verfaillie and Jussien (2005) for a survey on dynamic constraint solving. Work on conditional CSPs, initiated in Mittal and Falkenhainer (1990), addresses contingent variables by explicitly embedding the contingent nature within a CSP, but that work does not address the addition of variables to constraints.

Other forms of dynamism have been addressed in the context of constraints by allowing variable domains to be initially incomplete and expand over time (Faltings and Macho-Gonzalez 2005; Gavanelli *et al.* 2005), or by formulating constraints over a stream of values (Lallouet *et al.* 2011). That work is not technically related to the work in this paper.

We take *filtering* or *propagation* to refer to any algorithm f that reduces domains, that is, $\forall X\ f(D)(X) \subseteq D(X)$. A filtering algorithm f for a constraint C is *sound*, if every solution of C in D also appears in $f(D)$. Some filtering algorithms are characterized by consistency conditions. For closed constraints, the strongest filtering/consistency condition that addresses each constraint separately is domain consistency. A closed constraint $C(X_1, X_2, \ldots, X_n)$ is *domain consistent* if for every i, where $1 \leqslant i \leqslant n$ and every $d \in D(X_i)$, there is a word $d_1 \ldots d_n$ in L_C, such that $d_i = d$ and $d_j \in D(X_j)$ for $j = 1, \ldots, n$.

Because some of the variables in an open constraint will be unspecified during part of the execution, we need to adapt the definition of consistency. The following is an appropriate form of domain consistency for Barták's model.

Definition 1

Given a domain D, an occurrence of a constraint $C(\vec{X})$ is *open D-consistent* if, for every $X_i \in \vec{X}$ and every $d \in D(X_i)$, there is a word $d_1 \ldots d_m$ in L_C, such that $d_i = d$, $|\vec{X}| \leqslant m$, and $d_j \in D(X_j)$ for $j = 1, \ldots, |\vec{X}|$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

When C is closed, the only words of interest in L_C are those of length $|\vec{X}|$. In that case, open D-consistency reduces to domain consistency.

4 Contractibility

We want to extend a constraint $C([X_1,\ldots,X_n])$ with an extra variable Y to $C([X_1,\ldots,X_n,Y])$. We would like to do filtering on the smaller constraint without knowing, whether it will be extended to Y, or further, and without creating a choicepoint. When we can do this, we have a kind of monotonicity property of C.

Definition 2
We say a constraint $C([X_1,\ldots,X_n])$ is *contractible* if there is a number m such that for all $n \geqslant m$ we have

$$C([X_1,\ldots,X_n,Y]) \rightarrow C([X_1,\ldots,X_n])$$

The least such m is called the *contractibility threshold*.

For this paper, we consider only constraints with a contractibility threshold of 0.

Thus, C is contractible iff every solution of $C([X_1,\ldots,X_n,Y])$, when restricted to $X_1,\ldots,X_n$, where $m \leqslant n$, is a solution of $C([X_1,\ldots,X_n])$. The property is akin to the "optimal substructure" property that is a prerequisite for the use of dynamic programming in optimization problems (Cormen *et al.* 2001), which requires that optimal solutions of a problem also solve subproblems optimally. Here, it is only satisfiability, and not optimality, that is involved.

It follows that any sound form of filtering (such as arc consistency or bounds consistency) on a contractible constraint $C([X_1,\ldots,X_k])$ is safe in the sense that any values deleted from domains in that process could also be deleted while filtering on $C([X_1,\ldots,X_n])$ for any $n \geqslant k$. Recall that we use $\vec{X}$ and $[X_1,\ldots,X_n]$ interchangeably.

Proposition 1
Let C be a contractible constraint. Suppose a sound filtering algorithm for $C([X_1,\ldots,X_n])$ reduces the domain D for $\vec{X}$ to D'. Then,

$$D(\vec{X}) \wedge C([X_1,\ldots,X_n,Y]) \leftrightarrow D'(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$$

Furthermore, if this property holds for all domains and all sound filterings, then C must be contractible.

Proof
Let σ be a solution of $D(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$. By contractibility of C, σ satisfies $C([X_1,\ldots,X_n])$. By the soundness of the filtering, σ satisfies $D'(\vec{X})$. Hence, σ satisfies $D'(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$. Since σ is an arbitrary solution,

$$D(\vec{X}) \wedge C([X_1,\ldots,X_n,Y]) \rightarrow D'(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$$

Since D' results from filtering D, $D'(\vec{X}) \rightarrow D(\vec{X})$, and hence the reverse direction also holds.

Now, suppose this property holds for all sound filterings $D \rightsquigarrow D'$ but C is not contractible. Because C is not contractible, there must be a number n and a valuation

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

σ that satisfies $C([X_1,\ldots,X_n,Y])$ but not $C([X_1,\ldots,X_n])$. Let D be the domain that defines σ and D' be the empty (unsatisfiable) domain. Then, the reduction of D to D' is sound for $C([X_1,\ldots,X_n])$ and so, by the previous supposition

$$D(\vec{X}) \wedge C([X_1,\ldots,X_n,Y]) \rightarrow D'(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$$

However, $D(\vec{X}) \wedge C([X_1,\ldots,X_n,Y])$ is satisfiable by σ, while $D'(\vec{X})$ is unsatisfiable, which contradicts this statement. This contradiction shows that C must be contractible. □

Consequently, for contractible constraints, filtering does not need to be undone, if the list is lengthened. That is, algorithms for filtering a closed contractible constraint are valid also for the corresponding open constraint.

Conversely, any constraint that is not contractible might need to undo the effects of filtering, if the list is lengthened. If σ is a solution of $C([X_1,\ldots,X_n,Y])$, but not of $C([X_1,\ldots,X_n])$, then propagation on $C([X_1,\ldots,X_n])$ might eliminate σ. For example, a constraint $\sum_i X_i = 5$ would propagate $X_1 = 5$, if the sequence $\vec{X}$ contains just one variable, thus eliminating solutions, such as $X_1 = 2, X_2 = 3$. When the second variable is added, all propagation that is a consequence of the inference $X_1 = 5$ must be undone.

The second part of this proposition shows that contractibility exactly characterizes the guarantee that closed filtering is safe for open constraints. That is, it is exactly the contractible constraints for which it is always sound to interleave closed filtering and addition of new variables.

Furthermore, the proof of the second part requires very little of the filtering algorithm. Hence, whether we maintain arc consistency or weaker consistencies like bounds consistency or forward checking, contractibility is necessary to soundly interleave closed filtering and the addition of new variables.

We say a domain D *defines an assignment* if $\forall X \ |D(X)| = 1$; in that case, the assignment maps each X to the element of $D(X)$. We say filtering performs *complete checking* if, whenever D defines an assignment, the result of filtering with a constraint C is D iff the assignment satisfies C. Complete checking can be considered a minimal requirement for filtering methods (Schulte and Tack 2009). Any filtering method that satisfies this minimal requirement requires contractibility to guarantee that closed filtering is sound for an open constraint.

Corollary 2

Let C be a constraint, and consider a sound filtering method that performs complete checking. It is always sound to interleave filtering and the addition of new variables iff C is contractible.

The notion of contractibility is a variation of Barták's monotonicity (Barták 2003), where we do not explicitly discuss variable domains. Before proceeding, we make this claim precise. We formulate Barták's monotonicity as follows.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Definition 3
Let D be a domain. We say a constraint C is *monotonic* with respect to D if, for any pair of disjoint sequences of variables $\vec{X}$ and $\vec{Y}$

$$\{\vec{X} \mid C(\vec{X}\vec{Y}) \wedge D(\vec{X}) \wedge D(\vec{Y})\} \subseteq \{\vec{X} \mid C(\vec{X}) \wedge D(\vec{X})\}$$

Contractibility differs from monotonicity in that the definition is based entirely on the constraint, independent of the domains of variables. Hence, it is not tied to domain-based reasoning; it is equally compatible with the more general framework of Maher (2009a). On the other hand, monotonicity is more flexible in reasoning about constraints that are only "partly contractible". The close relationship between monotonicity and contractibility is clear.

Proposition 3
If C is contractible, then for any domain D, C is monotonic with respect to D. Conversely, if C is monotonic with respect to every domain D, then C is contractible.

Proof
By repeated application of the definition of contractibility, we have that $C([X_1,\ldots,X_n,\vec{Y}]) \rightarrow C([X_1,\ldots,X_n])$. It follows immediately that C is monotonic with respect to any particular D.

In the reverse direction, any valuation for $\vec{X}\vec{Y}$ can be represented by a domain D, where each $D(X_i)$ and $D(Y_i)$ is a singleton. Then, monotonicity with respect to D implies $C(\vec{X}\vec{Y}) \rightarrow C(\vec{X})$ under that valuation. If C is monotonic with respect to every domain D, then $C(\vec{X}\vec{Y}) \rightarrow C(\vec{X})$ holds under every valuation. That is, C is contractible. □

We, now turn to ways a constraint can be constructed to ensure it is contractible. As a trivial case, a constraint C of fixed arity k, when applied to a sequence of variables $\vec{X}$, is assumed to be applied only to the initial segment $X_1,\ldots,X_k$, or not at all if $\vec{X}$ is shorter than k. With this definition, C is contractible.

The $\textsc{Slide}_j$ meta-constraint (Bessiere *et al.* 2008) can be used to define several constraints on a sequence of variables. We use a variant of $\textsc{Slide}_j$ that starts applying the constraint at the pth position, rather than the first. $\textsc{Slide}_j^p(C,\vec{X})$ holds iff $C(X_{ij+p},\ldots,X_{ij+p+k-1})$ holds for $i = 0,1,\ldots,\lfloor\frac{n-p-k+1}{j}\rfloor$, where C has arity k. $\textsc{Slide}_j$ is equal to $\textsc{Slide}_j^1$.

Constraints defined directly with $\textsc{Slide}_j^p$ are contractible.

Proposition 4
Any constraint C defined by the $\textsc{Slide}_j^p$ meta-constraint as $C(\vec{X}) \leftrightarrow \textsc{Slide}_j^p(C',\vec{X})$, for some fixed arity constraint C', is contractible.

Proof
Let k be the arity of C'. The relationship between $C([X_1,\ldots,X_n,Y])$ and $C([X_1,\ldots,X_n])$ divides into cases, using the definition of C. If $n-p-k+2$ is non-negative and divisible by j, then

$$C([X_1,\ldots,X_n,Y]) \leftrightarrow C([X_1,\ldots,X_n]) \wedge C'([X_{n-k+2},\ldots,X_n,Y])$$

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

If, $n-p-k+2$ is negative or not divisible by j, then there is no additional application of C' and

$$C([X_1,\ldots,X_n,Y]) \leftrightarrow C([X_1,\ldots,X_n])$$

Thus, in both cases, $C([X_1,\ldots,X_n,Y]) \rightarrow C([X_1,\ldots,X_n])$, and hence C is contractible. □

Since the SEQUENCE and SLIDINGSUM constraints can each be defined as $\text{SLIDE}_1(C',\vec{X})$, for appropriate constraint C', it follows that they are both contractible.

For order-free constraints, we can define a meta-constraint analogous to SLIDE, which we will call SPLASH. Like SLIDE, it takes a fixed arity constraint C' and a sequence of variables $\vec{X}$ as arguments. Let C' have arity k, and $\vec{X}$ have length n, and let $S_k(\vec{X}) = \{[X_{i_1},\ldots,X_{i_k}] \mid i_j < i_{j+1} \text{ for } j = 1,\ldots,k-1\}$ be the set of subsequences of $\vec{X}$ of length k. Then, we define $\text{SPLASH}(C',\vec{X}) \leftrightarrow \bigwedge_{\vec{Y}\in S_k(\vec{X})} C'(\vec{Y})$. $\text{SPLASH}(C',\vec{X})$ applies C' to every subsequence of $\vec{X}$ of length k. For example, we can define $\text{ALLDIFFERENT}(\vec{X})$ as $\text{SPLASH}(\neq,\vec{X})$ and $\text{INTERDISTANCE}(\vec{X})$ as $\text{SPLASH}(C',\vec{X})$, where $C'(Z_1,Z_2) \leftrightarrow |Z_1 - Z_2| \geqslant p$. Thus, by the following proposition, ALLDIFFERENT and INTERDISTANCE are contractible.

Proposition 5
Any constraint C defined by the SPLASH meta-constraint as $C(\vec{X}) \leftrightarrow \text{SPLASH}(C',\vec{X})$, for some fixed arity constraint C', is contractible.

Proof
Let k be the arity of C'. It is straightforward to see that

$$C([X_1,\ldots,X_n,Y]) \leftrightarrow C([X_1,\ldots,X_n]) \wedge \bigwedge_{\vec{Z}\in S_{k-1}(\vec{X})} C'([Z_1,\ldots,Z_{k-1},Y])$$

It follows immediately from the definition that C is contractible. □

Once, we have some contractible constraints, there are many ways to build other contractible constraints, as the following proposition demonstrates. These are expressed as logic operators, but they can also be viewed as operators on formal languages: $\wedge$ and $\vee$ are intersection and union of languages, negation is complement, existential quantification projects out a variable, and universal quantification retains words that appear for all values of the relevant variable.

Proposition 6
Let $C_1(\vec{X})$ and $C_2(\vec{X})$ be contractible constraints on the same sequence of variables. Let $C(X_1,\ldots,X_k)$ be a constraint of fixed arity. Then,

- C is contractible
- $C_1 \wedge C_2$ is contractible
- $C_1 \vee C_2$ is contractible
- $\exists X_i\ C_1$ is contractible
- $\forall X_i\ C_1$ is contractible

where X_i is a variable in $\vec{X}$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Proof

We can view C as a constraint C' on the sequence $\vec{X}$, where $C'([X_1,\ldots,X_n]) \leftrightarrow true$ if $n < k$ and $C'([X_1,\ldots,X_n]) \leftrightarrow C(X_1,\ldots,X_k)$ if $n \geqslant k$. Note that $C(X_1,\ldots,X_k) \rightarrow true$ and hence $C'([X_1,\ldots,X_{k-1},Y]) \rightarrow C'([X_1,\ldots,X_{k-1}])$. When $n \neq k-1$, we clearly have $C'([X_1,\ldots,X_n,Y]) \leftrightarrow C'([X_1,\ldots,X_n])$.

Suppose, $C_i([X_1,\ldots,X_n,Y]) \rightarrow C_i([X_1,\ldots,X_n])$ for $i = 1,2$. Then, by propositional logic,

$$\bigwedge_i C_i([X_1,\ldots,X_n,Y]) \rightarrow \bigwedge_i C_i([X_1,\ldots,X_n])$$

and

$$\bigvee_i C_i([X_1,\ldots,X_n,Y]) \rightarrow \bigvee_i C_i([X_1,\ldots,X_n])$$

Similarly, using standard arguments, for any i we can conclude

$$\forall X_i\ C_1([X_1,\ldots,X_n,Y]) \rightarrow \forall X_i\ C_1([X_1,\ldots,X_n])$$

and

$$\exists X_i\ C_1([X_1,\ldots,X_n,Y]) \rightarrow \exists X_i\ C_1([X_1,\ldots,X_n])$$

□

In general, the negation of a contractible constraint and implication between two contractible constraints are not contractible. See Example 2, later.

The previous results give us an algebra for constructing complex contractible constraints, and can be used to demonstrate that some existing constraints are contractible. For example, CONTIGUITY is implemented in Maher (2002) essentially as

$$\exists \vec{L},\vec{R}\ SLIDE_3^2(C',[L_1,X_1,R_1,L_2,\ldots,X_n,R_n])$$

where C' has arity 7. Similarly, $(\vec{X} \leqslant_{lex} \vec{Y})$ is encoded in Bessiere *et al.* (2008) essentially as

$$\exists \vec{B}\ SLIDE_3(C',[B_1,X_1,Y_1,B_2,\ldots,X_n,Y_n])$$

where C' has arity 4. By the previous propositions, CONTIGUITY and $\leqslant_{lex}$ are contractible.

Similarly, we can define a weak version of GCC, where there are no lower bounds $\mathrm{GCC}(\vec{v},\vec{0},\vec{u},[X_1,\ldots,X_n])$ as $\bigwedge_{v_i \in \vec{v}} \mathrm{SPLASH}(C_i',\vec{X})$, where C_i' has arity u_i+1 and states that not all its arguments are equal to v_i. By the previous propositions, this weak form of GCC is contractible.

However, it is notable that the REGULAR constraint is not contractible, despite the implementation in terms of SLIDE outlined in Bessiere *et al.* (2008).

Example 1

Let $\mathscr{A}$ be an automaton that accepts the language $a+b^2$. Then, $\mathrm{REGULAR}(\mathscr{A},[X_1]) \rightarrow X_1 = a$ but $\mathrm{REGULAR}(\mathscr{A},[X_1,Y]) \rightarrow X_1 = b$. Thus, REGULAR is not contractible.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

The discrepancy arises because REGULAR is not constructed from the operations in the above propositions. Essentially, the implementation defines

$$\begin{aligned}&\textsc{Regular}(\mathscr{A}, [X_1, \ldots, X_n]) \leftrightarrow \\ &\quad \exists \vec{Q}\ \textsc{Slide}_2(Transition, [Q_0, X_1, Q_1, \ldots, X_n, Q_n]) \\ &\qquad \wedge\ Start(Q_0) \wedge Final(Q_n)\end{aligned}$$

where the 3-ary constraint *Transition* expresses the state transitions of $\mathscr{A}$, *Start* defines the start state(s) and *Final* defines the final state(s). It is the constraint on the final variable Q_n that leads to uncontractibility; the remainder is expressible within the algebra.

We now make a simple observation that provides a useful characterization of contractible constraints. If $\mathscr{A}$ defines a prefix-closed language, then $\textsc{Regular}(\mathscr{A}, \vec{X})$ is contractible. This claim holds more generally.

Proposition 7
Let $C(\vec{X})$ be a constraint over a sequence of variables. Then, C is contractible iff L_C is prefix-closed.

Proof
Suppose C is contractible. If σ is a solution of $C([X_1, \ldots, X_n, Y])$, then, by contractibility, the restriction of σ to $X_1 \ldots X_n$ is a solution of $C([X_1, \ldots, X_n])$. Thus, the set of solutions is prefix-closed.

Suppose S is prefix-closed. For any solution σ of $C([X_1, \ldots, X_n, Y])$, we know that the restriction of σ to $X_1 \ldots X_n$ is a solution of $C([X_1, \ldots, X_n])$. Since, this holds for any solution σ, we have $C([X_1, \ldots, X_n, Y]) \rightarrow C([X_1, \ldots, X_n])$, that is, C is contractible. □

This result applies to constraints based on formal languages, such as REGULAR and CFG, but it also applies to constraints that are formulated differently. Thus, for example, the solutions of SEQUENCE and ALLDIFFERENT are prefix-closed. Conversely, we see that constraining the final variable in a sequence, as in $Final(Q_n)$, is not contractible.

This characterization allows us to substantiate the claim, made earlier, that in general the negation or implication of contractible constraints is not contractible.

Example 2
Suppose, we have an alphabet $\{a, b\}$. If L_C is a^*, then $L_{\neg C}$ contains *aab*, but not its prefix *aa*. Hence, $\neg C$ is not contractible. Hence, also, $C \rightarrow false$ is not contractible that is, implication of contractible constraints is not, in general, contractible. To take another example, if L_{C_1} is $a^*b^*a^*$ and L_{C_2} is a^*b^*, then $L_{C_1 \rightarrow C_2}$ contains *bab* (since $bab \notin L_{C_1}$), but not its prefix *ba* (since $ba \in L_{C_1}$ but $ba \notin L_{C_2}$). Hence, $C_1 \rightarrow C_2$ is not contractible.

We can use the prefix-closed characterization both to determine, whether a constraint is contractible or not, and as the basis for approximations of uncontractible constraints. We explore these possibilities in the following sections.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

5 Classifying constraints

It is not within the scope of this paper to determine the contractibility of every global constraint. Nevertheless, we can outline and demonstrate some principles that make it easy, in most cases, to classify a global constraint as contractible or not.

In general, constraints based on counting with a lower bound (or equality) are not contractible. We can see this by noting that any non-trivial lower bound on the number of things in a sequence (or satisfied by a sequence) may be violated by a prefix of the sequence. This was already touched upon in Barták (2003), where the Sum constraint $\sum_{i=1}^{n} X_i = N$ was shown to be non-monotonic, but the argument holds for a wide range of constraints.

For example, Peak counts the number of peaks in a sequence, but a prefix of the sequence may have fewer peaks. Similarly, Stretch places lower bounds on the span of stretches, so that 1122 might be a solution, while 112 is not. By a similar argument, constraints identifying properties of an extreme element in a sequence, such as HighestPeak, are not contractible. On the other hand, NoPeak is contractible since, to the extent that there is counting, there is no lower bound – only an upper bound of 0.

We can generalize and formalize these observations. A function f is a *non-decreasing accumulation function,* if it maps sequences of values to numbers such that, for every sequence $\vec{X}$ and value Y, $f(\vec{X}Y) \geqslant f(\vec{X})$. We can similarly define the non-increasing functions. Among non-decreasing accumulation functions are counting the number of elements in a sequence with a fixed property, counting the number of different elements, identifying the highest peaks, and summing (some) non-negative elements of a sequence. Note that summing possibly negative elements of a sequence is not non-decreasing. The first part of the following proposition is an almost direct consequence of the definitions of contractibility and non-decreasing function.

Proposition 8
Let C be a global constraint.

- Suppose C can be expressed as $f(\vec{X}) \leqslant Z$. Then, C is contractible iff f is a non-decreasing accumulation function.
- Suppose C can be expressed as $f(\vec{X}) \geqslant Z$. Then, C is contractible iff f is a non-increasing accumulation function.
- Suppose C can be expressed as $f(\vec{X}) = Z$. Then, C is contractible iff f is a constant function.

Proof
If f is a non-decreasing accumulation function, whenever $f(\vec{X}Y) \leqslant Z$, we must have $f(\vec{X}) \leqslant Z$. Thus, $C(\vec{X}Y) \rightarrow C(\vec{X})$.

If f is a not a non-decreasing accumulation function, there is a sequence of values $\vec{X}$ and a value Y, such that $f(\vec{X}Y) < f(\vec{X})$. Choose Z such that $f(\vec{X}Y) \geqslant Z > f(\vec{X})$. Then, $C(\vec{X}Y)$ holds but $C(\vec{X})$ does not. Thus, C is not contractible.

The proof of the second and third parts is similar. □

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Thus, the constraints $\sum_{i=1}^{n} X_i = N$ and $\sum_{i=1}^{n} |X_i| = N$ are not contractible. Similarly, $\sum_{i=1}^{n} X_i \geqslant N$ is not contractible, while $\sum_{i=1}^{n} |X_i| \leqslant N$ is contractible. This result can be used to establish that PEAK, and HIGHESTPEAK are not contractible and that NOPEAK is contractible, but it also applies to many other counting and summing constraints in Beldiceanu *et al.* (2005).

Notice that in constraints like SEQUENCE and SLIDINGSUM, the use of a lower bound in the description of the constraint C' to which SLIDE is applied does not prevent contractibility. Each lower bound applies only to a small part of the sequence. However, the RELAXEDSLIDINGSUM constraint, which weakens the SLIDINGSUM constraint by putting bounds on the number of times the C' constraint is satisfied, is not contractible, because counting is an accumulation function that is not non-increasing and the lower bound applies to the entire sequence.

Some constraints can be recognized as contractible, based only on their informal semantics. For example, DIFFN and DISJUNCTIVE enforce that objects represented by the variables are non-overlapping. Clearly, if $\vec{X}Y$ forms a non-overlapping set, then so does $\vec{X}$ alone. Thus, contractibility follows directly from Definition 2. Similarly, CUMULATIVE[4], BINPACKING, and DISJOINT are contractible.

For other constraints, their informal semantics lead easily to counterexamples to contractibility. Constraints that involve computing the minimum, maximum, mean/average, median, mode, standard deviation, etc. of the sequence are not contractible. This is easily recognized, since these statistics are not, in general, preserved after eliminating part of the sample set, and hence are not prefix-closed. Alternatively, we could recognize that these functions are not non-increasing, nor non-decreasing and apply Proposition 8.

The idea of contractibility is not useful for all global constraints. For example, it appears irrelevant to cyclic constraints like the cyclic REGULAR, cyclic SEQUENCE, and cyclic STRETCH constraints. In these constraints, the sequence of variables is representing a cycle or circular list and there is no natural end at which to add variables. Thus, it is not surprising that these constraints are not contractible.

There is sometimes a fine line between contractible and uncontractible constraints. For example, while $\leqslant_{lex}$ is contractible, $<_{lex}$ is not. To see the latter, observe that $111 <_{lex} 112$, but the corresponding prefixes are not strictly smaller – they are equal. If the precedence constraint $s \prec_{\vec{X}} t$ also required that t appear in $\vec{X}$, then the constraint would not be contractible (because rst satisfies this constraint, but rs does not). Finally, notice that the SEQUENCE constraint is contractible, but it has the form SLIDE$(C', \vec{X})$, where C' is essentially a fixed-arity AMONG constraint; however, the (variable-arity) AMONG constraint is *not* contractible.

A quick survey of Beldiceanu *et al.* (2005) suggests that most current global constraints are not contractible, although we have noted several useful constraints that are contractible. In the next section, we address how to propagate uncontractible open constraints.

[4] Under the assumption that activities can only consume resources (and not produce resources).

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

6 Approximating constraints

When a constraint is not contractible, the closed propagator for that constraint is unsound as a propagator for the open constraint. However, following a proposal of Barták (2003), we can implement an uncontractible open constraint $C(\vec{X})$ by executing a safe contractible approximation C_{app} of C until $\vec{X}$ is closed, and then replacing C_{app} by C for the remainder of the execution. To employ this approach, we need to identify a contractible language containing the language of C, and a propagator C_{app} that implements it.

A language L is an approximation of a constraint C if $L_C \subseteq L$. An approximation L is contractible iff L is prefix-closed. A contractible approximation L_a to a language L is *tight* if for all contractible languages L'; if $L_a \supseteq L' \supseteq L$, then $L' = L_a$. By Proposition 7, there is a unique contractible approximation that is tighter than all others: the prefix-closure of L_C gives the tightest contractible approximation[5].

The prefix-closure $P(L)$ of a language L often appears to be simpler than L. For example, if L_1 is $\{a^{n^2} \mid n \in \mathbb{N}\}$, then $P(L_1)$ is a^*. But in general the prefix-closure is no simpler than the original language. For example, if L_2 is $\{a^{n^2}b \mid n \in \mathbb{N}\}$, then $P(L_2)$ is $a^* \cup L_2$. In some cases, it is easy to represent $P(L)$ when given a representation of L. In particular, when L is defined by a finite automaton the automaton accepting $P(L)$ is easily computed.

Proposition 9
Let $\mathscr{A}$ be a (possibly non-deterministic) finite state automaton, and let $\mathscr{A}'$ be the finite state automaton obtained from $\mathscr{A}$ by making final all states on a path from the start state to a final state. Then, $L(\mathscr{A}') = P(L(\mathscr{A}))$. $\mathscr{A}'$ can be computed in linear time.

Proof
Consider any prefix w of a word $wu \in L(\mathscr{A})$. wu describes a path in $\mathscr{A}$ that ends at a final state. Hence, w describes a path in $\mathscr{A}$ that ends at a state on a path to a final state. Hence, w is accepted by $\mathscr{A}'$. Thus, $L(\mathscr{A}') \supseteq P(L(\mathscr{A}))$.

Conversely, suppose w is accepted by $\mathscr{A}'$. By the construction of $\mathscr{A}'$, w describes a path in $\mathscr{A}$ that ends at a state Q on a path to a final state of $\mathscr{A}$. Let u be a word corresponding to a path from Q to a final state. Then, wu is accepted by $\mathscr{A}$ and hence w is a prefix of a word in $L(\mathscr{A})$. Thus, $L(\mathscr{A}') \subseteq P(L(\mathscr{A}))$.

We can construct $\mathscr{A}'$ as follows. Treat the automaton $\mathscr{A}$ as a directed graph with the states as vertices and where each transition from Q_1 to Q_2 is represented by an edge from Q_1 to Q_2. Perform depth-first search and mark all states reachable from the start state. Now, consider the graph with the edges reversed. Perform depth-first search from the reachable final states, marking each visited reachable state as a final state. $\mathscr{A}'$ is the automata $\mathscr{A}$ with these additional final states. The cost of the construction is $O(V + E)$, where V is the number of states, and E is the number of transitions. (Note that we could ignore reachability and define a variation of $\mathscr{A}'$ that may have some unreachable final states.) □

[5] Consequently, tight and tightest contractible approximations are synonyms.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Similarly, we can use the structure of a context-free grammar to construct a grammar for its prefix-closure.

Proposition 10
Given a context-free grammar $\mathscr{G}$ defining a language L, a context-free grammar $\mathscr{G}'$ for $P(L)$ can be generated in quadratic time, and in linear time if $\mathscr{G}$ is in Chomsky normal form.

Proof
(Sketch) We show only the construction when $\mathscr{G}$ is presented in Chomsky normal form, and leave the generalization to arbitrary grammars and the verification of its correctness to the reader.

Let $\mathscr{G} = (N, T, R, S)$, where N is a set of non-terminal symbols, T is a set of terminal symbols, R is the set of production rules, and S is the start symbol. In Chomsky normal form, production rules have the form $A \to BC$ or $A \to a$ or $S \to \varepsilon$, where A, B, and C are non-terminal symbols, a is a terminal symbol, and ε is the empty word. We define $\mathscr{G}' = (N', T, R', S')$, where $N' = N \cup \{S'\} \cup \{A_p \mid A \in N\}$ and

$$R' = R \cup \{S' \to \varepsilon\} \cup \{S' \to S_p\} \cup \{A_p \to a \mid (A \to a) \in R\} \cup \{A_p \to B_p \mid (A \to BC) \in R\} \cup \{A_p \to BC_p \mid (A \to BC) \in R\}$$

For each non-terminal $A \in N$, A_p generates all non-empty prefixes of words generated by A, including the words generated by A. It is clear that $\mathscr{G}'$ is larger than $\mathscr{G}$ by a factor of 3 or less. For an arbitrarily structured grammar, the size of $\mathscr{G}'$ can grow quadratically.

R' is not in Chomsky normal form, but it is easily simplified to that form. Non-terminals A_p, which are strongly connected by edges corresponding to productions of the form $X \to Y$ can be replaced by a single equivalent nonterminal, to give R''. Remaining productions $X \to Y$ can be replaced by a set of productions $\{X \to \psi \mid (Y \to \psi) \in R''\}$. In general, repeated replacements are necessary to eliminate all $X \to Y$ productions. A naive representation can increase the size of the grammar, but a more careful representation can share the right-hand side of productions so that the Chomsky normal form is not larger than $\mathscr{G}'$. □

Thus, the tightest contractible approximation of $\textsc{Regular}(\mathscr{A}, \vec{X}, N)$ is implemented by $\textsc{Regular}(\mathscr{A}', \vec{X}, N)$, and the tightest contractible approximation of $\text{CFG}(\mathscr{G}, \vec{X}, N)$ is implemented by $\text{CFG}(\mathscr{G}', \vec{X}, N)$.

As a corollary to Proposition 9, we can check in linear time whether a language defined by a deterministic finite automaton is prefix-closed: we simply check whether the construction of $\mathscr{A}'$ in Proposition 9 made any new final states. This improves on a result of Brzozowski *et al.* (2009). Unfortunately, recognizing when a language defined by a nondeterministic finite automaton $\mathscr{A}$ is prefix-closed is not so simple; $\mathscr{A}$ need not have the property that all states on a path from start to final state are final. It is shown in Brzozowski *et al.* (2009) that this problem is PSPACE-complete. The problem is undecidable for languages defined by context-free grammars (Brzozowski *et al.* 2009). However, the decision problem is much less important than the ability to construct (the representation of) the prefix-closure, so these negative results are not significant.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

REGULAR and CFG are complicated by flexible parameters, but approximations to simpler constraints are often correspondingly simpler to recognize. As discussed in Barták (2003), a constraint $\sum_{i=1}^{n} X_i = N$, where the X_i's must be non-negative is not monotonic but is approximated by the constraint $\sum_{i=1}^{n} X_i \leqslant N$. Using Proposition 8, we can recognize this as the tightest contractible approximation. Similarly, for a counting constraint such as PEAK($\vec{X}, N$), which states that there are exactly N peaks in $\vec{X}$, the tightest contractible approximation states that N is an upper bound on the number of peaks. In the same way, NVALUE($\vec{X}, N$) is best approximated by treating N only as an upper bound. The tightest approximation of the GCC is the weak form of GCC discussed in Section 4. In all these cases, since counting is a non-decreasing accumulation function, the tightest contractible approximation is to eliminate the lower bounds. In HIGHESTPEAK($\vec{X}, Z$), the height of the highest peak is a non-decreasing accumulation function and so the tightest approximation states that Z is an upper bound on the height of the highest peak.

On the other hand, for some constraints where the accumulation function is neither non-increasing nor non-decreasing there appear to be no non-trivial approximations. For example, consider a constraint AVERAGE($\vec{X}, M$) stating that M is the mean/average of the values of $\vec{X}$. Given a fixed M, *any* sequence of values can be a prefix of a sequence with mean M. Hence, the tightest contractible approximation of AVERAGE is the constraint that accepts any sequence, that is, the constraint *true*. For such a constraint, there is no propagation until the constraint is closed.

However, as the previous discussion shows, for many constraints the tightest contractible approximation is not only non-trivial, it has a clear and simple expression. For these constraints, a propagator for the approximation C_{app} is almost ready-made, given a propagator for the original constraint C. Furthermore, the transition of propagator from C_{app} to C when the constraint closes can be smooth and simple because, in the cases above, the propagator for C_{app} is simply a weakened form of the propagator for C. Some more detailed analysis of this similarity of propagators for C and C_{app}, for several constraints C, appears in Maher (2009c) and (for a slightly different model of open constraint) (Maher 2009b).

If we have domain consistent closed propagators and a tight contractible approximation, then we can obtain an open D-consistent propagator from Barták's proposal. Recall that under Barták's proposal (Barták 2003), a closed propagator for C_{app} is dynamized to handle extensions of the sequence of variables (possibly through his generic dynamization). This propagator is then executed until the sequence of variables is closed, at which point the propagator is replaced by a closed propagator for C.

Theorem 11

Let C_{app} be the tightest contractible approximation to C, and suppose we have closed propagators for C_{app} and C that maintain domain consistency for $\vec{X}$. Then, Barták's proposal maintains open D-consistency for C.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Proof
Since C_{app} is contractible, domain consistency of C_{app} for $\vec{X}$ is equivalent to open D-consistency on $C(\vec{X})$. This follows because C_{app} is the prefix-closure of C and so every support for domain consistency of $C_{app}(\vec{X})$ for $\vec{X}$ corresponds to a longer word that is a support for D-consistency on $C(\vec{X})$, and *vice versa* every support for open D-consistency on $C(\vec{X})$ has a corresponding prefix that is a support for domain consistency of $C_{app}(\vec{X})$ for $\vec{X}$. Once $\vec{X}$ is closed, domain consistency for $\vec{X}$ is identical to D-consistency on $C(\vec{X})$. □

We can obtain similar results for consistency conditions other than domain consistency. All that is required is to define the appropriate corresponding open consistency. For example, consider bounds consistency. Let $min(X)$ ($max(X)$) denote the smallest (largest) value in $D(X)$. The appropriate form of bounds consistency for open constraints is open B-consistency.

Definition 4
Given a domain D, an occurrence of a constraint $C(\vec{X})$ is *open B-consistent*, if
for every $X_i \in \vec{X}$, and for $d_i = min(X_i)$ and $d_i = max(X_i)$, there is a word $d_1 \ldots d_m$ in L_C, such that $|\vec{X}| \leqslant m$, and $d_j \in min(X_i)..max(X_i)$ for $j = 1, \ldots, |\vec{X}|$.

We can now express the corresponding result for bounds consistency. The proof is essentially the same as that for the previous theorem.

Corollary 12
Let C_{app} be the tightest contractible approximation to C, and suppose we have closed propagators for C_{app} and C that maintain bounds consistency for $\vec{X}$. Then, Barták's proposal maintains open B-consistency for C.

Notice that we still require a tightest contractible approximation. Any weakening of this requirement can lose open B-consistency, as is clear from Corollary 2.

7 Contractibility of soft constraints

We consider "soft" global constraints in the style of Petit *et al.* (2001). In such constraints, there is a violation measure[6], which measures the degree to which an assignment to the variables violates the associated "hard" constraint, and solutions are assignments that satisfy an upper bound on the violation measure. Thus, such soft constraints have the form $m(\vec{X}) \leqslant Z$, where m is the violation measure. We refer to the hard constraint as $C(\vec{X})$, and the corresponding soft constraint as $C_s(\vec{X}, Z)$.

Assessing the contractability of such constraints is made easier by Proposition 8, which says that a constraint $m(\vec{X}) \leqslant Z$ is contractible iff m is non-decreasing. Given this characterization, we will refer to non-decreasing accumulation functions as contractible functions. To evaluate whether or not soft constraints are contractible, we must consider the form of the violation measure, and whether it forms a contractible function.

[6] Also called violation cost (Petit *et al.* 2001).

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Definition 5
A *violation measure* for a sublanguage L of a language L' is a function m, which maps L' to the non-negative real numbers, such that if $w \in L$, then $m(w) = 0$. m is *proper* for L if for all words $w \in L'$, $m(w) = 0$ iff $w \in L$. A violation measure for a constraint $C(\vec{X})$ is a violation measure for L_C as a sublanguage of the static type $T(\vec{X})$.

For example, a use of AllDifferent might give the set $\mathbb{Z}$ of integers as the static type of each variable. A violation measure might then be the number of disequalities $X_i \neq X_j, i \neq j$ violated by a valuation for $\vec{X}$, or the number of variables equal to another variable under the valuation, or the minimum absolute value of the sum over i of values c_i such that, for each i and j with $i \neq j$, $X_i + c_i \neq X_j + c_j$[7]. It is easy to see that each of these defines a violation measure. The third is not a proper violation measure because, for example, the word $11,233$ can have perturbations c_i of $0, -1, 0, 0, 1$. Thus, $m(11233) = 0$ but $11233 \notin L_C$. (Summing the absolute value of the c_i, on the other hand, would lead to a proper measure.)

Proper violation measures for a language L are a refinement of the characteristic function of L[8]. Most violation measures in the literature are proper for their intended language. Although any function from words to non-negative reals can be considered a proper violation measure by appropriate choice of language L, in practice the hard constraint determines L and the violation measure is then designed to be proper. A non-proper measure can be considered misleading because a word w that violates the language L can have a violation measure of 0. We admit non-proper violation measures mainly because contractible approximations considered in Section 8 can be non-proper. However, we make some effort in this section to identify proper violation measures.

There are three broad classes of violation measures (Maher 2009d): those based on constraint decomposition, edit distance, and graph properties. We address the first two classes in the following subsections. The richness of the graph property framework (Beldiceanu and Petit 2004) makes it difficult to obtain broad results on contractibility. A somewhat narrow sufficient condition for contractability of soft constraints defined by graph property-based violation measures is presented in Maher (2009d). For each of the classes we consider, we will incorporate a weighting that adds greater flexibility and expressiveness to the class.

7.1 Decomposition-based violation measures

Many hard constraints can be decomposed into elementary constraints, whether naturally (such as the decomposition of AllDifferent into disequalities) or by a construction, as in Bessiere *et al.* (2009). Violation measures can be constructed by combining the violations of each elementary constraint. We define a general class of

[7] This latter measure expresses the smallest perturbation $\vec{c}$ of the values for the variables needed to satisfy the AllDifferent constraint. More formally, $m(\vec{X}) = \min_{\vec{c}}\{|\sum_{i=1}^{n} c_i| \mid \forall j\ j \neq i \rightarrow X_i + c_i \neq X_j + c_j\}$.
[8] Indeed, for any proper violation measure m, the corresponding hard constraint can be recovered as $m(\vec{X}) \leqslant 0$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

decomposition-based violation measures that includes as special cases: primal graph based violation costs (Petit *et al.* 2001), decomposition-based violation measures of van Hoeve *et al.* (2006), the value-based violation measure for GCC (Petit *et al.* 2001; van Hoeve *et al.* 2006), the measures used for the soft SEQUENCE constraint (Maher *et al.* 2008), and the soft CUMULATIVE constraint (Petit and Poder 2009), the weighted measures for ALLDIFFERENT and GCC (Métivier *et al.* 2007, 2009), and the class of decomposition-based measures discussed in Maher (2009d). We begin with several definitions.

A *weighted set* is a pair (S, w), where S is a set and w is a function mapping each element of S to a non-negative real number or ∞. Values not in S have weight 0. If these are the only values of weight 0 we say (S, w) is *proper*. A weighted set is a minor generalization of a multiset. A weighted set (S_1, w_1) is a *sub-weighted set* of weighted set (S_2, w_2) if, for every element $s \in S_1$, $w_1(s) \leqslant w_2(s)$. Union of weighted sets is defined by $(S_1, w_1) \cup (S_2, w_2) = (S_1 \cup S_2, w_1 + w_2)$, where $(w_1 + w_2)(x) = w_1(x) + w_2(x)$. When a weighted set contains things with variables that are subject to substitution, the application of a substitution might unify elements of the set. Hence, $(S, w)\theta$ denotes $(S\theta, w')$ where $w'(s)$ is the sum of $w_1(s')$ over all $s' \in S$ such that $s'\theta \equiv s$.

We need to carefully formalize the notion of decomposition. The definition takes as a parameter a class of elementary constraints. Usually the constraints in such a class have bounded arity.

Definition 6

A *decomposition* is a function that maps a constraint C with a given type T and a sequence of variables $\vec{X}$ to a tuple $(\vec{X}, \vec{U}, T', S, w)$, where $\vec{U}$ is a collection of new variables, T' is an extension of T to $\vec{U}$, and (S, w) is a proper weighted set of elementary constraints over $\vec{X}\vec{U}$, such that $C(\vec{X}) \leftrightarrow \exists \vec{U}\ T'(\vec{U}) \wedge \bigwedge_{s \in S} s$.

The weights in this definition are used only to emphasize some constraints in a decomposition over others; in particular, the infinite weight allows us to specify elementary constraints that must not be violated. An *unweighted decomposition* is one where all constraints in S have the same, non-zero weight. In that case, we may omit w. We write DECOMP($C(\vec{X})$) to express the weighted set (S, w), or simply S when the decomposition is unweighted.

This definition of decomposition is very broad, perhaps too broad, since it allows the set of elementary constraints and/or their weights to vary radically as the length of $\vec{X}$ changes. For example, it permits using the decomposition of ALLDIFFERENT($\vec{X}$) into disequalities when $|\vec{X}|$ is odd, and a decomposition from Bessiere *et al.* (2009) (see Example 6) when $|\vec{X}|$ is even. However, we will see in Example 5 a constraint whose expression requires some of the flexibility offered by this broad definition.

An *error function* e maps an elementary constraint and a valuation to a non-negative real number, representing the amount of error (or violation) of the constraint by the valuation. We require that $e(v, c) = 0$ iff c is satisfied by v. We extend e to weighted sets of constraints by defining $e(v, (S, w)) = (S', w')$, where $S' = \{e(v, s) \mid s \in S\}$ and $w'(x) = \sum_{s|v(s)=x} w(s)$.

A *combining function* maps a weighted set of numbers to a single number. A combining function *comb* is *monotonic* if, whenever (S_1, w_1) is a sub-weighted set

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

of (S_2, w_2), $comb(S_1, w_1) \leqslant comb(S_2, w_2)$. The function *comb* is *disjunctive* if for all weighted sets of reals (S, w), $comb(S, w) = 0$ iff $S = \{0\}$. We say *comb has unit 0* if, for every (S, w) and w', $comb((S, w) \cup (\{0\}, w')) = comb(S, w)$. Counting non-zero values, summation, sum of squares, and maximization are examples of monotonic, disjunctive combining functions with unit 0; product and minimization are neither monotonic nor disjunctive nor have unit 0.

Definition 7
A *decomposition-based violation measure m* for a constraint $C(\vec{X})$ with type T is based on a decomposition $(\vec{X}, \vec{U}, T', S, w)$ of $C(\vec{X})$, an error function e, and a combining function *comb* and is defined by, for each valuation v of $\vec{X}$,

$$m(v(\vec{X})) = \min_{v'} comb(e(v', \text{DECOMP}(C(\vec{X}))))$$

where, we minimize over all extensions v' of v to $\vec{U}$ that satisfy T'.

This definition was inspired by the formulation of hierarchical constraints in Borning *et al.* (1992, 1989). The violation counting decomposition measures of Petit *et al.* (2001); van Hoeve *et al.* (2006) can be obtained when the error function $e(v, c)$ returns 0 if v satisfies c and 1 otherwise, and the combining function is summation. The value-based measures of Petit *et al.* (2001), van Hoeve *et al.* (2006), Maher *et al.* (2008), Petit and Poder (2009) also use summation as the combining function, but use an error function that returns the amount by which the constraint c is violated by the valuation v. If we use maximization or the sum of squares in place of summation, we have new violation measures similar to the *worst-case-better* and *least-squares-better* comparators of Borning *et al.* (1989, 1992). Clearly, many violation measures are available for a constraint by making different choices for the decomposition and the error and combining functions.

There is a powerful sufficient condition for a decomposition-based violation measure to be proper.

Proposition 13
Let m be a decomposition-based violation measure for a constraint C, as defined in Definition 7 with combining function *comb*. m is proper for L_C if *comb* is disjunctive.

Proof
Let v be a valuation for $\vec{X}$. Suppose *comb* is disjunctive.
$m(v(\vec{X})) = 0$
iff $\min_{v'} comb(e(v', \text{DECOMP}(C(\vec{X})))) = 0$
iff for some v' extending v, $comb(e(v', \text{DECOMP}(C(\vec{X})))) = 0$
iff for some v' extending v, and some w, $e(v', \text{DECOMP}(C(\vec{X}))) = (\{0\}, w)$
iff for some v' extending v, v' satisfies every $c \in \text{DECOMP}(C(\vec{X}))$
iff v satisfies $C(\vec{X})$
iff $v(\vec{X}) \in L_C$.

Thus, for any valuation v, $m(v(\vec{X})) = 0$ iff $v(\vec{X}) \in L_C$. Hence, m is proper for L_C. □

We now turn to the problem of recognizing contractibility. We say that one formula $(\vec{X}, \vec{U}, T_1, S_1, w_1)$ is *covered* by another formula $(\vec{W}, \vec{V}, T_2, S_2, w_2)$, if there is

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

a substitution θ that maps $\vec{X}$ into $\vec{W}$ and $\vec{U}$ into $\vec{V} \cup \vec{W} \cup \Sigma$, where Σ is a set of constants, such that $T_1(\vec{X}) = T_2(\vec{X}\theta)$, $(S_1, w_1)\theta$ is a sub-weighted set of (S_2, w_2) and $T_2(\vec{U}\theta) \subseteq T_1(\vec{U})$. Covering has some similarity to characterizations of containment of conjunctive relational database queries (Chandra and Merlin 1977), (constraint) logic programming rule subsumption (Maher 1988, 1993), and sufficient conditions for query containment under bag semantics (Chaudhuri and Vardi 1993; Ioannidis and Ramakrishnan 1995).

Example 3
The decomposition of AllDifferent($\vec{X}$) into an unweighted set of disequalities is formalized as $(\vec{X}, \emptyset, T, S, w)$, where S is the set of disequalities and w gives every disequality a weight of 1. It is clear that the decomposition of AllDifferent($\vec{X}$) is covered by that of AllDifferent($\vec{X}Y$) where the substitution is the identity.

Example 4
Contiguity is implemented in Maher (2002) essentially by the decomposition

$$\text{Contiguity}(\vec{X}) \leftrightarrow \exists \vec{L}, \vec{R} \bigwedge_{i=2}^{n-1} C'(X_{i-1}, R_{i-1}, L_i, X_i, R_i, L_{i+1}, X_{i+1})$$

for a constraint C'. This decomposition is formalized as $(\vec{X}, \vec{L}\vec{R}, T, S, w)$, where T gives all variables a type of $\{0, 1\}$, S is the set of C' constraints, and w gives every constraint a weight of 1. Alternatively, if contiguity is more important for variables nearer the right end of the sequence $\vec{X}$, we might weight each C' constraint by the largest index of a variable appearing in it. The decomposition of Contiguity($\vec{X}Y$) covers that of Contiguity($\vec{X}$), where the substitution is the identity on $\vec{X}$, $\vec{L}$, and $\vec{R}$.

We can now provide a sufficient condition for a soft constraint with a decomposition-based violation measure to be contractible.

Proposition 14
Let C_s be a soft constraint with a decomposition-based violation measure defined using a monotonic combining function. Let $(\vec{X}, \vec{U}, T_1, S_1, w_1)$ be the decomposition of $C(\vec{X})$ and $(\vec{X}Y, \vec{V}, T_2, S_2, w_2)$ be the decomposition of $C(\vec{X}Y)$. If $(\vec{X}, \vec{U}, T_1, S_1, w_1)$ is covered by $(\vec{X}Y, \vec{V}, T_2, S_2, w_2)$ via a substitution that is the identity on $\vec{X}$, then C_s is contractible.

Proof
By the covering condition, there is a substitution θ that is the identity on $\vec{X}$ and maps $\vec{U}$ to $\vec{V} \cup \vec{X}Y \cup \Sigma$ such that $(S_1, w_1)\theta$ is a sub-weighted set of (S_2, w_2). Consider any assignment v to $\vec{X}Y \cup \vec{V}$. Then, $v \circ \theta$ is an assignment[9] to $\vec{X} \cup \vec{U}$. Furthermore, $v((S_1, w_1)\theta)$ is a sub-weighted set of $v(S_2, w_2)$ and hence $e(v \circ \theta, (S_1, w_1)) = e(v, (S_1, w_1)\theta)$ is a sub-weighted set of $e(v, (S_2, w_2))$. Consequently, since the combining function *comb* is monotonic, $comb(e(v \circ \theta, (S_1, w_1))) \leqslant comb(e(v, (S_2, w_2)))$. It follows that $m(v(\vec{X})) \leqslant m(v(\vec{X}Y))$. Thus, since v is arbitrary, m is non-decreasing and, by Proposition 8, C_s is contractible. □

[9] We define $(v \circ \theta)(x) = v(x\theta)$ for any term x.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

It follows that the constraints in Examples 3 and 4 are contractible. More generally, if an unweighted decomposition is defined via part of the algebra discussed in Section 4 (that is, using SLIDE or SPLASH meta-constraints, constraints on a fixed finite prefix of the variable sequence, conjunction, and existential quantification) and a monotonic combining function, then Proposition 14 is sufficient to establish contractibility. However, covering is not a necessary condition for contractibility, as the following example demonstrates.

Example 5

Consider the definition of a rising sawtooth relation *rs* on variables $\vec{X}$. In such a relation, the subsequence of values in even numbered positions forms a non-decreasing sequence, and every value in odd numbered positions is greater than or equal to its immediately adjacent neighbours[10]. This relation can be decomposed into elementary constraints as follows. The decomposition is defined recursively, but notably requires two recursive cases, corresponding to the distinction between odd and even length sequences.

$$\begin{aligned}
&\textsc{decomp}(rs([])) &&= \mathit{true}\\
&\textsc{decomp}(rs([X_1])) &&= \mathit{true}\\
&\textsc{decomp}(rs([X_1, X_2])) &&= X_1 \geqslant X_2\\
&\textsc{decomp}(rs([X_1, \ldots, X_{2n}, X_{2n+1}])) &&=\\
&\qquad \textsc{decomp}(rs([X_1, \ldots, X_{2n}])) \wedge X_{2n+1} \geqslant X_{2n}\\
&\textsc{decomp}(rs([X_1, \ldots, X_{2n}, X_{2n+1}, X_{2n+2}])) =\\
&\qquad \textsc{decomp}(rs([X_1, \ldots, X_{2n}])) \wedge X_{2n+1} \geqslant X_{2n+2} \wedge X_{2n+2} \geqslant X_{2n}
\end{aligned}$$

Consider the soft constraint derived from this decomposition by counting the number of violations. It is clear that the sufficient condition of Proposition 14 does not apply because there is no covering. Nevertheless, we can verify that a decomposition-based soft *rs* constraint is contractible. Note first that when $\vec{X}$ has even length $\textsc{decomp}(rs(\vec{X})) \subseteq \textsc{decomp}(rs(\vec{X}Y))$ and consequently the violation measure is non-decreasing in this case. When $\vec{X}$ has odd length the relationship is less obvious. However, we know that

$$(X_{2n+1} \geqslant X_{2n+2}) \wedge (X_{2n+2} \geqslant X_{2n}) \rightarrow (X_{2n+1} \geqslant X_{2n})$$

and its contrapositive

$$\neg(X_{2n+1} \geqslant X_{2n}) \rightarrow \neg(X_{2n+1} \geqslant X_{2n+2}) \vee \neg(X_{2n+2} \geqslant X_{2n})$$

Hence, any valuation for the variables that gives rise to a violation of $X_{2n+1} \geqslant X_{2n}$ will also give rise to a violation of $X_{2n+1} \geqslant X_{2n+2}$, or $X_{2n+2} \geqslant X_{2n}$, or both. Thus, the violation measure is non-decreasing in this case also. Since the violation measure is non-decreasing, the decomposition-based soft *rs* constraint is contractible.

Similarly, the violation measures derived from summing the amount of violation or taking the maximum amount of violation of any elementary constraint lead to contractible soft *rs* constraints.

[10] This is an artificial constraint, designed to demonstrate the point. However, the pricing of goods with volume discounts can have a similar rising sawtooth behaviour.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

This example demonstrates a major limitation of the sufficient condition in Proposition 14: it addresses only the syntactic structure of the decomposition. However, some constraints, such as *rs*, require reasoning about the semantics of the elementary constraints in order to recognize that the decomposition-based soft constraint is contractible. (For *rs* we exploited the knowledge that $\geqslant$ forms a total order.)

A second example is given by a decomposition of AllDifferent given in Bessiere *et al.* (2009).

Example 6

Consider the AllDifferent constraint with type T that maps each X_i to $1..d$, which we denote by AllDifferent_T. To define the decomposition we need to introduce variables A_{ilu} of type $\{0,1\}$ and constraints as follows.

For $1 \leqslant i \leqslant n$ and $1 \leqslant l \leqslant u \leqslant d$ we have the constraints

$$A_{ilu} = 1 \leftrightarrow X_i \in [l,u] \tag{7.1}$$

$$\sum_{i=1}^{n} A_{ilu} \leqslant u - l + 1 \tag{7.2}$$

This decomposition is formalized as $(\vec{X}, \vec{A}, T', S, w)$, where T' extends T to the A_{ilu} variables, S consists of the constraints (7.1) and (7.2) and w gives all constraints the same weight. It is easy to establish that $\text{AllDifferent}_T(\vec{X}) \leftrightarrow \exists \vec{A} \in T'(\vec{A})$ (7.1) $\wedge$ (7.2).

When $\vec{X}$ is extended by Y, the decomposition contains extra variables $A_{(n+1)lu}$, extra constraints of type (7.1) involving Y and the new variables, and replaces constraints (7.2) by

$$\sum_{i=1}^{n+1} A_{ilu} \leqslant u - l + 1 \tag{7.3}$$

Now, for each l and u, (7.3) $\wedge$ $(0 \leqslant A_{(n+1)lu} \leqslant 1) \rightarrow$ (7.2). Thus, every valuation that violates (7.2) will also violate (7.3). It follows that the soft constraint based on counting violations in this decomposition of AllDifferent is contractible. Similarly, soft constraints based on summing violation amounts or taking the maximum are also contractible, because $\sum_{i=1}^{n+1} A_{ilu} \geqslant \sum_{i=1}^{n} A_{ilu}$.

On the other hand, the decomposition of $\text{AllDifferent}(\vec{X}\,Y)$ cannot be a covering of the decomposition of $\text{AllDifferent}(\vec{X})$, because each constraint (7.2) is not covered by the corresponding constraint (7.3). Thus, again, the sufficient condition of Proposition 14 cannot be used.

To redress the weakness of covering in addressing Examples 5 and 6, we need to incorporate knowledge of the semantics of the elementary constraints and, more generally, the error function. We begin with some definitions.

A *division* of a weighted set (S,w) is a collection of sub-weighted sets (S_i, w_i), such that $\cup_i (S_i, w_i) = (S, w)$. When all S_i are singleton sets, we refer to this as *division into singletons*. Given a weighted set (S,w), we write $w\theta$ to denote the weight function of $(S,w)\theta$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Definition 8
A *semantic embedding* of $(\vec{X}, \vec{U}, T_1, S, w)$ in $(\vec{X}Y, \vec{V}, T_2, S', w')$ is a pair $\langle \phi, \theta \rangle$, where ϕ is a function and θ is a substitution, such that

- θ is the identity on $\vec{X}$ and maps $\vec{U}$ into $\vec{X}Y \cup \vec{V} \cup \Sigma$, where Σ is a set of constants, such that $T_2(\vec{U}\theta) \subseteq T_1(\vec{U})$;
- ϕ is an injective function from $(S, w)\theta$ to a division of (S', w'); and
- for every valuation v and every elementary constraint $c \in S\theta$, $e(v, (\{c\}, w\theta)) \leqslant comb(e(v, \phi(c)))$.

In a semantic embedding, the substitution θ shows how variables local to the first decomposition are represented in the second and the function ϕ shows how elementary constraints in the first decomposition are represented in the second. The third condition requires that these representations respect the semantics expressed by the error function e.

Covering is essentially a syntactic form of semantic embedding: a semantic embedding, where (S', w') is divided into singletons and any constraint $c\theta$ in $S\theta$ is mapped to $c\theta$ in S'.

We are now in a position to state a much broader sufficient condition for contractibility than Proposition 14.

Theorem 15
Let C_s be a soft constraint with a decomposition-based violation measure m defined using a monotonic combining function *comb*. Let $(\vec{X}, \vec{U}, T_1, S_1, w_1)$ be the decomposition of $C(\vec{X})$ and $(\vec{X}Y, \vec{V}, T_2, S_2, w_2)$ be the decomposition of $C(\vec{X}Y)$. Suppose there is a semantic embedding of (S_1, w_1) in (S_2, w_2). Then, C_s is contractible.

Proof
Consider the extension of $\vec{X}$ to $\vec{X}Y$ and a valuation v on $\vec{X}Y\vec{V}$. Let $\langle \phi, \theta \rangle$ be the semantic embedding. Then, for every elementary constraint $c \in S_1\theta$, $e(v, (\{c\}, w_1\theta)) \leqslant comb(e(v, \phi(c)))$. Hence, $e(v \circ \theta, (S_1, w_1)) = e(v, (S_1, w_1)\theta) = comb(\bigcup_{c \in S_1\theta} e(v, (\{c\}, w_1\theta))) \leqslant comb(\bigcup_{c \in S_1\theta} e(v, \phi(c))) \leqslant comb(e(v, (S_2, w_2)))$.

Since *comb* is monotonic, $comb(e(v \circ \theta, (S_1, w_1))) \leqslant comb(e(v, (S_2, w_2)))$. It follows that $\min_v comb(e(v, (S_1, w_1))) \leqslant \min_v comb(e(v, (S_2, w_2)))$, and hence $m(C(\vec{X})) \leqslant m(C(\vec{X}Y))$. Thus, C_s is contractible. □

For (unweighted) violation counting measures, the third condition of semantic embedding reduces to $D \models (T_2(\vec{V}) \wedge \phi(c)) \rightarrow c\theta$, where D expresses some properties of the elementary constraints. Thus, for these measures, we can reason about contractibility using conventional logic. In Example 5, θ can be the identity substitution, since no additional variables are used in the decomposition, and ϕ maps $(X_{2n+1} \geqslant X_{2n})$ to $(X_{2n+1} \geqslant X_{2n+2}) \wedge (X_{2n+2} \geqslant X_{2n})$. We know that $(X_{2n+1} \geqslant X_{2n+2}) \wedge (X_{2n+2} \geqslant X_{2n}) \rightarrow (X_{2n+1} \geqslant X_{2n})$ so, applying the previous theorem, a violation counting soft constraint of *rs* is contractible. In Example 6, using the natural choice of ϕ and θ (which maps variables A_{ilu} in DECOMP($C(\vec{X})$) to variables of the same name in DECOMP($C(\vec{X}Y)$), constraints (7.1) to themselves, and constraints (7.2)–(7.3)), the validity of (7.3) $\wedge$ ($0 \leqslant A_{(n+1)lu} \leqslant 1$) $\rightarrow$ (7.2), and the previous

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

theorem, we establish that the violation counting soft version of ALLDIFFERENT based on this decomposition is contractible.

There are two possible generalizations of the notion of semantic embedding that might be used to create a broader sufficient condition for contractibility. The first is to change the domain of ϕ from $(S, w)\theta$ to an arbitrary division of $(S, w)\theta$. The current definition essentially only applies to the division of $(S, w)\theta$ into singletons $\{c\}$. This generalization would allow the embedding to hold for some grouping of constraints in the first decomposition, even when the individual constraints cannot be embedded in the second. A second possible generalization is to employ multiple pairs $\langle \phi, \theta \rangle$ with a disjunctive condition. Such a generalization has been shown necessary to characterize conjunctive query containment/rule subsumption when queries/rules involve pre-defined relations (i.e. constraints) (Klug 1988; Maher 1993). These generalizations are left for future research.

7.2 *Edit-based violation measures*

The *edit-based* violation measures use a notion of edit distance, which is the minimum number of edit operations required to transform a word into a word of L_C. There are many possible edit operations but the common ones are: to substitute one letter for another, to insert a letter, to delete a letter, and to transpose two adjacent letters[11]. This class includes the *variable-based* violation measures (Petit *et al.* 2001; van Hoeve *et al.* 2006), since such measures are simply edit distances, where substitution is the only edit operation. The *object-based* measures of Beldiceanu and Petit (2004) are edit distances, where deletion is the only edit operation. In van Hoeve *et al.* (2006), an edit-based measure involving substitution, insertion, and deletion is used.

To address a wide range of edit-based measures, we generalize the measures. We allow non-negative weights $\alpha, \beta, \gamma, \delta$ for the edit operations substitution, insertion, deletion, and transposition, respectively, and let n_s, n_i, n_d, n_t be the number of the respective operations used in an edit. Then, we define $m_L(w) = \min_{edits} \alpha n_s + \beta n_i + \gamma n_d + \delta n_t$ to be the minimum, over all edits that transform w to an element of $P(L)$, of the weighted sum of the edit operations. We refer to all measures of this form as *edit-based*. Measures based on a subset of the four edit operation can be captured by giving effectively infinite weights to the other operations.

The edit-based violation measures used for closed constraints are not appropriate for open constraints, because they fail to take into account that the current sequence of variables may be extended with more variables.

For example, consider an open constraint C, where $L_C = abc + defghi$ and an occurrence of the constraint $C([X_1, X_2, X_3])$. If $X_1 = d$, $X_2 = e$ and $X_3 = f$, then the unweighted edit distance of this instance to L_C is 3, even though this instance is completely accurate if the sequence of variables is extended. Similarly, if $L_C = abc$

[11] Edit distance based on counting these operations is known as Damerau–Levenshtein distance. Other well-known edit distances are defined using a subset of these operations.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

and we have an occurrence $C([X_1, X_2])$ with $X_1 = a$ and $X_2 = b$, then the unweighted edit distance is 1, even though there is no violation.

To take account of the possibility that a sequence of variables may be extended, we employ the edit distance to $P(L_C)$, the prefix-closure of L_C. In Section 6, the prefix-closure was used to approximate a constraint so that constraint propagation is sound when the constraint is open. The use of the prefix closure here is somewhat different from its use in that section: rather than using $P(L_C)$ as an approximation to L_C, $P(L_C)$ is used here to formulate what it means to be an (edit-based) open soft constraint.

Definition 9
An *open edit-based violation measure* for a language L is an edit-based violation measure $m_{P(L)}$ for $P(L)$. An open edit-based violation measure m for L is *proper* if $m(w) = 0$ iff $w \in P(L)$. Since, in this paper, we only consider open edit-based measures they will simply be referred to as edit-based violation measures, except in the statement of theorems.

As a result of this definition, prefix-equivalent languages have the same possible edit-based (proper) violation measures. When L is clear from the context, we simply write m rather than m_L.

We can characterize when an open edit-based violation measure is proper. Roughly, m is improper iff some edits have zero cost and these are able to edit some $w \in L' \backslash P(L)$ to $w' \in P(L)$.

Proposition 16
Let m be an open edit-based violation measure for L, where $P(L)$ is a sublanguage of L', with weights α, β, γ, and δ.

m is proper iff one of the following conditions holds:

- $\min\{\alpha, \beta, \gamma, \delta\} > 0$
- $\alpha = 0$, $\min\{\beta, \gamma\} > 0$ and $L' \cap SameLength(P(L)) \subseteq P(L)$
- $\beta = 0$, $\min\{\alpha, \gamma, \delta\} > 0$ and $L' \cap SubSeq(P(L)) \subseteq P(L)$
- $\gamma = 0$ and $L' \subseteq P(L)$
- $\delta = 0$, $\min\{\alpha, \beta, \gamma\} > 0$ and $L' \cap Perm(P(L)) \subseteq P(L)$
- $\alpha = \beta = 0$, $\gamma > 0$ and $L' \subseteq Shorter(P(L))$
- $\beta = \delta = 0$, $\min\{\alpha, \gamma\} > 0$ and $L' \cap Subset(P(L)) \subseteq P(L)$

where, for any language L,
$SameLength(L)$ is the set of all words of the same length as a word of L,
$Shorter(L)$ is the set of all words the same length or shorter than a word of L,
$Perm(L)$ is the set of all permutations of words of L,
$SubSeq(L)$ is the set of all subsequences of a word of L, and
$Subset(L)$ is set of all words whose letters form a submultiset of the letters of a word of L.

Proof
Looking at the different constraints on the weights, it is easy to see that the conditions are mutually exclusive and they cover all possible combinations of weights. Thus to

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

prove the characterization, it is sufficient to show, in each case, that m is proper iff the remaining condition in the case holds.

If $\min\{\alpha, \beta, \gamma, \delta\} > 0$, then $m(w) = 0$ iff no edits are required to transform w to a word of $P(L)$ iff $w \in P(L)$. Thus, in this case, m is proper.

Let $\alpha = 0$, and $\min\{\beta, \gamma\} > 0$. Then, for any word $w \in L'$, $m(w) = 0$ iff w can be edited by substitutions (and possibly transpositions if $\delta = 0$) to a word of $P(L)$ iff w is the same length as a word of $P(L)$. From the definition of proper, m is proper iff $P(L) \cap L' = SameLength(P(L)) \cap L'$, that is $L' \cap SameLength(P(L)) \subseteq P(L)$.

Let $\beta = 0$ and $\min\{\alpha, \gamma, \delta\} > 0$. Then, for any word $w \in L'$, $m(w) = 0$ iff w can be edited by insertions to a word of $P(L)$ iff w is a subsequence of a word of $P(L)$. Hence, m is proper iff $P(L) \cap L' = Subseq(P(L)) \cap L'$.

If $\gamma = 0$, then for every word $w \in L'$, $m(w) = 0$ because w can be edited by deletions to the empty word, which is in $P(L)$. Hence, m is proper iff $L' = P(L) \cap L'$, that is $L' \subseteq P(L)$.

Let $\delta = 0$ and $\min\{\alpha, \beta, \gamma\} > 0$. Then, for any word $w \in L'$, $m(w) = 0$ iff w can be edited by transpositions to a word of $P(L)$ iff w is a permutation of a word of $P(L)$. Hence, m is proper iff $P(L) \cap L' = Perm(P(L)) \cap L'$.

Let $\alpha = \beta = 0$ and $\gamma > 0$. Then, for any word $w \in L'$, $m(w) = 0$ iff w can be edited by insertions and substitutions to a word of $P(L)$ iff w can be obtained by deletions and substitutions from a word of $P(L)$ iff w is shorter than a word of $P(L)$. Hence, m is proper iff $P(L) \cap L' = Shorter(P(L)) \cap L'$.

Let $\beta = \delta = 0$ and $\min\{\alpha, \gamma\} > 0$. Then, for any word $w \in L'$, $m(w) = 0$ iff w can be edited by insertions and transpositions to a word of $P(L)$ iff w can be obtained by deletions and transpositions from a word of $P(L)$ iff the letters of w form a submultiset of the letters of a word of $P(L)$. Hence, m is proper iff $P(L) \cap L' = Subset(P(L)) \cap L'$. □

Before presenting the main result on contractibility of edit-based soft constraints, we need to introduce some preliminary results on weighted edit distance.

We say a sequence of edit operations is in *normal form* if the edit operations are grouped by type so that all deletions are performed before all transpositions, which are performed before all substitutions, before all insertions, and no letter is subject to two or more substitutions. It is not difficult to show that any edit sequence has a corresponding sequence in normal form that achieves the same result at lower or equal cost.

Lemma 17

Consider a weighted edit-distance and a word $\vec{a}$. For any edit sequence that maps $\vec{a}$ to $\vec{b}$, there is an edit sequence in normal form that also maps $\vec{a}$ to $\vec{b}$ with a shorter or equal weighted edit distance.

It is straightforward to see that, for any edit sequence not involving transposition and any weighted edit measure, there is an equivalent edit sequence where each letter is edited at most once. Provided the edit weights satisfy a simple property, this result extends to edit sequences involving transposition.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Proposition 18
Consider an edit-based violation measure, where $\beta + \gamma \leqslant 2\delta$. Suppose, we wish to edit a word $\vec{a}$ so that it appears in a language L. Then, there is an edit of minimal cost where no letter is subject to more than one edit operation.

Proof
Suppose $\beta + \gamma \leqslant 2\delta$ and consider any edit sequence that maps $\vec{a}$ to $\vec{b} \in L$. We can assume (Lemma 17) that edit operations are grouped: deletions, then transpositions, substitutions, and finally insertions.

Suppose a letter a that participates in a transposition also participates in another edit operation. Then, the second operation is either another transposition or a substitution.

In the former case, consider all transposition operations that are applied to a. The effect of these edits is to move a from some position i to a position j. This sequence can be replaced by the deletion of a at position i and the insertion of a at position j. The revised edit sequence has a lower or equal cost because $\beta + \gamma \leqslant 2\delta$, and we assumed that at least two transpositions are involved.

In the latter case, aa' is edited to $a'a$ and later a is changed to b, for some a' and b. We can achieve the same effect by substituting a' for a and b for a instead of the transposition and substitution. The revised edit sequence has lower or equal cost if $\alpha \leqslant \delta$. Alternatively, we can replace the original edit operations by the deletion of a and the insertion of b on the right of a'. This revised edit sequence has lower or equal cost if $\delta \leqslant \alpha$, because $\beta + \gamma \leqslant 2\delta \leqslant \alpha + \delta$. Thus, independent of whether $\alpha \leqslant \delta$ or $\delta \leqslant \alpha$, a lower cost edit sequence is obtained with fewer instances of a letter involved in two edit operations.

The remaining possibility is that a substitution operation is applied twice to a letter. It is clear that the first substitution operation can be omitted.

Repeatedly applying normal form transformations and the edit modifications described above, all occurrences of a letter being edited twice can be removed. □

In particular, this lemma holds when the edit operations are unweighted (that is, when $\alpha = \beta = \gamma = \delta$). The property that each letter is edited at most once is important for network flow implementations of propagators such as the propagators for soft REGULAR in van Hoeve *et al.* (2006) and Maher (2009d).

Edit-based violation measures are monotonic with respect to both the weights and the language.

Lemma 19
Let m (m') be edit-based violation measures with weights $\alpha, \beta, \gamma, \delta$ (respectively, $\alpha', \beta', \gamma', \delta'$) for the same language. If $\alpha \leqslant \alpha'$, $\beta \leqslant \beta'$, $\gamma \leqslant \gamma'$, and $\delta \leqslant \delta'$, then, for all words w, $m(w) \leqslant m'(w)$.

Proof
For every word w, consider an edit that achieves the minimum violation $m'(w)$. Let n_s, n_i, n_d, n_t be the number of the respective operations used in the edit. Then, $m'(w) = \alpha' n_s + \beta' n_i + \gamma' n_d + \delta' n_t \geqslant \alpha n_s + \beta n_i + \gamma n_d + \delta n_t \geqslant m(w)$. Hence, $m(w) \leqslant m'(w)$. □

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Lemma 20
Let m_1 and m_2 be edit-based violation measures with the same weights, for languages L_1 and L_2, respectively. If $L_1 \subseteq L_2$, then for all words w, $m_1(w) \geqslant m_2(w)$.

Proof
For every word w, any edit to L_1 is also an edit to L_2. Since an edit-based violation measure minimizes over all edits, we must have $m_1(w) \geqslant m_2(w)$. □

In many cases, edit-based violation measures lead to contractible soft constraints.

Theorem 21
Let C_s be a soft constraint with an open edit-based violation measure, and suppose $\min\{\alpha, \beta, \gamma\} \leqslant \delta$.

Then, C_s is contractible.

Proof
Consider the sequence of edits that transforms an instance $\vec{a}a'$ of $\vec{X}Y$ into an element $\vec{b}$ of $P(L_C)$ at minimum cost. By Lemma 17, we can assume that all deletions occur before any transpositions, and all insertions and substitutions occur after all transpositions. We now identify modifications of this sequence of edits that transform $\vec{a}$ into an element of $P(L_C)$ at lower (or equal) cost than the original sequence.

If a' is deleted in the original sequence, then the sequence of edits omitting this deletion transforms $\vec{a}$ to $\vec{b}$ at lower or equal cost. Otherwise, if a' is not involved in a transposition, then the subsequence of edits that do not involve a' transforms $\vec{a}$ into a prefix of $\vec{b}$ (which is an element of $P(L_C)$). The subsequence has a lower or equal cost, since it involves a subset of the edits.

The remaining possibility is that a' is involved in a transposition. Let p be the position of a' after all transpositions. The sequence of edits that omits all transpositions involving a' and then inserts a' at position p transforms $\vec{a}$ to $\vec{b}$. These edits have a lower or equal cost if $\beta \leqslant \delta$.

Alternatively, let the length of $\vec{a}a'$ after all deletions be $n+1$ (so that a' is in position $n+1$). Every transposition involving position $n+1$ in the original sequence can be replaced by a substitution that replaces the letter at position n by the letter at position $n+1$ at the corresponding stage of the original transformation. This transforms $\vec{a}$ into a prefix of $\vec{b}$ at lower or equal cost if $\alpha \leqslant \delta$.

Finally, let $\vec{a}_1 a' \vec{a}_2$ be the result of deletions and transpositions on $\vec{a}a'$. The length of $\vec{a}_2$ is a lower bound for number of transpositions involving a' in editing $\vec{a}$ into $\vec{b}$. The sequence of edits that deletes all letters of $\vec{a}_2$ and applies all substitutions and insertions that apply to $\vec{a}_1$ transforms $\vec{a}$ into a prefix of $\vec{b}$. These edits have a lower or equal cost if $\gamma \leqslant \delta$, since transpositions are replaced by deletions and some edits might now be omitted.

In each case, for all words $\vec{a}a'$, we find that $\vec{a}$ has a smaller weighted edit distance to $P(L_C)$ than $\vec{a}a'$. This demonstrates that the violation measure is non-decreasing and hence, by Proposition 8, C_s is contractible. □

Example 7 below shows that this theorem cannot be strengthened without imposing extra conditions on C_s.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

It follows from the theorem that edit-based measures that only involve substitutions, insertions and deletions provide contractible constraints. Thus, the variable-based measures (Petit *et al.* 2001; van Hoeve *et al.* 2006), the object-based measures (Beldiceanu and Petit 2004), and the edit-based measures of van Hoeve *et al.* (2006) induce contractible soft constraints.

For order-free constraints, transposition is not needed in an edit and can be effectively given infinite weight. Thus, by Theorem 21, we have

Corollary 22
If C is an order-free constraint and the corresponding soft constraint C_s is based on an open edit-based violation measure m, then C_s is contractible.

We also have the following curious result.

Corollary 23
Let C_s be a soft constraint based on an open edit-based violation measure m with weights $\alpha, \beta, \gamma, \delta$ for the hard constraint C. If any of α, β, γ, or δ is 0, then C_s is contractible.

Proof
If α, β, or γ is 0, then the condition of Theorem 21 is satisfied and consequently C_s is contractible. If δ is 0, then transpositions can place the letters in a word in any order, at no cost. Let

$$C'([X_1, \ldots, X_n]) \leftrightarrow \bigvee_{\pi} C([X_{\pi(1)}, \ldots, X_{\pi(n)}])$$

where the disjunction is over all permutations π of $1..n$. Then, the violation measure m of C is equal to the violation measure m' of C', where m' uses the same weights as m. C' is order-free and, by Corollary 22, is contractible. □

From these results, we see that soft constraints based on a wide range of edit-based measures are contractible. However, when transpositions are allowed and have a comparatively low cost, an edit-based violation measure can lead to a soft constraint that is not contractible.

Example 7
Consider a constraint C with $L_C = (ab)^* + (ab)^* a$, which is a prefix-closed language, and consider the corresponding soft constraint C_s that uses an edit-based violation measure. Suppose $\delta < \min\{\alpha, \beta, \gamma\}$. The word *abba* has edit distance δ, by transposing the last two letters, but its prefix *abb* has edit distance $\min\{\alpha, \beta, \gamma\}$, since we could either substitute *a* for *b*, insert *a* before the second *b*, or delete a *b*. Thus, the weighted edit-based violation measure is not non-decreasing and hence, by Proposition 8, C_s is not contractible.

This example reinforces a point made earlier: the introduction of $P(L_C)$ to the definition of edit-based violation measure plays a different role than its use for hard constraints; in this case, its use does not ensure contractibility.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

8 Contractible approximations of soft constraints

Although we have identified powerful sufficient conditions for soft constraints to be contractible, we must also be able to support uncontractible soft constraints. As with hard constraints, when a soft constraint is uncontractible, we can use a contractible approximation as the basis for filtering while the constraint is open.

We reformulate the notion of tight approximation for soft constraints of the form $m(\vec{X}) \leqslant Z$ as follows. A violation measure m_1 is an *approximation* of the violation measure m if, for all words $\vec{a}$, $m_1(\vec{a}) \leqslant m(\vec{a})$. We order violation measures with the pointwise extension of the ordering on the reals: $m_1 \leqslant m_2$ iff $\forall \vec{a}\ m_1(\vec{a}) \leqslant m_2(\vec{a})$. A contractible approximation m_1 to a violation measure m is *tight* if, for all contractible functions m_2, if $m_1 \leqslant m_2 \leqslant m$, then $m_2 = m_1$. Given two contractible approximations m_1 and m_2 to a violation measure m, we say m_2 is *tighter* than m_1 if $m_1 \leqslant m_2$. We write m^* to denote the tightest contractible approximation of m.

We can characterize the tightest contractible approximation of a violation measure, independent of how the violation measure is formulated.

Proposition 24
Let m be a violation measure. The tightest contractible approximation to m is characterized by $m^*(\vec{a}) = \inf_{\vec{b}} m(\vec{a}\vec{b})$, where the infimum is taken over all finite sequences $\vec{b}$.

Proof
By definition, $m^*(\vec{a}) \leqslant m(\vec{a})$, so m^* approximates m. Consider a sequence $\vec{a}$ and a letter c. $m^*(\vec{a}c) = \inf_{\vec{b}} m(\vec{a}c\vec{b}) \geqslant \inf_{c\vec{b}} m(\vec{a}c\vec{b}) \geqslant \inf\{m(\vec{a}), \inf_{c\vec{b}} m(\vec{a}c\vec{b})\} = m^*(\vec{a})$. Thus, m^* is contractible.

Suppose some function k is a strictly tighter contractible approximation than m^*. Then, for some $\vec{a}$, $k(\vec{a}) > m^*(\vec{a})$, that is, $k(\vec{a}) > \inf_{\vec{b}} m(\vec{a}\vec{b})$. Hence, there is a $\vec{d}$ such that $k(\vec{a}) > m(\vec{a}\vec{d})$. But, for any $\vec{c}$, $k(\vec{a}\vec{c}) \geqslant k(\vec{a})$. Thus, we have $m(\vec{a}\vec{d}) > m(\vec{a}\vec{d})$. This contradiction shows that k cannot exist; m^* is the tightest contractible approximation to m. □

This proposition only provides a mathematical characterization; it does not suggest an implementation. Indeed, it appears very difficult to implement this tightest contractible approximation, in general, in contrast to the tightest contractible approximation of hard constraints. Nevertheless, we can identify some contractible approximations.

8.1 Decomposition-based violation measures

One way to obtain a contractible approximation to a decomposition-based soft constraint is to ignore parts of a decomposition that cause incontractibility. A *weakening* of a decomposition of a constraint $C(\vec{X})$ is a function that, for every sequence $\vec{X}$, maps the decomposition $(\vec{X}, \vec{U}, T, S, w)$ to $(\vec{X}, \vec{U}, T, S', w')$, where (S', w') is a sub-weighted set of (S, w). For this weakened decomposition, we can apply the sufficient condition of Theorem 15.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Proposition 25

Consider a decomposition-based violation measure m for a constraint $C(\vec{X})$ and a weakening W of the decomposition. Suppose m is defined via a monotonic combining function. If, for every sequence $\vec{X}$, the weakening of the decomposition of $C(\vec{X})$ can be semantically embedded in the weakening of the decomposition of $C(\vec{X}Y)$, then the measure m' defined by using the weakened decompositions is a contractible approximation of m.

Proof

m' is an approximation of m because the combining function is monotonic and the weakened decomposition employs a sub-weighted set of the original decomposition. m' is contractible by application of Theorem 15. □

This result shows an approach to finding a contractible approximation to $C_s(\vec{X})$. However, there is no guarantee that it will find a good approximation; in the worst case, it might provide only the trivial approximation, where all of $C(\vec{X})$ is ignored. Nevertheless, it appears to be useful.

The next example presents an uncontractible decomposition-based soft constraint. It employs a decomposition of the global cardinality constraint GCC given in Bessiere *et al.* (2009).

Example 8

Consider the global cardinality constraint $\mathrm{GCC}(\vec{X}, \vec{l}, \vec{u})$ with type T that maps each X_i to $1..d$, which we denote by GCC_T. This constraint expresses that, for each value t in $1..d$, the number of occurrences of t in $\vec{X}$ lies between l_t and u_t (u_t may be infinite). $\vec{l}$ and $\vec{u}$ are fixed. To define the decomposition of Bessiere *et al.* (2009), we need to introduce variables A_{ilu} of type $\{0,1\}$ and N_{lu} of type non-negative integers, and elementary constraints as follows. Let $n = |\vec{X}|$.

For $1 \leqslant i \leqslant n$, $1 \leqslant l \leqslant u \leqslant d$, and $1 \leqslant k < u$, we have the constraints

$$A_{ilu} = 1 \leftrightarrow X_i \in [l, u] \tag{8.1}$$

$$N_{lu} = \sum_{i=1}^{n} A_{ilu} \tag{8.2}$$

$$N_{1u} = N_{1k} + N_{(k+1)u} \tag{8.3}$$

$$\sum_{j=l}^{u} l_j \leqslant N_{lu} \leqslant \sum_{j=l}^{u} u_j \tag{8.4}$$

Formally, the decomposition of GCC_T is $(\vec{X}, \vec{A}, \vec{N}, T', S, w)$, where T' is the extension of T to $\vec{A}$ and $\vec{N}$, S is the collection of (8.1), (8.2), (8.3), and (8.4), and w is a constant function. It is easy to establish that $\mathrm{GCC}_T(\vec{X}, \vec{l}, \vec{u}) \leftrightarrow \exists \vec{A} \in T'(\vec{A})\ \exists \vec{N} \in T'(\vec{N})\ S$.

When $\vec{X}$ is extended by Y, the decomposition contains extra variables $A_{(n+1)lu}$, extra constraints of type (8.1), involving Y and the new variables, and replaces

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

constraints (8.2) by

$$N_{lu} = \sum_{i=1}^{n+1} A_{ilu} \tag{8.5}$$

Consider an occurrence of the constraint $\mathrm{GCC}_T([X_1, X_2], [0, 1, 0, 0], [2, 2, 2, 2])$, where $T(X_i)$ is 1..4. Consider a valuation v where $X_1 = 1, X_2 = 1$. For all extensions of v to $\vec{A}$ and $\vec{N}$, there will be an elementary constraint violated (fundamentally because the lower bound for occurrences of the domain value 2 has not been satisfied). If $\vec{X}$ is extended by X_3 and v has $X_3 = 2$, then v can be extended to $\vec{A}$ and $\vec{N}$ in the obvious way to satisfy all elementary constraints. Thus, any proper violation measure for GCC based on this decomposition is not contractible.

Let m be a proper violation measure that is defined with a combining function that is monotonic and has unit 0. If we weaken the decomposition by ignoring the lower bounds in (8.4), then we have a contractible approximation m' of m. (This is essentially the same as for the tight contractible approximation of the hard GCC constraint, which is also obtained by ignoring lower bounds. This point is not so surprising when we recall that the hard constraint is a special case of the soft constraint.) We can see this using the natural semantic embedding (which maps all constraints to themselves, except that (8.2) is mapped to (8.5)) and Proposition 25.

We conjecture that the weakening of the soft GCC constraint in this example is its tightest contractible approximation. However, the many variables and constraints in the decomposition make it difficult to confirm this conjecture.

8.2 Edit-based violation measures

Recall that an edit-based violation measure m is contractible if $\delta \geqslant \min\{\alpha, \beta, \gamma\}$ (Theorem 21). If $\delta < \min\{\alpha, \beta, \gamma\}$, then m might be uncontractible and we must consider contractible approximations. We can provide generic contractible approximations for edit-based soft constraints by modifying the weights to accord with the sufficient conditions of Theorem 21 and Corollary 23.

Proposition 26
Let m be an open edit-based violation measure for a constraint C with weights $\alpha, \beta, \gamma, \delta$, where $\delta < \min\{\alpha, \beta, \gamma\}$. Then, the following violation measures are contractible approximations of m for C.

1. m_1 based on weights $\delta, \beta, \gamma, \delta$ (that is, $\alpha := \delta$)
2. m_2 based on weights $\alpha, \delta, \gamma, \delta$ (that is, $\beta := \delta$)
3. m_3 based on weights $\alpha, \beta, \delta, \delta$ (that is, $\gamma := \delta$)
4. m_4 based on weights $\alpha, \beta, \gamma, 0$ (that is, $\delta := 0$)
5. m_5 defined by $m_5(w) = \max\{m_1(w), m_2(w), m_3(w), m_4(w)\}$

Proof
By Lemma 19, for any w, $m_1(w) \leqslant m(w)$, $m_2(w) \leqslant m(w)$, $m_3(w) \leqslant m(w)$, and $m_4(w) \leqslant m(w)$. It then follows from the definition of m_5 that $m_5(w) \leqslant m(w)$. Thus

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

,m_1, m_2, m_3, m_4, and m_5 are approximations of m. By Theorem 21, m_1, m_2, and m_3 are contractible and, by Corollary 23, m_4 is contractible. For any word w and letter a,

$$\begin{aligned} m_5(wa) &= \max\{m_1(wa), m_2(wa), m_3(wa), m_4(wa)\} \\ &\geqslant \max\{m_1(w), m_2(w), m_3(w), m_4(w)\} \\ &= m_5(w) \end{aligned}$$

using the contractibility of $m_1, \ldots, m_4$.

Thus, m_5 is contractible. □

Note that, by Lemma 19, other uses of Corollary 23 yield only measures that are not as tight as m_1, m_2, or m_3. Clearly, m_5 is the tightest of these approximations. However, in general, this approximation is not tight, as the following example shows.

Example 9

Let $L = (abc)^*$, so that $P(L) = L \cup La \cup Lab$. Let $\alpha = \beta = \gamma = 4$ and $\delta = 1$. Consider $w = bbb(abc)^3ca$. Two kinds of edits are needed, addressing the initial b's and the trailing ca. Then, $m(w) = 12$ from substituting for the first and third b, and deleting the last c. $m(wb) = 10$ using the same substitutions and two transpositions on c. Thus, m is not contractible.

Notice that the initial b's in w are too far from the end of w to be cheaply addressed by transpositions. For example, the cost of moving the third b to the trailing ca is 6, which is more expensive than addressing it by substitution. The other b's are even more expensive to address by transposition. Thus, the minimal cost of addressing the initial b's is 8. The minimal cost of addressing the trailing ca arises when a b is appended to the end of w and c is transposed twice. This has a cost of 2, and it is easy to see that no word appended to w will allow ca to be addressed by a single transposition. Thus, the tightest approximation to m has $m^*(w) = 10$.

Now consider the approximations in Proposition 26. If we reduce α to 1, then $m_1(w) = 4$ by applying four substitutions. If we reduce β to 1, then $m_2(w) = 8$ by inserting a and c around each initial b and inserting ab before the last c. If we reduce γ to 1, then $m_3(w) = 4$ by deleting the three b's and the last c. If we reduce δ to 0, then $m_4(w) = 4$ by applying transpositions to reorder w to $(abc)^4bb$ and then substituting a for b. Thus, $m_5(w) = 8$.

This shows that m_5 is not the tightest contractible approximation to m, since $m_5(w) \neq m^*(w)$.

The question now arises: how to express m^* in edit-based terms so that a closed propagator for $m(\vec{X}) \leqslant Z$ might be adapted to implement $m^*(\vec{X}) \leqslant Z$, as was done for hard constraints in Section 6. Disappointingly, this turns out to be impossible, in general.

We first establish a straightforward lemma that gives a simple way of identifying the value of $m^*(w)$ in some cases.

Lemma 27

Let m^* be the tightest contractible approximation to an edit-based violation measure m. Let w be a word. If, for all words u, $m(wu) \geqslant m(w)$, then $m^*(w) = m(w)$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

Proof
If, for all words u, $m(wu) \geqslant m(w)$, then $\inf_u m(wu) \geqslant m(w)$. Thus, $m^*(w) \geqslant m(w)$. Since m^* approximates m, $m^*(w) \leqslant m(w)$. Hence, $m^*(w) = m(w)$. □

Now, we show that, in general, the tightest contractible approximation m^* to an edit-based violation measure m cannot be expressed as a proper edit-based violation measure.

Theorem 28
There is an open edit-based violation measure m for a language L such that its tightest contractible approximation cannot be expressed as a proper edit-based violation measure on any language.

Proof
Consider the alphabet $\Sigma = \{a, b, c, d\}$. As in Example 9, let $L = (abc)^*$ (so $P(L) = L \cup La \cup Lab$), consider $P(L)$ as a sublanguage of Σ^*, and let m be the edit-based violation measure for L, where $\alpha = \beta = \gamma = 4$ and $\delta = 1$. As shown in Example 9, m is not contractible. Note that m is proper. Let m^* be the tightest contractible approximation to m. Suppose m^* can be expressed as a proper edit-based violation measure m' on some language L'.

Suppose there is some word w such that $w \in L' \backslash P(L)$. Then, $m'(w) = 0$. Hence, $m^*(w) = 0$ and, from Proposition 24, there is a word u such that $m(wu) = 0$. Since m is proper, $wu \in P(L)$ and hence, $w \in P(L)$. This contradiction shows that no such w exists and hence $L' \subseteq P(L)$.

For every $w \in L$, $m(w) = 0$. Hence, $m^*(w) = 0$ and $m'(w) = 0$. Hence, $w \in L'$, since m' is proper. Hence, $L \subseteq L'$.

There are weights α', β', γ', and δ' used to define m'. We now consider different words w of Σ^* and derive conditions on the weights of m'. We use the fact that every edit of w to $P(L)$ must have cost greater than or equal to $m^*(w)$. Because $L' \subseteq P(L)$, the conditions we derive about editing a word to $P(L)$ also apply to L'.

$w = d$.
$m(w) = 4$ by deleting d and no word wu has a smaller violation measure because d must be deleted or substituted. Thus, by Lemma 27, $m^*(w) = 4$. w might be edited to L (and hence also L') by deleting d or substituting a for d. This gives rise to the conditions $\gamma' \geqslant 4$ and $\alpha' \geqslant 4$, since, for example, if $\gamma' = 3$, then $m'(w) = 3 \neq m^*(w)$.

$w = bc(abc)^3$.
$m(w) = 4$, by inserting a at the beginning of w. No word wu has a lower cost because the initial bc is too far from the end of w to use transposition from u at a lower cost. Thus, $m^*(w) = 4$. From this, we obtain the condition $\beta' \geqslant 4$, among others.

$w = ba$.
$m(w) = 1$ by transposition and we find that $m^*(w) = 1$. Because we know that $\alpha' \geqslant 4$, $\beta' \geqslant 4$ and $\gamma' \geqslant 4$, we must have $\delta' = 1$.

[We can now establish that $L' \subseteq (a+b+c)^*$. For any word w involving d, $m(w) \geqslant 4$, since the d must be deleted or substituted. That includes wu, for any u, and hence $m^*(w) \geqslant 4$ for any word containing d. Consequently, also $m'(w) \geqslant 4$ and, since m' is proper, L' does not contain a word involving d.]

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

$w = adc$.

$m(w) = 4 = m^*(w)$. Given that $\delta' = 1$ and the small size of w, no word is edit distance 4 from w using transposition alone. But $\alpha' \geqslant 4$, $\beta' \geqslant 4$, and $\gamma' \geqslant 4$, so the minimum cost edit from w to L' does not involve transposition and the edit consists of a single operation. Since L' does not contain words involving d, the only candidates are deletion or substitution of d. The deletion results in ac which is not in L' since it is not in $P(L)$. Hence, the only edit that can achieve this cost is a substitution of b for d, and $\alpha' = 4$.

$w = d(abc)^3$.

As with $w = d$, $m^*(w) = m(w) = 4$. Given that $\alpha' \geqslant 4$, only deletion of d can achieve this cost. Hence, $\gamma' = 4$.

$w = bc(abc)^3$, again.

Given that $\alpha' \geqslant 4$ and $\gamma' \geqslant 4$, the only edit that can achieve $m^*(w) = 4$ is an insertion. Hence, $\beta' = 4$.

Thus, the weights for m' are exactly the same as the weights for m. For every word w, $m'(w) \geqslant m(w)$, by Lemma 20. From the definition of m^*, $m(w) \geqslant m^*(w)$. But $m' = m^*$, by assumption, and hence $m^* = m$. But this is a contradiction, because by definition m^* is contractible, while m is not. Thus, the assumption that m^* can be expressed as an edit-based violation measure is false. □

The language and violation measure demonstrating this claim are those from Example 9. Given that the language is so simple, we can expect that many uncontractible edit-based violation measures cannot be tightly approximated by a contractible edit-based violation measure. This contrasts markedly with our work on hard constraints in Section 6, where tight contractible approximations of several uncontractible hard constraints were formulated in terms of the original hard constraint.

It suggests some difficulties in implementing tight contractible approximations. It seems that the edit-based implementation of the closed constraint is not a suitable basis for implementing the tight approximation. At least, we need a different framework if we are to have a comprehensive method to derive open D-consistent propagators for incontractible soft constraints.

It is demonstrated in He *et al.* (2013) that using an approximation of the violation measure of a *closed* edit-based soft constraint can lead to incorrect answers to constraint problems. However, using a non-tight contractible approximation in an *open* constraint is less serious, assuming the correct violation measure is used for the closed constraint: the search may perform less-than-optimal pruning, leading to a greater search space than for a tight contractible approximation, but not to incorrect answers. Thus, Theorem 28 does not represent a failure of correctness, only a degree of inefficiency if a non-tight edit-based contractible approximation is used.

9 Discussion

We have discussed open constraints, where variables are added to the right-hand end of the sequence. This directly affects the characterization of contractibility

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

and the definition of open D-consistency. If, instead, variables are added to the left-hand end, then the appropriate characterization of contractibility is suffix-closure. If additions may be made at either end, then contractibility requires both closures, which corresponds to closure under taking subwords[12]. Constraints like SEQUENCE and CONTIGUITY are subword-closed. Of course, all order-free constraints are subword-closed. On the other hand, the lexicographic ordering constraint $\leqslant_{lex}$ and the precedence constraint $s \prec_{\vec{X}} t$ are prefix-closed but not suffix-closed, that is, they are contractible if variables are added on the right, but not if variables are added on the left.

If additional variables may be inserted anywhere within the sequence, then contractibility corresponds to closure under taking subsequences[13]. Apart from the order-free constraints, it is not clear whether there is any useful constraint that is closed under taking subsequences.

In Maher (2009b), a dual notion to contractibility, called extensibility, is investigated. In contrast to contractibility, in general there is no closure operation corresponding to extensibility and consequently no tightest extensible approximation.

We have seen some differences between contractibility for hard and soft constraints. For hard constraints, contractibility depends on the relation, whereas for soft constraints it depends on the violation measure. For example, the soft REGULAR constraint is contractible under the edit-based measure of van Hoeve *et al.* (2006) but not under decomposition-based measures. We have also seen that many tight contractible approximations of hard constraints are similar to, though weaker than, the hard constraint. On the other hand, for many soft constraints it appears that the tight contractible approximations cannot be expressed in the same way as the soft constraint. This suggests that it may be difficult to formulate full open D-consistent propagators, for example, for uncontractible open soft constraints.

There are several similarities between violation measures and other treatments of soft constraints. For example, the Valued CSP (Schiex *et al.* 1995) and the Semi-Ring CSP (Bistarelli *et al.* 1997) frameworks define a soft constraint essentially as a function from valuations to an ordered set (the set may be partially ordered in the case of SCSPs) that might be considered a violation measure. Both frameworks use a combining function to extend this definition to a collection of constraints, and so they are, in many ways, like decomposition-based violation measures. However, both frameworks consider only closed constraints and focus on finite relations defined extensionally.

Weighted violation measures are used in Métivier *et al.* (2007, 2009). As noted earlier, the decomposition measures presented here generalize the weighted decomposition measures for Σ-ALLDIFFERENT and Σ-GCC (Métivier *et al.* 2007, 2009). However, the edit-based violation measures presented here do not generalize the weighted edit distance for Σ-ALLDIFFERENT and Σ-REGULAR of Métivier *et al.*

[12] A word w is a *subword* of $a_1 \dots a_n$ if w is empty or has the form $a_i a_{i+1} \dots a_j$ for some $1 \leqslant i \leqslant j \leqslant n$.
[13] A word w is a *subsequence* of $a_1 \dots a_n$ if w is empty or has the form $a_{i_1} \dots a_{i_k}$ for some $k \leqslant n$ where $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$.

https://doi.org/10.1017/S1471068417000126 Published online by Cambridge University Press

(2007, 2009). These measure use only substitution edits but they assign weights to each variable.

Violation measures play a similar role to query measures (Maher and Stuckey 1989) that were used to specify preferences on query solutions in a CLP system. In this context, contractible violation measures are similar to pruning measures in Maher and Stuckey (1989) in that both are non-decreasing functions, although over different domains, and both permit the safe pruning of search trees.

Contractible global soft constraints are amenable to a nested representation (Bessiere *et al.* 2014) in a distributed constraint optimization setting, which has significant performance gains over other representations (Bessiere *et al.* 2014).

Finally, we note that the semantics of soft constraints are examples of quantitative languages, in the terminology of Chatterjee *et al.* (2010). From this point of view, an approximation of a violation measure is a quantitative language inclusion. However, Chatterjee *et al.* (2010) focuses on languages of infinite words defined via automata, so the results of Chatterjee *et al.* (2010) do not seem to have application to the subject of this paper. In Colcombet (2009), a notion of cost function on languages of finite words is used but this is only used to define equivalence classes and is not related to this paper.

10 Conclusions

We have introduced the notion of contractibiliity of global constraints, which ensures that constraint propagation for closed constraints is safe for open constraints, and characterized it in language-theoretic terms. The concept of contractibility is remarkably robust. It is based only on the relation, or language, defining the constraint. Thus, it is independent of the form of propagator used (monolithic or decomposed) and the consistency condition (if any) that characterizes the propagation.

Contractibility appears to be central to the re-use of closed constraint propagators for open propagation. When a constraint is contractible, we only need to modify a closed propagator to support the addition of variables. When a constraint is incontractible, we also need a contractible approximation of the propagator, for use while the constraint is open, in addition to the closed propagator. We showed that the use of a tight contractible approximation and domain consistent closed propagators achieves open D-consistency of the resulting open propagator. Furthermore, for many hard constraints (Regular, CFG, GCC, and many others), we showed that the tightest contractible approximation has a similar form to the original constraint, and hence can be propagated by the same techniques. This suggests that a close integration of the two propagation phases will be easy for these constraints.

To address soft constraints, we formulated two general classes of soft constraints that include most previous proposals of soft constraints. For the two classes – based on decomposition and edit-distance, respectively – we identified properties and developed mathematical tools for reasoning about them, which we used to demonstrate the contractibility of a wide range of soft constraints. We identified pragmatic contractible approximations of soft constraints in these classes. However,

we also established that the tightest contractible approximation of an edit-based soft constraint is not expressible, in general, as an edit-based constraint. This suggests difficulties in designing open D-consistency propagators in the general case, but fortunately many edit-based soft constraints are contractible.

These results provide a good basis for adapting existing algorithms and implementations of global constraint propagators to open constraints.

Acknowledgements

The author thanks the referees of this paper and previous conference papers, whose thorough reviews and detailed comments improved this paper. The work in this paper was mostly conducted while the author was employed by NICTA.

References

Barták, R. 1999. Dynamic constraint models for planning and scheduling problems. In *New Trends in Constraints, Joint ERCIM/Compulog Net Workshop*, Paphos, Cyprus, October 25–27, 1999.

Barták, R. 2003. Dynamic global constraints in backtracking based environments. *Annals of Operations Research 118,* 1–4, 101–119.

Beldiceanu, N. and Carlsson, M. 2001. Revisiting the cardinality operator and introducing the cardinality-path constraint family. In *Proc. of 17th International Conference on Logic Programming, ICLP*, Paphos, Cyprus, November 26–December 1, 2001, 59–73.

Beldiceanu, N., Carlsson, M. and Rampon, J.-X. 2005. Global constraint catalog. Technical Report T2005:08, SICS. Current version available at `http://sofdem.github.io/gccat/`.

Beldiceanu, N. and Contejean, E. 1994. Introducing global constraints in CHIP. *Mathematical Computer Modelling 20,* 12, 97–123.

Beldiceanu, N. and Petit, T. 2004. Cost evaluation of soft global constraints. In *Proc. of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 1st International Conference, CPAIOR 2004*, Nice, France, April 20–22, 2004, 80–95.

Bessière, C. 1991. Arc-consistency in dynamic constraint satisfaction problems. In *Proc. of the 9th National Conference on Artificial Intelligence*, Anaheim, CA, USA, July 14–19, 1991, vol. 1, 221–226.

Bessiere, C., Brito, I., Gutierrez, P. and Meseguer, P. 2014. Global constraints in distributed constraint satisfaction and optimization. *Computer Journal 57,* 6, 906–923.

Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z. and Walsh, T. 2008. SLIDE: A useful special case of the CARDPATH constraint. In *Proc. of 18th European Conference on Artificial Intelligence, ECAI*, Patras, Greece, July 21–25, 2008, 475–479.

Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C. and Walsh, T. 2009. Decompositions of all different, global cardinality and related constraints. In *Proc. of the 21st International Joint Conference on Artificial Intelligence, IJCAI*, Pasadena, California, USA, July 11–17, 2009, 419–424.

Bistarelli, S., Montanari, U. and Rossi, F. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM 44,* 2, 201–236.

Borning, A., Freeman-Benson, B. N. and Wilson, M. 1992. Constraint hierarchies. *Lisp and Symbolic Computation 5,* 3, 223–270.

Borning, A., Maher, M. J., Martindale, A. and Wilson, M. 1989. Constraint hierarchies and logic programming. In *Proc. of 6th International Conference on Logic Programming,* Lisbon, Portugal, June 19–23, 1989, 149–164.

Brzozowski, J. A., Shallit, J. and Xu, Z. 2009. Decision problems for convex languages. In *Proc. of Language and Automata Theory and Applications, 3rd International Conference, LATA*, 2009, Tarragona, Spain, April 2–8, 2009, 247–258.

Chandra, A. K. and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th Annual ACM Symposium on Theory of Computing*, May 4–6, 1977, Boulder, Colorado, USA, 77–90.

Chatterjee, K., Doyen, L. and Henzinger, T. A. 2010. Quantitative languages. *ACM Transactions on Computational Logic 11,* 4.

Chaudhuri, S. and Vardi, M. Y. 1993. Optimization of *Real* conjunctive queries. In *Proc. of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 25–28, 1993, Washington, DC, USA, 59–70.

Colcombet, T. 2009. The theory of stabilisation monoids and regular cost functions. In *Proc. of Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP Part II*, Rhodes, Greece, July 5–12, 2009, 139–150.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. 2001. *Introduction to Algorithms,* 2nd ed. The MIT Press and McGraw-Hill Book Company.

Debruyne, R., Ferrand, G., Jussien, N., Lesaint, W., Ouis, S. and Tessier, A. 2003. Correctness of constraint retraction algorithms. In *Proc. of the 16th International Florida Artificial Intelligence Research Society Conference*, May 12–14, 2003, St. Augustine, Florida, USA, 172–176.

Dechter, R. 2003. *Constraint Processing*. Elsevier Morgan Kaufmann.

Dechter, R. and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proc. of the 7th National Conference on Artificial Intelligence*, St. Paul, MN, August 21–26, 1988, 37–42.

Faltings, B. and Macho-Gonzalez, S. 2002. Open constraint satisfaction. In *Proc. of Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002*, Ithaca, NY, USA, September 9-13, 2002, 356–370.

Faltings, B. and Macho-Gonzalez, S. 2005. Open constraint programming. *Artificial Intelligence 161,* 1–2, 181–208.

Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I. and Walsh, T. 2002. Global constraints for lexicographic orderings. In *Proc. of Principles and Practice of Constraint Programming – CP 2002, 8th International Conference, CP 2002*, Ithaca, NY, USA, September 9–13, 93–108.

Gavanelli, M., Lamma, E., Mello, P. and Milano, M. 2005. Dealing with incomplete knowledge on clp(*FD*) variable domains. *ACM Transactions on Programming Languages and Systems 27,* 2, 236–263.

Georget, Y., Codognet, P. and Rossi, F. 1999. Constraint retraction in CLP(FD): Formal framework and performance results. *Constraints 4,* 1, 5–42.

Gervet, C. 1997. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints 1,* 3, 191–244.

He, J., Flener, P. and Pearson, J. 2013. Underestimating the cost of a soft constraint is dangerous: Revisiting the edit-distance based soft regular constraint. *Journal of Heuristics 19,* 5, 729–756.

Hentenryck, P. V. and Provost, T. L. 1991. Incremental search in constraint logic programming. *New Generation Computing 9,* 3/4, 257–276.

Hopcroft, J. and Ullman, J. 1979. *Introduction to Automata Theory Languages and Computation.* Addison-Wesley.

Ioannidis, Y. E. and Ramakrishnan, R. 1995. Containment of conjunctive queries: Beyond relations as sets. *ACM Transaction on Database System 20,* 3, 288–324.

Jaffar, J. and Maher, M. J. 1994. Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, 503–581.

Klug, A. C. 1988. On conjunctive queries containing inequalities. *Journal of ACM 35,* 1, 146–160.

Lallouet, A., Law, Y. C., Lee, J. H. and Siu, C. F. K. 2011. Constraint programming on infinite data streams. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, Barcelona, Catalonia, Spain, July 16–22, 2011, 597–604.

Law, Y. C. and Lee, J. H. 2004. Global constraints for integer and set value precedence. In *Proc. of Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004*, Toronto, Canada, September 27–October 1, 2004, 362–376.

Maher, M. J. 1988. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming*. J. Minker (Ed). Morgan Kaufmann, 627–658.

Maher, M. J. 1993. A logic programming view of CLP. In *Proc. of 10th International Conference on Logic Programming*, Budapest, Hungary, June 21–25, 1993, 737–753.

Maher, M. J. 2002. Analysis of a global contiguity constraint. In *Proc. of Workshop on Rule-Based Constraint Reasoning and Programming*.

Maher, M. J. 2009a. Local consistency for extended CSPs. *Theoretical Computer Science 410,* 46, 4769–4783.

Maher, M. J. 2009b. Open constraints in a boundable world. In *Proc. of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, Pittsburgh, PA, USA, May 27–31, 2009, 163–177.

Maher, M. J. 2009c. Open contractible global constraints. In *Proc. of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, California, USA, July 11-17, 2009, 578–583.

Maher, M. J. 2009d. SOGgy constraints: Soft open global constraints. In *Proc. of Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009*, Lisbon, Portugal, September 20–24, 2009, 584–591.

Maher, M. J. 2010. Contractibility and contractible approximations of soft global constraints. In *Proc. of Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010*, July 16–19, 2010, Edinburgh, Scotland, UK, 114–123.

Maher, M. J., Narodytska, N., Quimper, C. and Walsh, T. 2008. Flow-based propagators for the SEQUENCE and related global constraints. In *Proc. of Principles and Practice of Constraint Programming, 14th International Conference, CP 2008*, Sydney, Australia, September 14–18, 2008, 159–174.

Maher, M. J. and Stuckey, P. J. 1989. Expanding query power in constraint logic programming languages. In *Proc. of the North American Conference 1989*, Cleveland, Ohio, USA, October 16–20, 1989, vol. 2, 20–36.

Métivier, J., Boizumault, P. and Loudni, S. 2007. All different: Softening alldifferent in weighted CSPs. In *Proc. of 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, October 29–31, 2007, Patras, Greece, vol. 1, 223–230.

Métivier, J., Boizumault, P. and Loudni, S. 2009. Softening GCC and regular with preferences. In *Proc. of the 2009 ACM Symposium on Applied Computing (SAC)*, Honolulu, Hawaii, USA, March 9–12, 2009, 1392–1396.

Mittal, S. and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proc. of the 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, July 29–August 3, 1990, vol. 2, 25–32.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J. and Tack, G. 2007. Minizinc: Towards a standard CP modelling language. In *Proc. of Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007*, Providence, RI, USA, September 23–27, 2007, 529–543.

Pachet, F. and Roy, P. 1999. Automatic generation of music programs. In *Proc. of Principles and Practice of Constraint Programming - CP'99, 5th International Conference*, Alexandria, Virginia, USA, October 11–14, 1999, 331–345.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *Proc. of Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004*, Toronto, Canada, September 27 - October 1, 2004, 482–495.

Petit, T. and Poder, E. 2009. The soft cumulative constraint. *CoRR abs/0907.0939*.

Petit, T., Régin, J. and Bessière, C. 2001. Specific filtering algorithms for over-constrained problems. In *Proc. of Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001*, Paphos, Cyprus, November 26–December 1, 2001, 451–463.

Quimper, C. and Walsh, T. 2006. Global grammar constraints. In *Proc. of Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006*, Nantes, France, September 25–29, 2006, 751–755.

Régin, J. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proc. of the 12th National Conference on Artificial Intelligence*, Seattle, WA, USA, July 31–August 4, 1994, vol. 1, 362–367.

Régin, J. 1996. Generalized arc consistency for global cardinality constraint. In *Proc. of the 13th National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96*, Portland, Oregon, August 4–8, 1996, vol. 1, 209–215.

Rossi, F., van Beek, P. and Walsh, T., Eds. 2006. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence, vol. 2. Elsevier.

Schiex, T., Fargier, H. and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95*, Montréal Québec, Canada, August 20–25 1995, vol. 2, 631–639.

Schulte, C. and Tack, G. 2009. Weakly monotonic propagators. In *Proc. of Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009*, Lisbon, Portugal, September 20–24, 2009, 723–730.

Sellmann, M. 2006. The theory of grammar constraints. In *Proc. of Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006*, Nantes, France, September 25-29, 2006, 530–544.

van Hoeve, W. J., Pesant, G. and Rousseau, L. 2006. On global warming: Flow-based soft global constraints. *Journal of Heuristics 12,* 4-5, 347–373.

van Hoeve, W. J. and Régin, J. 2006. Open constraints in a closed world. In *Proc. of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 3rd International Conference, CPAIOR 2006*, Cork, Ireland, May 31–June 2, 2006, 244–257.

Verfaillie, G. and Jussien, N. 2005. Constraint solving in uncertain and dynamic environments: A survey. *Constraints 10,* 3, 253–281.