J Heuristics manuscript No.
(will be inserted by the editor)

Underestimating the Cost of a Soft Constraint is Dangerous:

Revisiting the Edit-Distance Based Soft Regular Constraint

Jun He · **Pierre Flener** · **Justin Pearson**

Received: 2012 Feburary / Accepted: 2013 May

Abstract Many real-life problems are over-constrained, so that no solution satisfying all their constraints exists. Soft constraints, with costs denoting how much the constraints are violated, are used to solve these problems. We use the edit-distance based SOFTREGULAR constraint as an example to show that a propagation algorithm that sometimes underestimates the cost may guide the search to incorrect (non-optimal) solutions to an over-constrained problem. To compute correctly the cost for the edit-distance based SOFTREGULAR constraint, we present a quadratic-time propagation algorithm based on dynamic programming and a proof of its correctness. We also give an improved propagation algorithm using an idea of computing the edit distance between two strings, which may also be applied to other constraints with propagators based on dynamic programming. The asymptotic time complexity of our improved propagator is always at least as good as the one of our quadratic-time propagator, but significantly better when the edit distance is small. Our propagators achieve domain consistency on the problem variables and bounds consistency on the cost variable. Our method can also be adapted for the violation measure of the edit-distance based REGULAR constraint for constraint-based *local* search.

Keywords Constraint programming · Soft regular constraint · Edit distance · Network flows · Dynamic programming

Jun He (✉)
Department of Information Technology, Uppsala University, 751 05 Uppsala, Sweden
School of Information System and Managemnet, National Univeristy of Defense Technology, 410073 Changsha, Hunan, China
E-mail: hejunnudt@gmail.com

Pierre Flener
Department of Information Technology, Uppsala University, 751 05 Uppsala, Sweden
E-mail: Pierre.Flener@it.uu.se

Justin Pearson
Department of Information Technology, Uppsala University, 751 05 Uppsala, Sweden
E-mail: Justin.Pearson@it.uu.se

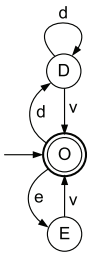

Figure 1 A DFA for a simple work scheduling constraint with three states and five transitions over an alphabet of three letters

1 Introduction

In constraint programming (CP), soft constraints provide a natural way to solve over-constrained problems, by allowing constraints to be partially satisfied. A soft constraint is allowed to be violated and is obtained by adding to the original constraint a cost variable, which represents how much that constraint is violated. When soft constraints are used, an optimal solution that violates the soft constraints as little as possible, while satisfying the other constraints, is to be found by the CP solver.

The REGULAR constraint (Pesant 2004; a generalisation of which is also known as the AUTOMATON constraint, see Beldiceanu et al. 2004) is defined as REGULAR(X,M), where $X = \langle x_1, \ldots, x_n \rangle$ is a sequence of n decision variables; and M is a deterministic finite automaton (DFA). A DFA M is defined as $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a finite set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states.

For example, Figure 1 gives a DFA M that describes a simple work scheduling constraint for one employee. There are values for two work shifts, namely day (d) and evening (e), as well as a value for enjoying a one-day vacation (v). Shift sequences are subject to the following four constraints: one must start with a work shift, and must end with some vacation; one must enjoy some vacation before a change of work shift; one cannot enjoy a vacation of more than one day; and one must enjoy a vacation after working an evening. The start state O is marked by a transition entering from nowhere, while state O is also the unique accepting state and is marked by a double circle. Missing transitions, say from state D upon reading value e, are assumed to go to an implicit failure state, with a self-loop transition for every symbol of the alphabet (so that no accepting state is reachable from it). The set of words accepted by M defines the set of acceptable shift sequences for one employee, e.g., the set of acceptable shift sequences of length 5 is $\{$ddddv, ddvdv, ddvev, dvddv, evddv$\}$.

The SOFTREGULAR constraint is the softened version of the REGULAR constraint, and is defined as SOFTREGULAR(X,M,z), where z is the cost variable. There are two versions of the SOFTREGULAR constraint (van Hoeve et al. 2004, 2006), namely the Hamming-distance based and edit-distance based SOFTREGULAR constraints, based on two different cost measures. The edit distance (also known as Levenshtein distance) between two words is the minimum number of non-copying edit operations (namely substitution, insertion, and deletion of a letter) needed to transform one word into

the other. Compared with the Hamming-distance based cost measure (where only substitution is allowed) for the SOFTREGULAR constraint, the edit-distance based cost measure, which is the minimum edit distance between any possible assignment of X under the current domains and the words of length $|X|$ of the regular language of M (see Definition 10 on page 365 of van Hoeve et al. 2006), is argued to be more suitable for scheduling problems in (van Hoeve et al. 2004, 2006). For example, evddv is a word accepted by the DFA of Figure 1, but not devdd. The Hamming distance between the two words is 4, but their edit distance is only 2 since we can delete the 'd' at the beginning of the second word and insert a 'v' at its end. In this paper, we are only concerned with the edit-distance based SOFTREGULAR constraint, hence whenever the SOFTREGULAR constraint is mentioned in the rest of the paper, we mean the edit-distance based SOFTREGULAR constraint.

The authors of (van Hoeve et al. 2004, 2006) represent soft constraints with weighted flow networks, and then introduce a generic propagator based on computing flows, with the precondition that every integer source-to-sink flow *necessarily* represents a solution to the constraint and that the value of a minimum-weight flow is *exactly* the cost measure of the constraint (see Algorithm 1 on page 354 of van Hoeve et al. 2006). Note that whenever a flow is mentioned in this paper, we mean a minimum-weight maximum integer flow from the source to the sink; and we use propagator to mean a propagation algorithm in the rest of this paper.

Consider a SOFTREGULAR(X,M,z) constraint, where X is a sequence of $|X| = n$ decision variables and $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ is a DFA. The authors of (van Hoeve et al. 2004, 2006) introduce a flow network representation of the SOFTREGULAR(X,M,z) constraint and a propagator that implements the generic propagator based on topological sort with table lookups (on pages 368 and 369 of van Hoeve et al. 2006). For each decision variable x_i in X and each value v in the domain of x_i, the propagator computes a flow (of value 1) that passes an arc related with the assignment $x_i := v$, and taking $O((n+|Q|)\cdot|\delta|)$ time with $O(n\cdot(|\delta|+|Q|)+|Q|^2) = O(n\cdot|\delta|+|Q|^2)$ space (namely $O(n\cdot|\delta|)$ space to store the flow network, $n\cdot|Q|$ space to compute minimum-weight flows, $|Q|^2$ space to store the shortest distance between any two states in Q, and $|\delta| = |Q|\cdot|\Sigma| \geq |Q|$). Note that we use binding to mean an assignment of a value to *just* one decision variable in the rest of this paper, hence an assignment of X is a set of $|X| = n$ bindings. However the constructed input flow network (on page 368 of van Hoeve et al. 2006) for the propagator is not suitable for the reason that the flows represent words in the *whole* regular language underlying the constraint instead of the n-letter regular language (here n-letter regular language denotes the sub-language of words of length n of the regular language), as we show in Section 4, and a propagator with such an unsuitable flow network may thus *underestimate* the cost measure. Hence the propagator cannot be used as in Corollary 6 (on page 368 van Hoeve et al. 2006), because its precondition is violated.

Furthermore, as we show in Section 4 for the SOFTREGULAR constraint, using a propagator that sometimes underestimates the cost measure has an unwanted consequence, as the propagator may guide the search to incorrect (non-optimal) solutions to an over-constrained problem. Hence we argue that it is crucial for a propagator for a soft constraint to compute the exact cost measure.

The rest of the paper is organised as follows:

– Section 2 gives some background on CP.
– Section 3 gives a brief review of the weighted flow network representation of the SOFTREGULAR constraint.
– Section 4 shows (by example) the danger of underestimating the cost measure for a soft constraint, namely missed propagations leading to incorrect (non-optimal) solutions being found. Another example is given in Appendix A, based on a reviewer error rather than an error in a publication.
– Section 5 presents our quadratic-time propagator for the SOFTREGULAR constraint based on dynamic programming instead of flow theory, as well as a proof of its correctness, and then gives an improved propagator, the asymptotic time complexity of which is always at least as good as the one of our quadratic-time propagator, but significantly better when the edit distance is small. Our propagators achieve domain consistency on the decision variables X and bounds consistency on the cost variable z.
– Section 6 theoretically compares our propagators with two other propagators that we propose, one based on the propagator of (van Hoeve et al. 2004, 2006), the other based on the propagator for the WEIGHTEDGRAMMAR constraint (Katsirelos et al. 2008, 2011).
– Section 7 demonstrates the efficiency of our propagators with some experiments.
– Section 8 shows how to adapt our method for the violation measure of an edit-distance based REGULAR constraint for constraint-based *local* search (CBLS).
– Section 9 summarises this work.

2 Background

We first give some background material on constraint programming (CP, e.g., see Apt 2003), which is a declarative paradigm to model and solve combinatorial problems.

2.1 Constraints and Decision Variables

In CP, a problem is modelled by a set of constraints. Let $X = \langle x_1, \ldots, x_n \rangle$ be a sequence of n decision variables, where the domain of a decision variable x_i (for all $x_i \in X$) is a finite set of values that can be assigned to x_i and is denoted by $\mathrm{dom}(x_i)$. A constraint C on X is usually specified by an intensionally defined subset of the Cartesian product of the domains of all decision variables in X: $C \subseteq \mathrm{dom}(x_1) \times \cdots \times \mathrm{dom}(x_n)$. An assignment $X := \langle v_1, \ldots, v_n \rangle \in C$ is called a *solution* to C, and is called a *solution to a problem* if and only if it is a solution to all constraints of the problem.

2.2 Search and Propagation

In CP, a problem is solved by exploring a search tree, where all possible variable-value combinations in the domains are intelligently enumerated until a solution to the problem is found or it is proved that none exists. At each node of the search tree, constraint propagation is performed separately for all constraints in the problem to

remove some (but not necessarily all) inconsistent values, which cannot be part of a solution to the constraint, from the domains, and is repeated until no more pruning is possible (a fix point). Hence, each constraint is associated with a propagator for this purpose.

2.3 Consistency

To solve a problem efficiently in CP, one objective is to construct a small search tree, hence the propagator should remove as many inconsistent values from the domains as possible; the other objective is to design low-complexity propagators, as propagators are called many times during search. However the two objectives are conflicting, as a propagator that can remove more values from the domains is usually of higher complexity. This motivates the introduction of levels of consistency. We give definitions of the two levels of consistency that are used in this paper.

Definition 1 (Domain Consistency) Given a sequence $X = \langle x_1, \ldots, x_n \rangle$ of n decision variables and a constraint C on X, we say that the $\mathrm{dom}(x_i)$ are *domain consistent* if for each $1 \leq i \leq n$ and each value $v_i \in \mathrm{dom}(x_i)$, there exist values $d_j \in \mathrm{dom}(x_j)$ for all $j \neq i$ such that $\langle d_1, \ldots, d_{i-1}, v_i, d_{i+1}, \ldots, d_n \rangle \in C$.

Definition 2 (Bounds Consistency) Given a sequence $X = \langle x_1, \ldots, x_n \rangle$ of n decision variables and a constraint C on X, we say that the $\mathrm{dom}(x_i)$ are *bounds consistent* if for each $1 \leq i \leq n$ and each value $v_i \in \{\min \mathrm{dom}(x_i), \max \mathrm{dom}(x_i)\}$, there exist values $d_j \in [\min \mathrm{dom}(x_j), \max \mathrm{dom}(x_j)]$ for all $j \neq i$ such that $\langle d_1, \ldots, d_{i-1}, v_i, d_{i+1}, \ldots, d_n \rangle \in C$.

Note that domain consistency is a stronger level of consistency than bounds consistency, as domain consistency checks every value in every domain while the latter only checks the lower and upper bound values.

3 A Flow Network Representation of the SoftRegular Constraint

Given a $\textsc{SoftRegular}(X, M, z)$ constraint, where $X = \langle x_1, \ldots, x_n \rangle$ is a sequence of $|X| = n$ decision variables, and $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ is a DFA, the constraint is represented with a weighted flow network (van Hoeve et al. 2004, 2006). In the flow network (an example is given in Figure 2), there is a source S and a sink T. Between S and T, there are $n+1$ vertical layers of nodes, where each layer has a node for each state in Q. A node labelled with state $q_k \in Q$ in layer j is named q_k^j. Let node q_0^0 (recall that q_0 is the start state) in layer 0 be called the *start node*; let each node q_k^n (for $q_k \in F$) in layer n be called an *accepting node*.

There are four arc sets in the flow network, depending on the current domains of the decision variables: the copy arc set A_{copy} (which is called A in van Hoeve et al. 2004, 2006), the substitution arc set A_{sub}, the insertion arc set A_{ins}, and the deletion arc set A_{del}. Every arc in $A_{\mathrm{sub}} \cup A_{\mathrm{ins}} \cup A_{\mathrm{del}}$ has a weight of 1, while every arc in A_{copy} has a weight of 0. All arcs have capacity 1. For each arc set, we give the original

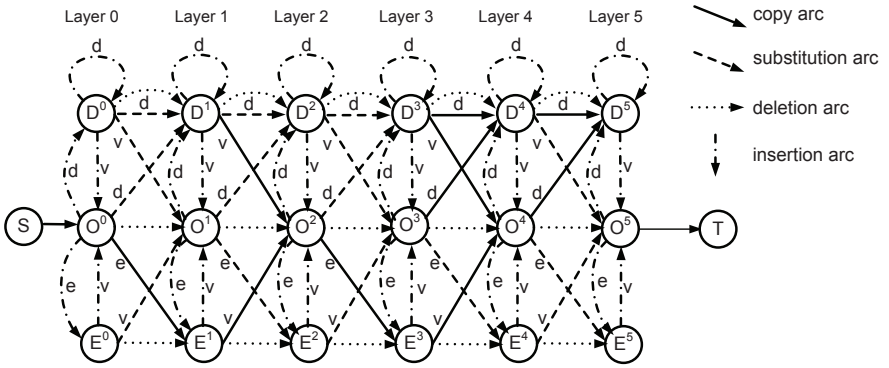

Figure 2 The revised flow network representation of the SOFTREGULAR(X,M,z) constraint for a sequence $X = \langle x_1,\ldots,x_5\rangle$ of 5 decision variables, with current domains $\mathrm{dom}(x_1) = \mathrm{dom}(x_3) = \{\mathrm{e}\}$, $\mathrm{dom}(x_2) = \{\mathrm{v}\}$, $\mathrm{dom}(x_4) = \{\mathrm{d},\mathrm{v}\}$, and $\mathrm{dom}(x_5) = \{\mathrm{d}\}$; where $M = \langle Q,\Sigma,\delta,q_0,F\rangle$ is the DFA depicted in Figure 1; and z is the cost variable. Node S is the source, and node T is the sink. There are $|X|+1 = 6$ vertical layers of nodes between S and T, where each layer has a node for each state in Q. A node labelled with state $q_k \in Q$ in layer j is named q_k^j, e.g., D^2 denotes the node labelled with state D in layer 2. There are four kinds of arcs in the flow network: copy arcs (solid arcs), substitution arcs (purple dashed arcs), deletion arcs (red dotted arcs), and insertion arcs (green dash-dotted arcs). All arcs have capacity 1; each solid arc has a weight of 0; each non-solid arc has a weight of 1. Note that the letters for the arcs are *not* part of the flow network, but show how the network was constructed.

definition of (van Hoeve et al. 2004, 2006) and revise it in a way that will be useful in the rest of the paper (but not for our propagator):

– An arc for symbol $\sigma \in \Sigma$ at position i is in the *copy arc* set A_{copy} if it is used when measuring the edit distance to a word where letter i is a copy of σ (i.e., $i = \sigma$). Formally, A_{copy} is made up of three disjoint arc subsets containing the following arcs respectively: the arc from the source S to the start node q_0^0; every arc from node q_k^{i-1} to node q_ℓ^i satisfying $\delta(q_k,t) = q_\ell$ with some value $t \in \mathrm{dom}(x_i)$; and every arc from any accepting node to the sink T.

$$A_{\mathrm{copy}} = \\ \left\{(\mathrm{S},q_0^0)\right\} \cup \bigcup_{i=1}^{n} \left\{(q_k^{i-1},q_\ell^i) \mid \exists t \in \mathrm{dom}(x_i) : \delta(q_k,t) = q_\ell\right\} \cup \left\{(q_k^n,\mathrm{T}) \mid q_k \in F\right\}$$

Hence, each integer flow that *only* passes arcs in A_{copy} represents a solution to the *hard* REGULAR(X,M) constraint.

– An arc for symbol $\sigma \in \Sigma$ at position i is in the *substitution arc* set A_{sub} if it is used when measuring the edit distance to a word where letter i is substituted by σ. Formally, A_{sub} contains every arc not in A_{copy} that goes from node q_k^{i-1} to node q_ℓ^i satisfying $\delta(q_k,t) = q_\ell$ with some value $t \in \Sigma$:

$$A_{\mathrm{sub}} = \bigcup_{i=1}^{n} \left\{(q_k^{i-1},q_\ell^i) \mid \exists t \in \Sigma : \delta(q_k,t) = q_\ell\right\} \setminus A_{\mathrm{copy}}$$

This definition is different from the one in (van Hoeve et al. 2006) (but the same as in van Hoeve et al. 2004), where the 0-weight arcs in A_{copy} are not excluded

from A_{sub} even though they are superfluous. Indeed, if a flow passes a substitution arc α that is a duplicate of an arc α' in A_{copy}, then we can get another flow of the same maximum value but with a smaller weight by just replacing α with α'. Therefore substitution arcs that are duplicate of arcs in A_{copy} cannot belong to any minimum-weight maximum flow, and we can safely remove them from A_{sub}.

– An arc for symbol $\sigma \in \Sigma$ at position i is in the *insertion arc* set A_{ins} if it is used when measuring the edit distance to a word where σ is inserted after position i. Formally, A_{ins} contains every intra-layer arc from node q_k^i to node q_ℓ^i satisfying $\delta(q_k,t) = q_\ell$ with some value $t \in \Sigma$:

$$A_{\text{ins}} = \bigcup_{i=0}^{n} \left\{ (q_k^i, q_\ell^i) \mid \exists t \in \Sigma : \delta(q_k,t) = q_\ell \right\}$$

This definition is different from the one in (van Hoeve et al. 2006) (but the same as in van Hoeve et al. 2004), where the self-loops are excluded from A_{ins}. A counterexample is given in Path 1 of Section 5.1, where a self-loop insertion arc must be used.

– An arc for symbol $\sigma \in \Sigma$ at position i is in the *deletion arc* set A_{del} if it is used when measuring the edit distance to a word where σ is deleted from position i. Formally, A_{del} contains *every* arc from node q_k^{i-1} to node q_k^i (for $q_k \in Q$):

$$A_{\text{del}} = \bigcup_{i=1}^{n} \left\{ (q_k^{i-1}, q_k^i) \mid q_k \in Q \right\}$$

This definition is different from the one in (van Hoeve et al. 2004, 2006), where the arcs in A_{copy} are excluded from A_{del}. A counterexample is given in Path 2 of Section 5.1, where a deletion arc that has a duplicate copy arc must be used.

Note that the flow network is domain-specific, as the arc sets A_{copy} and A_{sub} change incrementally upon propagation, and that arcs only move from A_{copy} to A_{sub}, but never otherwise.

For example, Figure 2 gives the flow network representation of the SoftRegular (X, M, z) constraint for a sequence $X = \langle x_1, \ldots, x_5 \rangle$ of 5 decision variables, with current domains $\text{dom}(x_1) = \text{dom}(x_3) = \{\text{e}\}$, $\text{dom}(x_2) = \{\text{v}\}$, $\text{dom}(x_4) = \{\text{d,v}\}$, and $\text{dom}(x_5) = \{\text{d}\}$, where M is the DFA depicted in Figure 1.

4 An Example of an Incorrectly Computed Solution

The authors of (van Hoeve et al. 2004, 2006) represent soft constraints with weighted flow networks, and then introduce a generic propagator based on computing flows (see Algorithm 1 on page 354 of van Hoeve et al. 2006), with the precondition that every integer flow *necessarily* represents a solution to the constraint and that the value of a minimum-weight flow is *exactly* the cost measure of the constraint. Given a SoftRegular constraint with the unrevised flow network representation of Section 3, a propagator that implements the generic propagator based on topological sort with table lookups is introduced. However, the input flow network for the propagator is not suitable, as the flows may pass different numbers of insertion arcs and deletion

arcs. The propagator with the unsuitable flow network computes the minimum edit distance between any possible assignment of X under the current domains and the *whole* regular language instead of the n-letter regular language, and may thus underestimate the cost measure. Now, we use the SOFTREGULAR(X,M,z) constraint as an example to show that using a propagator that sometimes underestimates the cost measure has an unwanted consequence, as the propagator may guide the search to incorrect (non-optimal) solutions to an over-constrained problem.

Consider the following over-constrained problem P:

- There is a sequence of 5 decision variables $X = \langle x_1, \ldots, x_5 \rangle$, with the initial domains $\mathrm{dom}(x_1) = \mathrm{dom}(x_3) = \{\mathrm{e}\}$, $\mathrm{dom}(x_2) = \{\mathrm{v}\}$, $\mathrm{dom}(x_4) = \{\mathrm{d}, \mathrm{v}\}$, and $\mathrm{dom}(x_5) = \{\mathrm{d}\}$.
- There is only one constraint, namely a SOFTREGULAR(X,M,z) constraint, where M is the DFA depicted in Figure 1, and z is the cost variable with the initial domain $\mathrm{dom}(z) = \{0, \ldots, 2\}$.
- The problem is over-constrained, hence the objective is to find a solution that minimises z.

There are two possible assignments for X, namely $X := \langle \mathrm{e}, \mathrm{v}, \mathrm{e}, \mathrm{v}, \mathrm{d} \rangle$ and $X := \langle \mathrm{e}, \mathrm{v}, \mathrm{e}, \mathrm{d}, \mathrm{d} \rangle$. The minimum edit distance between evevd and the 5-letter regular language accepted by M (namely {ddddv, ddvdv, ddvev, dvddv, evddv}) is 3 (the edit distance to evddv); the minimal edit distance between evedd and the 5-letter regular language is 2 (the edit distance to evddv). Hence the optimal solution to P is $X := \langle \mathrm{e}, \mathrm{v}, \mathrm{e}, \mathrm{d}, \mathrm{d} \rangle$, with $z = 2$.

However when using the propagator of (van Hoeve et al. 2004, 2006) with the unsuitable flow network that may underestimate the edit-distance based cost measure, the non-optimal solution ($X := \langle \mathrm{e}, \mathrm{v}, \mathrm{e}, \mathrm{v}, \mathrm{d} \rangle$) is found. Figure 3 shows the difference of using (denoted by dashed lines) the propagator of (van Hoeve et al. 2004, 2006) with the unsuitable flow network and using (denoted by solid lines) the propagator computing the exact cost measure that will be given in Section 5 to solve P, where w denotes the edit distance between two given words. The propagator of (van Hoeve et al. 2004, 2006) with the unsuitable flow network first finds a flow for each binding. For the binding $x_4 := \mathrm{v}$, a flow of weight 1 is found, namely

$$\mathrm{S} \to \mathrm{O}^0 \xrightarrow{\text{copy 'e'}} \mathrm{E}^1 \xrightarrow{\text{copy 'v'}} \mathrm{O}^2 \xrightarrow{\text{copy 'e'}} \mathrm{E}^3 \xrightarrow{\text{copy 'v'}} \mathrm{O}^4 \xrightarrow{\text{delete 'd'}} \mathrm{O}^5 \to \mathrm{T}.$$

This flow passes one deletion arc and no insertion arcs, and measures the edit distance from evevd to the 4-letter word evev accepted by M. Recall that evevd is actually at edit distance 3 (not 1) from the 5-letter language accepted by M. For the binding $x_4 := \mathrm{d}$, a flow of weight 2 is found, namely

$$\mathrm{S} \to \mathrm{O}^0 \xrightarrow{\text{copy 'e'}} \mathrm{E}^1 \xrightarrow{\text{copy 'v'}} \mathrm{O}^2 \xrightarrow{\text{substitute 'd'}} \mathrm{D}^3 \xrightarrow{\text{copy 'd'}} \mathrm{D}^4 \xrightarrow{\text{substitute 'v'}} \mathrm{O}^5 \to \mathrm{T}.$$

This flow passes two substitution arcs, and measures the edit distance from evedd to the 5-letter word evddv accepted by M. As both of the flows have a weight not larger than max $\mathrm{dom}(z)$, which is 2, no value of $\mathrm{dom}(x_4)$ is removed; furthermore, min $\mathrm{dom}(z)$ is updated to the minimum weight of the two flows, which is 1. Next, assume without loss of generality that the CP solver enumerates $\mathrm{dom}(x_4)$. For the

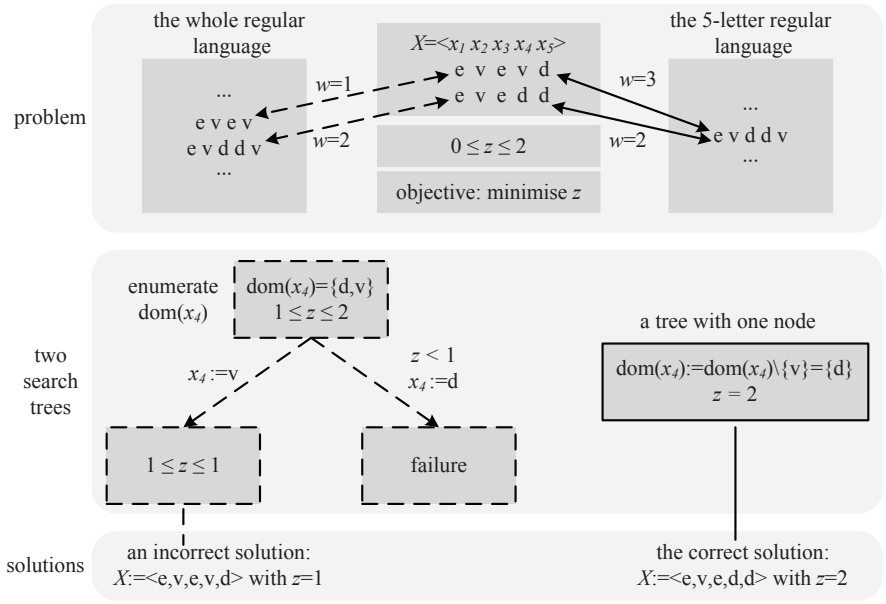

Figure 3 Comparison between two ways of solving the over-constrained problem P: one uses the propagator of (van Hoeve et al. 2004, 2006) with the unsuitable flow network, and is denoted by dashed lines; the other uses the propagator computing the exact cost measure that will be given in Section 5, and is denoted by solid lines. The symbol w denotes the edit distance between two given words

binding $x_4 :=$ v, the same flow of weight 1 is found by the propagator, and a solution, namely $X := \langle$e, v, e, v, d$\rangle$ with $z = 1$, is found. Thereafter the betterness constraint $z < 1$ is added by the solver. For the binding $x_4 :=$ d, no flow of weight less than 1 is found, hence the value d is removed from dom(x_4); since the domain of x_4 is wiped out, there is no solution in this branch and the proof of optimality is completed. Hence the CP solver finds an incorrect optimal solution minimising z, which is $X := \langle$e, v, e, v, d$\rangle$, with an incorrectly computed $z = 1$.

5 A Correct Propagator for the SoftRegular Constraint

The propagator for the SoftRegular(X, M, z) constraint with the unsuitable flow network of (van Hoeve et al. 2004, 2006) may guide the search to incorrect (non-optimal) solutions. Hence one way to fix this problem is *changing the input flow network* so that every flow *necessarily* represents an $|X|$-letter word of the regular language (as shown in Section 6.1). However, we prefer to *change the propagator* so that it computes the cost measure with the unchanged (but revised) flow network of flows representing the *whole* regular language, as the space complexity is lower and as our experimental results in Section 7 show that the new propagator works better in practice. Hence we now present, prove, analyse, and improve a new propagator for the SoftRegular constraint. Our propagators achieve domain consistency on the decision variables X and bounds consistency on the cost variable z.

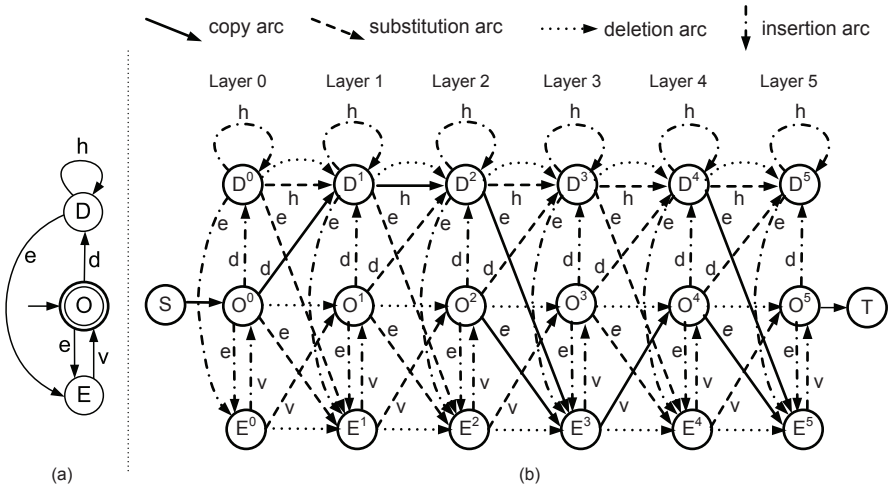

Figure 4 Subfigure (a) is a DFA (not the same as in Figure 1). Subfigure (b) is the revised digraph representation of the SOFTREGULAR(X,M,z) constraint for a sequence $X = \langle x_1,\ldots,x_5\rangle$ of 5 decision variables, with current domains $\mathrm{dom}(x_1) = \{\mathrm{d}\}$, $\mathrm{dom}(x_2) = \{\mathrm{h}\}$, $\mathrm{dom}(x_3) = \mathrm{dom}(x_5) = \{\mathrm{e}\}$, and $\mathrm{dom}(x_4) = \{\mathrm{v}\}$; where $M = \langle Q,\Sigma,\delta,q_0,F\rangle$ is the DFA depicted in subfigure (a); and z is the cost variable

5.1 Description of the Propagator

Given a SOFTREGULAR(X,M,z) constraint with $|X| = n$ decision variables and a DFA $M = \langle Q,\Sigma,\delta,q_0,F\rangle$, let G be the revised flow network (seen as a digraph now) in Section 3 with four arc sets: A_{copy}, A_{sub}, A_{ins}, and A_{del}. We introduce a propagator (see Algorithm 1) for the SOFTREGULAR constraint based on dynamic programming, which is a popular way of designing propagators (e.g., see Quimper and Walsh 2006; Kadioğlu and Sellmann 2010). Similarly to (van Hoeve et al. 2004, 2006), our propagator computes for each binding a minimum-weight path from the source S to the sink T in the digraph G, but it ensures that every computed minimum-weight path passes the same number of insertion and deletion arcs, hence it computes the cost measure, which is the minimum edit distance between any possible assignment of X under the current domains and the n-letter regular language.

Note that the revised digraph is necessary for computing a minimum-weight path that passes the same number of insertion and deletion arcs. For example, Figure 4(b) is the revised digraph representation of the SOFTREGULAR(X,M,z) constraint for a sequence $X = \langle x_1,\ldots,x_5\rangle$ of 5 decision variables, with current domains $\mathrm{dom}(x_1) = \{\mathrm{d}\}$, $\mathrm{dom}(x_2) = \{\mathrm{h}\}$, $\mathrm{dom}(x_3) = \mathrm{dom}(x_5) = \{\mathrm{e}\}$, and $\mathrm{dom}(x_4) = \{\mathrm{v}\}$; where $M = \langle Q,\Sigma,\delta,q_0,F\rangle$ is the DFA depicted in Figure 4(a); and z is the cost variable. We can find the following two minimum-weight paths that pass the same number of insertion and deletion arcs: in the first path, a self-loop insertion arc (namely $\mathrm{D}^2 \xrightarrow{\text{insert 'h'}} \mathrm{D}^2$) must be used; in the second path, a deletion arc (namely $\mathrm{D}^1 \xrightarrow{\text{delete 'h'}} \mathrm{D}^2$) that has a duplicate copy arc (namely $\mathrm{D}^1 \xrightarrow{\text{copy 'h'}} \mathrm{D}^2$) must be used.

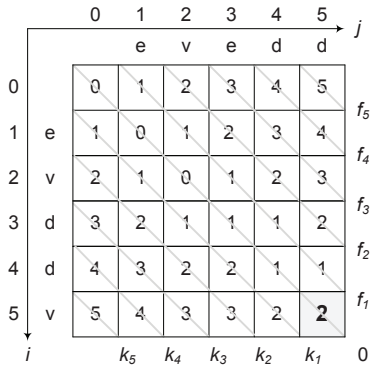

Figure 5 The matrix h for computing the edit distance $w = 2$ between the words evddv and evedd

Path 1: S $\rightarrow$ O^0 $\xrightarrow{\text{copy 'd'}}$ D^1 $\xrightarrow{\text{copy 'h'}}$ D^2 $\xrightarrow{\text{insert 'h'}}$ D^2 $\xrightarrow{\text{copy 'e'}}$ E^3 $\xrightarrow{\text{copy 'v'}}$ O^4 $\xrightarrow{\text{delete 'e'}}$ O^5 $\rightarrow$ T

Path 2: S $\rightarrow$ O^0 $\xrightarrow{\text{copy 'd'}}$ D^1 $\xrightarrow{\text{delete 'h'}}$ D^2 $\xrightarrow{\text{copy 'e'}}$ E^3 $\xrightarrow{\text{copy 'v'}}$ O^4 $\xrightarrow{\text{copy 'e'}}$ E^5 $\xrightarrow{\text{insert 'v'}}$ O^5 $\rightarrow$ T

5.1.1 Computing the Cost Measure

In order to compute the edit distance w between the words a_1 and a_2 of length n (where $a_1[i]$ is the i^{th} letter of the word a_1 and $a_1[i \dots j]$ is the subword of a_1 starting from the i^{th} letter to the j^{th} letter), Wagner and Fischer (1974) introduced a dynamic programming algorithm taking $O(n^2)$ time by computing an $(n+1) \times (n+1)$ matrix h row by row as follows:

$$h[i,j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{pmatrix} h[i-1,j-1] + \textbf{if } a_1[i] = a_2[j] \textbf{ then } 0 \textbf{ else } 1, \\ h[i,j-1]+1, \\ h[i-1,j]+1 \end{pmatrix} & \text{otherwise} \end{cases}$$

where $h[i,j]$ denotes the edit distance between the subwords $a_1[1 \dots i]$ and $a_2[1 \dots j]$, so that $w = h[n,n]$ (the value of the cell in the lower-right corner) is the edit distance between the words a_1 and a_2. For example, Figure 5 gives the matrix h when computing the edit distance $w = 2$ between the words evddv and evedd.

Similarly, Algorithm 1 computes a matrix $c[0..n, 0..n, Q]$ using a dynamic programming algorithm (lines 4 to 15). The matrix c has one more dimension than h, and any cell $h[i,j]$ is represented by $|Q|$ cells in c (namely $\{c[i,j,q_\ell] \mid q_\ell \in Q\}$), with $c[i,j,q_\ell]$ denoting the minimum edit distance between any possible assignment of $\langle x_1, \dots, x_i \rangle$ under the current domains and any word of j symbols accepted by M from state q_0 upon passing j transitions to state q_ℓ. Here only the $c[i,j,q_\ell]$ values

Algorithm 1 A propagator computing the cost measure u for the SOFTREGULAR(X,M,z) constraint, with $|X| = n$ decision variables, a DFA $M = \langle Q,\Sigma,\delta,q_0,F\rangle$, and the cost variable z

1: **global variable**: G stores the digraph representing the SOFTREGULAR(X,M,z) constraint
2: **global variable**: **set** $reach[i$ **in** $0..n]$ stores the set of states in Q reachable from S in layer i of G
3: **function** *propagator*(SOFTREGULAR$(X,M,z),G$)
4: **int** $c[i$ **in** $0..n, j$ **in** $0..n, q_\ell$ **in** $Q] \leftarrow n$
5: $c[i$ **in** $0..n, 0, q_0] \leftarrow i$; $c[0, j$ **in** $1..n, q_\ell$ **in** $reach[j]] \leftarrow j$
6: **for all** $i \leftarrow 1$ **to** n **do**
7: **for all** $j \leftarrow 1$ **to** n **do**
8: **for all** arc $(q_k^{j-1}, q_\ell^j) \in A_{\text{copy}}$ such that $q_k \in reach[j-1]$ **do**
9: $c[i,j,q_\ell] \leftarrow \min(c[i-1,j-1,q_k], c[i,j,q_\ell])$
10: **for all** arc $(q_k^{j-1}, q_\ell^j) \in A_{\text{sub}}$ such that $q_k \in reach[j-1]$ **do**
11: $c[i,j,q_\ell] \leftarrow \min(c[i-1,j-1,q_k]+1, c[i,j,q_\ell])$
12: **for all** arc $(q_k^{j-1}, q_\ell^{j-1}) \in A_{\text{ins}}$ such that $q_k \in reach[j-1]$ **do**
13: $c[i,j,q_\ell] \leftarrow \min(c[i,j-1,q_k]+1, c[i,j,q_\ell])$
14: **for all** arc $(q_\ell^{j-1}, q_\ell^j) \in A_{\text{del}}$ such that $q_\ell \in reach[j]$ **do**
15: $c[i,j,q_\ell] \leftarrow \min(c[i-1,j,q_\ell]+1, c[i,j,q_\ell])$
16: **int** $u \leftarrow \min\{c[n,n,q_\ell] \mid q_\ell \in F \cap reach[n]\}$
17: **if** $u > \min$ dom(z) **then**
18: min dom$(z) \leftarrow u$
19: **if** $u > \max$ dom(z) **then**
20: **return fail**
21: **else if** $u+1 \leq \max$ dom(z) **then**
22: **return succeed**
23: **bool** $r[i$ **in** $0..n, j$ **in** $0..n, q_\ell$ **in** $Q] \leftarrow$ **false**
24: **for all** state $q_\ell \in F \cap reach[n]$ **do**
25: **if** $c[n,n,q_\ell] = u$ **then**
26: $r[n,n,q_\ell] \leftarrow$ **true**
27: **set** $s[i$ **in** $1..n] \leftarrow \emptyset$
28: **for all** $i \leftarrow n$ **to** 1 **do**
29: **for all** $j \leftarrow n$ **to** 1 **do**
30: **for all** arc $(q_k^{j-1}, q_\ell^j) \in A_{\text{copy}}$ such that $q_k \in reach[j-1]$ **do**
31: **if** $r[i,j,q_\ell]$ **and** $c[i,j,q_\ell] = c[i-1,j-1,q_k]$ **then**
32: $r[i-1,j-1,q_k] \leftarrow$ **true**; $s[i] \leftarrow s[i] \cup \{$the value labelled on the arc$\}$
33: **for all** arc $(q_k^{j-1}, q_\ell^j) \in A_{\text{sub}}$ such that $q_k \in reach[j-1]$ **do**
34: **if** $r[i,j,q_\ell]$ **and** $c[i,j,q_\ell] = c[i-1,j-1,q_k]+1$ **then**
35: $r[i-1,j-1,q_k] \leftarrow$ **true**; $s[i] \leftarrow$ dom(x_i)
36: **for all** arc $(q_k^{j-1}, q_\ell^{j-1}) \in A_{\text{ins}}$ such that $q_k \in reach[j-1]$ **do**
37: **if** $r[i,j,q_\ell]$ **and** $c[i,j,q_\ell] = c[i,j-1,q_k]+1$ **then**
38: $r[i,j-1,q_k] \leftarrow$ **true**
39: **for all** arc $(q_\ell^{j-1}, q_\ell^j) \in A_{\text{del}}$ such that $q_\ell \in reach[j]$ **do**
40: **if** $r[i,j,q_\ell]$ **and** $c[i,j,q_\ell] = c[i-1,j,q_\ell]+1$ **then**
41: $r[i-1,j,q_\ell] \leftarrow$ **true**; $s[i] \leftarrow$ dom(x_i)
42: **for all** $i \leftarrow 1$ **to** n **do**
43: dom$(x_i) \leftarrow s[i]$
44: **return succeed**

with $q_\ell \in reach[j]$ are interesting, as there is no word of j symbols accepted by M from q_0 passing j transitions to q_ℓ for any $q_\ell \notin reach[j]$, hence c is a sparse matrix unlike h. The global variable $reach[0..n]$ is a vector of sets of states, with $reach[i]$ denoting the set of states of M labelled on the nodes in layer i of G that can be reached from the start state q_0 through i transitions of M. For example, in Figure 2,

Algorithm 2 Computing the vector *reach* for the SOFTREGULAR(X,M,z) constraint, with $|X| = n$ decision variables, a DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, and the cost variable z

```
1: global variable: G stores the digraph representing the SOFTREGULAR(X,M,z) constraint
2: global variable: set reach[i in 0..n] stores the set of states in Q reachable from S in layer i of G
3: procedure comp_reach(M)
4: reach[0] ← {q_0}
5: for all i ← 1 to n do
6:     reach[i] ← ∅
7:     for all state q_k ∈ reach[i−1] do
8:         for all transition δ(q_k, v) = q_ℓ do
9:             reach[i] ← reach[i] ∪ {q_ℓ}
```

$reach[0] = \{\text{O}\}$, $reach[1] = \{\text{D, E}\}$, and $reach[2] = \cdots = reach[5] = \{\text{O, D, E}\}$. The vector *reach* only needs to be computed once before the first call of the propagator, by exploring at most n times all transitions of M (as shown in Algorithm 2), and it never changes during propagation and search. First, the matrix c is created and initialised (line 4), and then each cell of c is computed similarly to h (lines 5 to 15). Note that, for any word in the n-letter regular language, the DFA M recognises the word as a sequence of $n+1$ states in Q, hence the minimum edit distance u between any possible assignment of X under the current domains and the n-letter regular language accepted by M, is assigned the minimum among the $c[n,n,q_\ell]$ with $q_\ell \in reach[n] \cap F$ (line 16).

5.1.2 Removing Inconsistent Values

Considering a binding $x_i := t$ for a decision variable x_i (with $1 \leq i \leq n$) and a value $t \in \text{dom}(x_i)$, we say that a path from the source S to the sink T is *minimum-weight-related with the binding* if the following conditions are satisfied:

- The path passes the same number of insertion and deletion arcs, and thus represents a word in the n-letter regular language accepted by M.
- The path passes an arc representing the binding.
- The path has a weight that is the minimum edit distance between any assignment to X (with $x_i := t$) and the n-letter regular language accepted by M.

Lemma 1 *The* SOFTREGULAR(X,M,z) *constraint is domain consistent on X (and is bounds consistent on z) if and only if*

1. *For every binding, a minimum-weight-related path has a weight not larger than max dom(z).*
2. *The minimum weight of all such paths is not larger than min dom(z).*

Proof: The result follows from the theorem on domain consistency for soft constraints (Theorem 2 on page 354 of van Hoeve et al. 2006). □

Note that for a soft constraint, the objective is to minimise its cost, hence only the lower bound on the cost is considered.

Revisit the example in Figure 2, where the minimum edit distance between any possible assignment of X under the current domains and the 5-letter regular language

accepted by M is 2. We find that every minimum-weight-related path for every binding has a weight of either 2 or $2+1=3$. For example, the minimum-weight-related path for $x_4 := d$, namely

$$\mathrm{S} \rightarrow \mathrm{O}^0 \xrightarrow{\text{copy 'e'}} \mathrm{E}^1 \xrightarrow{\text{copy 'v'}} \mathrm{O}^2 \xrightarrow{\text{substitute 'd'}} \mathrm{D}^3 \xrightarrow{\text{copy 'd'}} \mathrm{D}^4 \xrightarrow{\text{substitute 'v'}} \mathrm{O}^5 \rightarrow \mathrm{T},$$

has a weight of 2; and the minimum-weight-related path for $x_4 := v$, namely

$$\mathrm{S} \rightarrow \mathrm{O}^0 \xrightarrow{\text{copy 'e'}} \mathrm{E}^1 \xrightarrow{\text{copy 'v'}} \mathrm{O}^2 \xrightarrow{\text{substitute 'd'}} \mathrm{D}^3 \xrightarrow{\text{substitute 'd'}} \mathrm{D}^4 \xrightarrow{\text{substitute 'v'}} \mathrm{O}^5 \rightarrow \mathrm{T},$$

has a weight of 3. We have the following lemma:

Lemma 2 *Given the minimum edit distance u between any possible assignment of X under the current domains and the n-letter regular language accepted by M, every minimum-weight-related path for every binding has a weight of either u or $u+1$.*

Proof: There exist a word $\langle k_1,\ldots,k_n\rangle$ in the n-letter regular language accepted by M and an assignment $X := \langle a_1,\ldots,a_n\rangle$ (with $a_i \in \mathrm{dom}(x_i)$), such that the edit distance between $\langle k_1,\ldots,k_n\rangle$ and $\langle a_1,\ldots,a_n\rangle$ is exactly u. Give any binding $x_i := t$ (with $t \in \mathrm{dom}(x_i)$), if $t = a_i$ then the edit distance between $\langle k_1,\ldots,k_n\rangle$ and $\langle a_1,\ldots,a_{i-1},t,a_{i+1},\ldots,a_n\rangle$ is exactly u; otherwise the edit distance is at most $u+1$ by substituting t with a_i first. □

Similarly to the propagator for the SoftGCC constraint (Zanarini et al. 2006), using Lemmas 1 and 2, Algorithm 1 removes inconsistent values (if necessary) from the domains for achieving domain consistency on X (and bounds consistency on z) for the SoftRegular(X,M,z) constraint (lines 17 to 43), after computing the cost measure u. If $u > \min\ \mathrm{dom}(z)$, then $\min\ \mathrm{dom}(z)$ is updated to u (lines 17 and 18) so that the second condition in Lemma 1 is satisfied. If $u > \max\ \mathrm{dom}(z)$, then by Lemma 2 we have that for every binding, every minimum-weight-related path has a weight (either u or $u+1$) larger than $\max\ \mathrm{dom}(z)$, hence all values of X are inconsistent (lines 19 and 20); else if $u+1 \leq \max\ \mathrm{dom}(z)$, then by Lemma 2 we have that for every binding, all minimum-weight-related paths have a weight not larger than $\max\ \mathrm{dom}(z)$, and all values of X are domain consistent (lines 21 and 22); otherwise $u = \max\ \mathrm{dom}(z)$, a vector s of n sets is computed by tracing paths of weight u backwards (lines 23 to 41) such that $s[i]$ (for $1 \leq i \leq n$) stores all values in $\mathrm{dom}(x_i)$ that have minimum-weight-related paths of weight u, and all values not in $s[i]$ are removed from $\mathrm{dom}(x_i)$ (lines 42 and 43; by Lemma 2, they all have minimum-weight-related paths of weight $u+1$, which is larger than $\max\ \mathrm{dom}(z) = u$). The matrix r is used to trace paths of weight u backwards, with $r[i,j,q] =$ true (or false) denoting whether $c[i,j,q]$ is (or not) a support to obtain the minimum weight u.

5.2 Correctness of the Propagator

Lemma 3 *Each element $c[i,j,q_\ell]$ with $q_\ell \in reach[j]$ is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_i\rangle$ under the current domains and any word of j symbols accepted by M from q_0 passing j transitions to q_ℓ.*

Proof: (1) $c[i,0,q_0]$ is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_i\rangle$ and the empty path, thus is i by i deletions; $c[0,j,q_\ell]$ with $q_\ell \in \mathit{reach}[j]$ is the minimum edit distance between the empty sequence and any word of j symbols, thus is j by j insertions (line 5).

(2) For any $(0,0) \leq_{\text{lex}} (i_0,j_0) <_{\text{lex}} (i,j) \leq_{\text{lex}} (n,n)$, and any state $p_0 \in \mathit{reach}[j_0]$, the induction hypothesis is that $c[i_0,j_0,p_0]$ is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_{i_0}\rangle$ and any word of j_0 symbols from q_0 to p_0. For any $q_\ell \in \mathit{reach}[j]$, we prove that $c[i,j,q_\ell]$ is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_i\rangle$ and any word of j symbols from q_0 to q_ℓ. The following four cases must hold:

1. If there is a copy arc $(q_k^{j-1},q_\ell^j) \in A_{\text{copy}}$ such that $q_k \in \mathit{reach}[j-1]$, then $c[i,j,q_\ell] = c[i-1,j-1,q_k]$, as this arc has a weight of zero (no edit operation). Therefore we have $c[i,j,q_\ell] = \min\{c[i-1,j-1,q_k] \mid (q_k^{j-1},q_\ell^j) \in A_{\text{copy}} \wedge q_k \in \mathit{reach}[j-1]\}$ (lines 8 and 9). The condition $q_k \in \mathit{reach}[j-1]$ is crucial: if $q_k \notin \mathit{reach}[j-1]$, then we cannot compute $c[i,j,q_\ell]$ from $c[i-1,j-1,q_k]$, as there is no word of $j-1$ symbols accepted by M from q_0 passing $j-1$ transitions to q_k; note that $q_k \in \mathit{reach}[j-1]$ also implies $q_\ell \in \mathit{reach}[j]$, so that $c[i-1,j-1,q_k]$ and $c[i,j,q_\ell]$ are well-defined.
2. If there is a substitution arc $(q_k^{j-1},q_\ell^j) \in A_{\text{sub}}$ such that $q_k \in \mathit{reach}[j-1]$, then $c[i,j,q_\ell] = c[i-1,j-1,q_k]+1$, as this arc has a weight of one (one substitution). Therefore we have $c[i,j,q_\ell] = \min\{c[i-1,j-1,q_k]+1 \mid (q_k^{j-1},q_\ell^j) \in A_{\text{sub}} \wedge q_k \in \mathit{reach}[j-1]\}$ (lines 10 and 11). The condition $q_k \in \mathit{reach}[j-1]$ is crucial similarly to case 1.
3. If there is an insertion arc $(q_k{}^{j-1},q_\ell^{j-1})$ in A_{ins} such that $q_k \in \mathit{reach}[j-1]$, then $c[i,j,q_\ell] = c[i,j-1,q_k]+1$, as this arc has a weight of one (one insertion). Therefore we have $c[i,j,q_\ell] = \min\{c[i,j-1,q_k]+1 \mid (q_k^{j-1},q_\ell^{j-1}) \in A_{\text{ins}} \wedge q_k \in \mathit{reach}[j-1]\}$ (lines 12 and 13). The condition $q_k \in \mathit{reach}[j-1]$ is crucial similarly to case 1.
4. There is always a deletion arc (q_ℓ^{j-1},q_ℓ^j) in A_{del} for *every* $q_\ell \in Q$, so $c[i,j,q_\ell] = c[i-1,j,q_\ell]+1$, as such an arc has a weight of one (one deletion). Therefore we have $c[i,j,q_\ell] = \min\{c[i-1,j,q_\ell]+1 \mid q_\ell \in \mathit{reach}[j]\}$ (lines 14 and 15). The condition $q_\ell \in \mathit{reach}[j]$ is crucial so that $c[i-1,j,q_\ell]$ and $c[i,j,q_\ell]$ are well-defined.

Hence $c[i,j,q_\ell]$ with $q_\ell \in \mathit{reach}[j]$ is the minimum among the four cases, which is the same as computed by the method. By complete induction, we finish the proof. □

Lemma 4 *The value of u computed by Algorithm 1 is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_n\rangle$ under the current domains and the n-letter regular language accepted by M.*

Proof: From Lemma 3, we know that $c[n,n,q_\ell]$ is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_n\rangle$ and any word of n symbols accepted by M from q_0 passing n transitions to q_ℓ. So if $q_\ell \in \mathit{reach}[n] \cap F$, then the sequence of n values labelled on the n transitions of such a path is a word of length n accepted by M. Hence u is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_n\rangle$ and the n-letter regular language accepted by M (line 16). □

Lemma 5 *Algorithm 1 computes all bindings that have minimum-weight-related paths of weight u.*

Proof: Consider the following four cases, where u is the minimum edit distance between any possible assignment of $\langle x_1,\ldots,x_n\rangle$ under the current domains and the n-letter regular language accepted by M:

1. Assume a path of weight u passes a copy arc (line 31), and the arc is labelled with t. The path is a minimum-weight path related with the binding $x_i := t$, hence the value t is added to $s[i]$ (line 32).
2. Assume a path of weight u passes a substitution arc (line 34). We know there is an assignment a, namely $X := \langle V_1,\ldots,V_n\rangle$, such that the edit distance between a and the word represented by the path is u. For any value $t \in \mathrm{dom}(x_i)$, we can get an assignment a', namely $\langle V_1,\ldots,V_{i-1},t,V_{i+1},\ldots,V_n\rangle$, and the edit distance between a' and the word represented by the path is also u (as the arc denotes a substitution operation on V_i: replacing V_i by t cannot increase the edit distance, and the edit distance cannot be larger than u; as u is the minimum weight, the edit distance cannot be less than u also). Hence for any value $t \in \mathrm{dom}(x_i)$, the path is a minimum-weight path related with $x_i := t$, and $s[i]$ is assigned $\mathrm{dom}(x_i)$ (any value in $\mathrm{dom}(x_i)$ is domain consistent) (line 35).
3. Assume a path of weight u passes an insertion arc (line 37). As an insertion arc is not an edit operation related with the binding to x_i, no value is added to $s[i]$ (no related path is found; line 38).
4. Assume a path of weight u passes a deletion arc (line 40). Similarly to case 2, for any value $t \in \mathrm{dom}(x_i)$, the path is a minimum-weight path related with $x_i := t$, and $s[i]$ is assigned $\mathrm{dom}(x_i)$ (any value in $\mathrm{dom}(x_i)$ is domain consistent) (line 41).

Hence Algorithm 1 correctly computes all bindings that have minimum weight related paths of weight u. □

Theorem 1 *Algorithm 1 computes the cost measure and achieves domain consistency on X and bounds consistency on z for the* SoftRegular(X,M,z) *constraint.*

Proof: The result follows from Lemmas 1, 2, 4, and 5. □

5.3 Complexity of the Propagator

To establish the time complexity of Algorithm 1, note that the vector *reach* only needs to be computed once, by exploring at most n times all transitions of M (as shown in Algorithm 2) in $O(n\cdot|\delta|)$ time, where $|\delta|$ denotes the number of transitions of M. The initialisation of the matrix c takes $O(n+n\cdot|Q|) = O(n\cdot|Q|)$ time. For any $1 \le i,j \le n$, the set of all elements $c[i,j,q_\ell]$ with $q_\ell \in reach[j]$ can be computed (upon using distributive laws) by exploring *once* each arc in $A_{\mathrm{copy}} \cup A_{\mathrm{sub}} \cup A_{\mathrm{ins}} \cup A_{\mathrm{del}}$ (which can have $O(|\delta|+|Q|)$ arcs: A_{copy} has $O(|\delta|)$ arcs, A_{sub} has $O(|\delta|)$ arcs, A_{ins} has $O(|\delta|)$ arcs, and A_{del} has $\Theta(|Q|)$ arcs), in $O(|\delta|+|Q|)$ time total. Hence computing the matrix c takes $O(n^2\cdot(|\delta|+|Q|))$ time in total, and the same holds for tracing all minimum-weight paths backwards. Computing u takes $O(|Q|)$ time by querying

$O(|Q|)$ elements of c. As $|\delta| = |Q| \cdot |\Sigma| \geq |Q|$, the overall complexity of the algorithm is $O(n \cdot |\delta| + n \cdot |Q| + n^2 \cdot (|\delta| + |Q|) + |Q|) = O(n^2 \cdot |\delta|)$ time, which is $\frac{n^2}{n+|Q|}$ times more expensive than the propagator of (van Hoeve et al. 2004, 2006), which however may guide the search to incorrect (non-optimal) solutions with an unsuitable flow network.

In (van Hoeve et al. 2006) (on page 369), it is assumed that $|Q| \leq n$. However in the worst case, we have $|Q| = |\Sigma|^{n+1}$, as M is at most a complete tree of depth n, where each state has $|\Sigma|$ transitions.

Considering space complexity, Algorithm 1 takes $O(n \cdot |Q| + n \cdot (|Q| + |\delta|)) = O(n \cdot |\delta|)$ space to store the nodes and arcs in the digraph G (as there are $O(n \cdot |Q|)$ nodes and $O(n \cdot (|Q| + |\delta|))$ arcs in G, and $|\delta| = |Q| \cdot |\Sigma| \geq |Q|$); it takes $O(n^2 \cdot |Q|)$ space for the matrices c and r, and $O(n \cdot |Q|)$ space for the vector *reach*; in addition, it takes $O(n \cdot |\Sigma|)$ space for the vector s. Hence Algorithm 1 takes $O(n \cdot |\delta| + n^2 \cdot |Q| + n \cdot |Q| + n \cdot |\Sigma|) = O(n \cdot |\delta| + n^2 \cdot |Q|)$ space in total, as $|\delta| = |Q| \cdot |\Sigma| \geq |\Sigma|$.

5.4 Revisiting the Example of Section 4

Given the same over-constrained problem as in Section 4, a CP solver using the propagator of Algorithm 1 will find the correct optimal solution to the problem. The propagator first computes the minimum edit distance u between all possible assignments (namely $X := \langle \text{e, v, e, v, d} \rangle$ and $X := \langle \text{e, v, e, d, d} \rangle$) and the 5-letter regular language (namely {ddddv, ddvdv, ddvev, dvddv, evddv}), which is 2 here. As $u > \min\ \text{dom}(z)$, we have that $\text{dom}(z)$ is updated to $\{2\}$. As $u = \max\ \text{dom}(z)$, the propagator traces all related minimum weight paths of weight u backwards, and finds two flows, namely

$$\text{S} \to \text{O}^0 \xrightarrow{\text{copy 'e'}} \text{E}^1 \xrightarrow{\text{copy 'v'}} \text{O}^2 \xrightarrow{\text{substitute 'd'}} \text{D}^3 \xrightarrow{\text{copy 'd'}} \text{D}^4 \xrightarrow{\text{substitute 'v'}} \text{O}^5 \to \text{T}$$

(assigns $\text{dom}(x_5) = \{\text{d}\}$ to $s[5]$, inserts $\{\text{d}\}$ into $s[4]$, assigns $\text{dom}(x_3) = \{\text{e}\}$ to $s[3]$, inserts $\{\text{v}\}$ into $s[2]$, and inserts $\{\text{e}\}$ into $s[1]$) and

$$\text{S} \to \text{O}^0 \xrightarrow{\text{copy 'e'}} \text{E}^1 \xrightarrow{\text{copy 'v'}} \text{O}^2 \xrightarrow{\text{delete 'e'}} \text{O}^3 \xrightarrow{\text{copy 'd'}} \text{D}^4 \xrightarrow{\text{copy 'd'}} \text{D}^5 \xrightarrow{\text{insertion 'v'}} \text{O}^5 \to \text{T}$$

(inserts $\{\text{d}\}$ into $s[5]$, inserts $\{\text{d}\}$ into $s[4]$, assigns $\text{dom}(x_3) = \{\text{e}\}$ to $s[3]$, inserts $\{\text{v}\}$ into $s[2]$, and inserts $\{\text{e}\}$ into $s[1]$). Hence $s[1] = \{\text{e}\}$, $s[2] = \{\text{v}\}$, $s[3] = \{\text{e}\}$, $s[4] = \{\text{d}\}$, and $s[5] = \{\text{d}\}$. As $v \notin s[4]$, the value v is removed from $\text{dom}(x_4)$. The domains of each decision variable and the cost variable z contain only one value, and the correct optimal solution (also the unique solution), namely $X := \langle \text{e, v, e, d, d} \rangle$ with $z = 2$, is found.

5.5 An Improved Propagator

Ukkonen (1985) observed that the dynamic programming algorithm of (Wagner and Fischer 1974), which computes the edit distance between two words, is often not efficient in practice, as it often evaluates unnecessary values of the matrix h. Revisit

the example in Figure 5, which gives the matrix h for computing the edit distance w between the words evddv and evedd of length $n = 5$. Each cell on the diagonal k_i (or f_i) has a value larger than or equal to i, as it is computed by operating at least i insertions (or deletions). If a cell on the diagonal k_i (or f_i) is a support to obtain the edit distance w, i.e., if there exists a path from this cell to the cell in the lower-right corner to compute the edit distance w, then this path passes at least another i deletions (or insertions), and hence $w = h[n,n] \geq 2 \cdot i$. Therefore only the sequence Δ_w of diagonals (namely $\Delta_w = \left[k_{\lfloor \frac{w}{2} \rfloor}, \ldots, k_1, 0, f_1 \ldots, f_{\lfloor \frac{w}{2} \rfloor}\right]$, see Figure 7) is necessary for computing the edit distance w. Recall that each cell $h[i, j]$ (with $i, j \geq 1$) is computed from three adjacent cells (namely $h[i-1, j-1]$, $h[i, j-1]$, and $h[i-1, j]$). If $h[i, j]$ is on the first or last diagonal of Δ_w, then $h[i, j-1]$ or $h[i-1, j]$ may be an unnecessary cell (we call a cell outside Δ_w an unnecessary cell), and its value is not computed. Whenever an unnecessary cell is queried, we just assume the value of this cell to be ∞ (as this cell is not a support to obtain the edit distance w). For example, in Figure 7, $h[2,1] = \min\{h[1,0]+1, h[2,0]+1, h[1,1]+1\} = \min\{1+1, \infty+1, 0+1\} = 1$, as $h[2,0]$ is an unnecessary cell.

Based on the observation above, an improved dynamic programming algorithm taking $O(w \cdot n)$ time is introduced in (Ukkonen 1985). Initially, the improved algorithm assumes the edit distance is $w' = 1$ and computes the edit distance using the sequence $\Delta_{w'}$ of diagonals (namely $\Delta_{w'} = \left[k_{\lfloor \frac{w'}{2} \rfloor}, \ldots, k_1, 0, f_1, \ldots, f_{\lfloor \frac{w'}{2} \rfloor}\right]$, which is $[0]$ here). If the computed edit distance w is larger than w', then the algorithm doubles the value of w' and recomputes w with the enlarged sequence $\Delta_{w'}$ of diagonals. This process repeats until w is not larger than w'. The improved algorithm runs for $\lceil \log_2 w \rceil + 1$ iterations and computes $1 + 2^{i-1}$ (or 1) diagonals in iteration i with $i > 1$ (or $i = 1$), where each diagonal has at most n cells. Hence the overall time complexity of the improved algorithm is

$$\left(1 + \sum_{i=2}^{\lceil \log_2 w \rceil + 1} (1 + 2^{i-1})\right) \cdot n = (2 \cdot w + \lceil \log_2 w \rceil - 1) \cdot n = O(w \cdot n)$$

and the worst-case time complexity is $O(n^2)$, since $w \leq n$.

For example, the improved dynamic programming algorithm computes the edit distance w between the words evedd and evddv as follows. Initially it assumes $w' = 1$ and the matrix h is computed as in Figure 6. As $w = h[5,5] = 2 > 1 = w'$, it doubles w'. Now we have $w' = 2$ and the matrix h is computed as in Figure 7. As $w = h[5,5] = 2 \leq w'$, the algorithm terminates and returns the edit distance $w = 2$ between the words evedd and evddv after computing $6 + (5 + 6 + 5) = 22$ cells of the matrix h (instead of computing all 36 cells of h as in Figure 5 when using the algorithm of Wagner and Fischer 1974).

Similarly to the algorithm of (Ukkonen 1985), the propagator of Algorithm 3 computes the cost measure and achieves domain consistency on X (and bounds consistency on z) for the SoftRegular(X, M, z) constraint in $O(\min(u, \max\ \mathrm{dom}(z)) \cdot n \cdot |\delta|)$ time with $O(n \cdot |\delta| + n^2 \cdot |Q|)$ space, where u is the minimum edit distance between any possible assignment of $\langle x_1, \ldots, x_n \rangle$ under the current domains and the n-letter regular language accepted by M. Although the space complexity of Algorithm 3

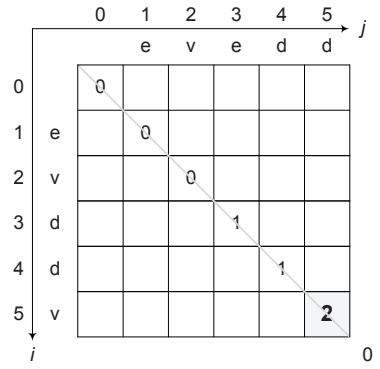

Figure 6 The matrix h for computing the edit distance $w = 2$ between the words evddv and evedd with $i = 1$ and $w' = 1$

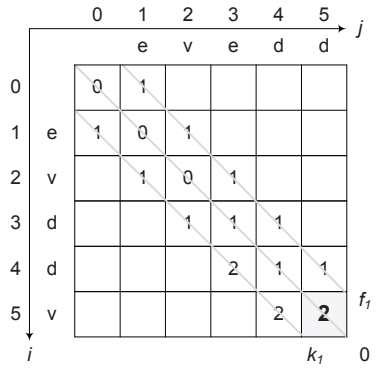

Figure 7 The matrix h for computing the edit distance $w = 2$ between the words evddv and evedd with $i = 2$ and $w' = 2$

Algorithm 3 An improved propagator computing the cost measure u for the SOFTREGULAR(X,M,z) constraint, with $|X| = n$ decision variables, a DFA $M = \langle Q,\Sigma,\delta,q_0,F\rangle$, and the cost variable z

1: lines 1 to 2 of Algorithm 1
2: **function** *propagator_improved*(SOFTREGULAR$(X,M,z),G$)
3: lines 4 to 5 of Algorithm 1
4: **int** u
5: **int** $u' \leftarrow 1$
6: **repeat**
7: **for all** $i \leftarrow 1$ **to** n **do**
8: **for all** $j \leftarrow \max(1, i - \frac{u'}{2})$ **to** $\min(n, i + \frac{u'}{2})$ **do**
9: lines 8 to 15 of Algorithm 1
10: $u \leftarrow \min\{c[n,n,q_\ell] \mid q_\ell \in F \cap reach[n]\}$
11: $u' \leftarrow 2 \cdot u'$
12: **until** $(u \leq \frac{u'}{2})$ **or** $(u > \max$ dom(z))
13: lines 16 to 27 of Algorithm 1
14: **for all** $i \leftarrow n$ **to** 1 **do**
15: **for all** $j \leftarrow \min(n, i + \frac{u'}{2})$ **to** $\max(1, i - \frac{u'}{2})$ **do**
16: lines 30 to 41 of Algorithm 1
17: lines 42 to 44 of Algorithm 1

is the same as the one of Algorithm 1, the asymptotic time complexity of Algorithm 3 is always at least as good as the one of Algorithm 1 (as $u \leq n$), but significantly better when u is small. Indeed, our experimental results in Section 7 show that Algorithm 3 works much better than Algorithm 1 in practice.

6 Other Propagators

Given a SOFTREGULAR(X,M,z) constraint with $|X| = n$ decision variables and the DFA $M = \langle Q,\Sigma,\delta,q_0,F\rangle$, we introduce and analyse two other correct propagators to compute the cost measure and achieve domain consistency on X (and bounds consistency on z) for the SOFTREGULAR(X,M,z) constraint.

We can also use the propagator for the MULTICOSTREGULAR constraint (Menana and Demassey 2009) with two cost variables z and z_1, where z_1 with $\mathrm{dom}(z_1) = \{0\}$ denotes the number of insertion arcs minus the number of deletion arcs in each flow of the flow network constructed from M with the revisions we indicated in Section 3, to compute the cost measure and achieve domain consistency on X (and bounds consistency on z). However there is no published time and space complexity of the propagator for the MULTICOSTREGULAR constraint, hence we cannot compare asymptotically.

6.1 Making Every Flow Represent a Word of Length n

Given a flow network with every flow *necessarily* passing the same number of deletion and insertion arcs, the propagator of (van Hoeve et al. 2004, 2006) for the SOFTREGULAR(X, M, z) constraint correctly computes the edit-distance based cost measure, as the precondition of the generic propagator of (van Hoeve et al. 2004, 2006) is now satisfied. We can construct such a flow network as follows: first compute the minimised intersection of M with the DFA accepting Σ^n, which will give a DFA M' that only accepts words of length n accepted by M; and then construct a flow network from M' according to Section 3 (with or without the revisions we indicated).

The time complexity of the propagator is established in two parts. The first part of the propagator computes the smallest distance from q'_k to q'_ℓ, for every pair of states q'_k and q'_ℓ of M'. This can be done in $\Theta(|Q'| \cdot |\delta'|)$ time through breadth-first search from every state of M', where $|Q'| = O(n \cdot |Q|)$ is the number of states in M', and $|\delta'| = O(n \cdot |\delta|)$ is the number of transitions in M'.

The second part of the propagator computes shortest paths from the source S to the sink T of the flow network through topological sort with table lookups, taking $O(n \cdot |\delta'|)$ time. However we cannot match this $O(n \cdot |\delta'|)$ time complexity for our implementation, which takes $O(|Q'|^2)$ time to deal with the insertion arcs on one layer by querying the smallest distance from q'_k to q'_ℓ, for all pairs of states q'_k and q'_ℓ of M'. Hence our implementation of the propagator takes $O(n \cdot (|Q'|^2 + |\delta'|)) = O(n^3 \cdot |Q|^2 + n^2 \cdot |\delta|)$ time to compute shortest paths from the source S to the sink T, with $O(n \cdot |Q'|^2)$ time for insertion arcs and $O(n \cdot |\delta'|)$ time for the other arcs.

Therefore, the overall time complexity is $O(|Q'| \cdot |\delta'| + n \cdot (|Q'|^2 + |\delta'|)) = O(n^2 \cdot |Q| \cdot |\delta| + n^3 \cdot |Q|^2)$, which is more expensive than the worst case $O(n^2 \cdot |\delta|)$ of Algorithm 3.

Considering that the DFA M' is necessarily acyclic, we can improve our implementation of the propagator by skipping the first part and changing the second part to use topological sort *without* table lookups. Hence our improvement of this propagator takes $O(n \cdot |\delta'|) = O(n^2 \cdot |\delta|)$ time, which is the same as the worst case of Algorithm 3. However our experimental results in Section 7 show that Algorithm 3 works better in practice.

The propagator takes $O(n \cdot (|\delta'| + |Q'|))$ space to store the flow network constructed from M', where $|Q'| = O(n \cdot |Q|)$ is the number of states in M', with $O(n \cdot |\delta'|)$ space for arcs and $O(n \cdot |Q'|)$ for nodes. Hence the overall space complexity is

$O(n^2 \cdot (|\delta| + |Q|))$, which is more expensive than the $O(n \cdot |\delta| + n^2 \cdot |Q|)$ space of Algorithm 3.

6.2 Using the WEIGHTEDGRAMMAR Constraint

Katsirelos et al. (2008, 2011) present a method of encoding the edit-distance based SOFTGRAMMAR constraint into the WEIGHTEDGRAMMAR constraint, and give a propagator for the WEIGHTEDGRAMMAR constraint based on the Cocke-Younger-Kasami (CYK) parser. Given an edit-distance based SOFTREGULAR(X, M, z) constraint (with $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ and $|X| = n$), we can use the propagator for the WEIGHTEDGRAMMAR constraint to compute the edit-distance based cost measure and achieve domain consistency on X (and bounds consistency on z) for the SOFTREGULAR constraint as follows: first, we construct a DFA M' (with $M' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$) that only accepts words of length n accepted by M in the same way as in Section 6.1; second, we encode every transition of δ' into a zero-weight production of a grammar; third, we add unit-weight productions into the grammar to simulate substitution, insertion, and deletion operations (as in Katsirelos et al. 2008, 2011), and the resulting weighted grammar has a size of $O(|\delta'|)$; finally, we use the CYK-based propagator on the obtained WEIGHTEDGRAMMAR constraint. As the obtained weighted grammar is necessarily linear, the CYK-based propagator uses $O(n^2 \cdot |\delta'|) = O(n^3 \cdot |\delta|)$ time and space, as $|\delta'| = O(n \cdot |\delta|)$ (Katsirelos et al. 2009), which is n times more expensive (in both time and space) than the worst case $O(n^2 \cdot |\delta|)$ of Algorithm 3. Our experimental results in Section 7 confirm that Algorithm 3 works much better in practice.

Note that it is *necessary* to use the DFA M' (and not M) to generate the input grammar. If we use the DFA M, then the obtained grammar accepts words of the *whole* regular language instead of the n-letter regular language of M, and the CYK-based propagator with such an unsuitable grammar may thus underestimate the cost measure. A counterexample is given in Appendix A.

7 Experimental Evaluation

We now investigate experimentally the efficiency of Algorithm 3 by comparing it to Algorithm 1 and the two propagators in Sections 6.1 and 6.2. We implemented all these propagators for the CP back-end of COMET (Van Hentenryck and Michel 2005). We did two experiments, where each model contains two SOFTREGULAR constraints sharing the variables X but each constraint has its own cost variable, and the objective is to find a solution that minimises the sum of the two cost variables. All experiments use the same search heuristic, which uses the first-fail principle first on the two cost variables and then on the decision variables X. We need not try other branching heuristics, as that is *orthogonal* to our purpose of giving a fair comparison of the four propagators. All experiments were run under COMET (version 2.1.1) and Suse Linux 11.3 on a 3.07 GHz Intel Core i7 with a 3GB RAM.

The first experiment uses the two small DFAs in Figure 8: one with 3 states and 4 transitions, and the other with 6 states and 7 transitions. The initial domains for

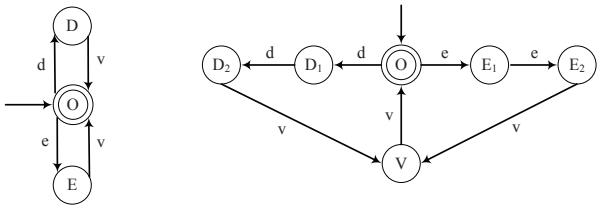

Figure 8 Two DFAs used in the first experiment

n	obj	Algorithm 3	Algorithm 1	Section 6.1	Section 6.2	#branch	#propag	#fail
12	6	0.6	1.7	2.2	16.5	538	3,489	1,522
16	8	31.0	90.4	115.9	1,190.6	16,289	105,097	46,462
20	10	1,401.6	4,390.5	5,416.3	74,644.0	517,364	3,342,003	1,477,278

Table 1 Results for the experiment with two small DFAs, where each row indicates $|X| = n$, the computed objective value, the runtime (in seconds) of the four propagators, the number of branchings, the number of propagations, and the number of failures

all variables of X are the same, namely $\{\mathrm{d}, \mathrm{e}, \mathrm{v}\}$, and the initial domains for the two cost variables are the same, namely $\{0, \ldots, |X|\}$. In Table 1, each row indicates $|X| = n$, the computed objective value (the minimum sum of the two cost variables), the runtime (in seconds) of the four propagators, the number of branchings, the number of propagations, and the number of failures. Note that for each of the four columns obj, #branch, #propag, and #fail, all four propagators necessarily have the same values. From Table 1, we observe that: Algorithm 3 has the best runtime among the four propagators, and is even about 3 times faster than the second-best propagator (Algorithm 1); the propagator in Section 6.2 has the worst runtime, as it is the only one with an $O(n^3)$ time complexity, while the other three have an $O(n^2)$ time complexity; Algorithm 1 and the propagator in Section 6.1 have close runtimes, but Algorithm 1 is about 1.5 times faster. In this experiment, the computed objective value increases linearly with n, but the runtime of all propagators increases super-linearly with n.

The second experiment uses the two large DFAs from case 15 and case 16 of (Beldiceanu et al. 2013), which are used to model a nurse scheduling problem. In case 15, the DFA has 1,115 states and 2,272 transitions; in case 16, the DFA has 1,309 states and 3,698 transitions. The initial domains for all variables of X are the same, namely $\{1,2,3,4\}$, and the initial domains for the two cost variables are the same, namely $\{0, \ldots, |X|\}$. Table 2 gives the results for the second experiment. We observe that: the minimum sum of the two cost variables is always 0 in this experiment, hence it is a test where the computed minimum edit distance u during each propagation takes small values; Algorithm 3 is the best among the four propagators, and is already 13 times faster than the second-best propagator (Algorithm 1) when $n = 28$; the runtime of Algorithm 3 only increases a little when n increases, as its complexity is bounded by an expression on u; and the comparisons of the other three propagators are the same as for Table 1.

n	obj	Algorithm 3	Algorithm 1	Section 6.1	Section 6.2	#branch	#propag	#fail
12	0	1.7	1.8	2.4	7.8	14	13	0
16	0	2.5	4.2	5.1	91.9	18	33	2
20	0	2.0	7.6	10.9	406.9	22	41	2
24	0	2.5	19.2	26.3	1,334.3	26	61	4
28	0	2.9	39.4	54.2	3,180.3	30	81	6

Table 2 Results for the experiment with two large DFAs, where each row indicates $|X| = n$, the computed objective value, the runtime (in seconds) of the four propagators, the number of branchings, the number of propagations, and the number of failures

8 Constraint-Based Local Search

Constraint-based local search (CBLS, e.g., Van Hentenryck and Michel 2005) is the local search approach to CP. In CBLS, constraints are used to describe and control local search. Given an initial assignment of values to all the variables, CBLS tries to find a better assignment that decreases the amount of constraint violation, by exploring a neighbourhood of the current assignment, that is a set of assignments that do not differ much from the current one. An assignment with zero (or minimum) violation is to be found. Meta-heuristics are used to escape local minima.

In (Pralong 2007; He et al. 2011), two Hamming-distance based violation measures of the REGULAR constraint for CBLS have been introduced. However there is no work on an edit-distance based REGULAR constraint for CBLS, as far as we know. Interestingly, our method can also be used for the edit-distance based REGULAR(X,M) constraint for CBLS. Given the current assignment a to the n decision variables $X = \langle x_1,\ldots,x_n\rangle$, the flow network G can be constructed by setting the domain of each decision variable x_i to a singleton, namely $\mathrm{dom}(x_i) = \{a(x_i)\}$, where $a(x_i)$ is the value assigned to x_i under a. Given a SOFTREGULAR(X,M,z) constraint, in Algorithm 1 the variable u is computed as the minimum edit distance between the current domains of X and the n-letter regular language accepted by the DFA M; thus u can be taken as the edit-distance based constraint violation measure of a REGULAR(X,M) constraint in CBLS.

9 Conclusion

We have used the edit-distance based SOFTREGULAR constraint as an example to show that a propagator that sometimes underestimates the cost measure for a soft constraint may guide the search to incorrect (non-optimal) solutions to an over-constrained problem. We have presented a propagator and an improved version that correctly compute the cost measure for the SOFTREGULAR constraint, and favourably compared theoretically and experimentally our propagators with two other propagators. We have shown that our method can also be adapted for the violation measure of the edit-distance based REGULAR constraint for constraint-based local search.

Future work includes using the idea of (Ukkonen 1985) to improve propagators that are based on dynamic programming for other constraints.

Acknowledgements The authors are supported by grants 2007-6445 and 2011-6133 of the Swedish Research Council (VR), and Jun He is also supported by grant 2008-611010 of China Scholarship Council and the National University of Defence Technology of China. Many thanks to George Katsirelos for some useful discussions on Section 6.2, to Louis-Martin Rousseau for some useful discussions on Section 4, and to the anonymous referees of this paper for their helpful comments.

Appendix A: An Example of Encoding the SOFTREGULAR Constraint Incorrectly

Given an edit-distance based SOFTREGULAR(X,M,z) constraint with $M = \langle Q,\Sigma,\delta,q_0,F\rangle$ and $|X| = n$, we give an example to show that the CYK-based propagator for the WEIGHTEDGRAMMAR constraint of (Katsirelos et al. 2008, 2011) with a weighted grammar obtained from M may underestimate the edit-distance based cost measure, as the grammar accepts words of the *whole* regular language instead of the n-letter regular language of M. Hence we cannot use the weighted grammar obtained from M when encoding the SOFTREGULAR(X,M,z) constraint with the WEIGHTEDGRAMMAR constraint.

The DFA M in Figure 1 can be converted into the following context-free grammar (CFG) G_1 by encoding every transition of M into a linear production, where O is the start symbol:

$$\begin{aligned} G_1: \quad O &\to dD \mid eE \mid \varepsilon \\ D &\to dD \mid vO \\ E &\to vO \end{aligned}$$

The CFG G_1 can be converted into the following Chomsky normal form (CNF) G_2:

$$\begin{aligned} G_2: \quad O &\to Y_dD \mid Y_eE \\ D &\to Y_dD \mid Y_vO \mid v \\ E &\to Y_vO \mid v \\ Y_d &\to d \\ Y_e &\to e \\ Y_v &\to v \end{aligned}$$

The WEIGHTEDGRAMMAR constraint can be used to encode the edit-distance based SOFTGRAMMAR constraint (Katsirelos et al. 2008, 2011). Given the CNF G_2, the following weighted productions will be added to simulate substitution, insertion, and deletion operations:

$$\begin{aligned} \text{substitution productions}: \quad Y_d &\to e \mid v, \text{ with weight 1} \\ Y_e &\to d \mid v, \text{ with weight 1} \\ Y_v &\to d \mid e, \text{ with weight 1} \\ \text{insertion productions}: \quad Y_d &\to \varepsilon, \text{ with weight 1} \\ Y_e &\to \varepsilon, \text{ with weight 1} \\ Y_v &\to \varepsilon, \text{ with weight 1} \\ D &\to \varepsilon, \text{ with weight 1} \\ E &\to \varepsilon, \text{ with weight 1} \\ \text{deletion productions}: \quad O &\to HO \mid OH, \text{ with weight 0} \\ D &\to HD \mid DH, \text{ with weight 0} \\ E &\to HE \mid EH, \text{ with weight 0} \\ Y_d &\to HY_d \mid Y_dH, \text{ with weight 0} \\ Y_e &\to HY_e \mid Y_eH, \text{ with weight 0} \\ Y_v &\to HY_v \mid Y_vH, \text{ with weight 0} \\ H &\to d \mid e \mid v, \text{ with weight 1} \end{aligned}$$

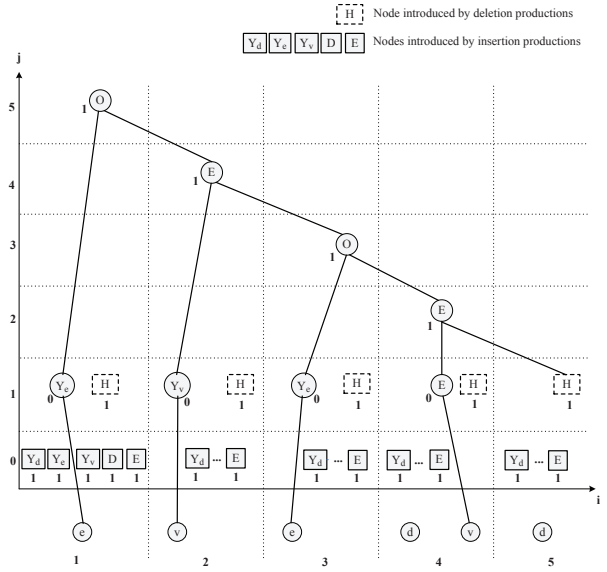

Figure 9 A minimum-weight parse tree computed by the CYK-based propagator of (Katsirelos et al. 2011)

Consider the SOFTREGULAR(X,M,z) constraint, where $X = \langle x_1,\ldots,x_5\rangle$ is a sequence of $|X| = 5$ decision variables, with current domains $\mathrm{dom}(x_1) = \mathrm{dom}(x_3) = \{\mathrm{e}\}$, $\mathrm{dom}(x_2) = \{\mathrm{v}\}$, $\mathrm{dom}(x_4) = \{\mathrm{d}, \mathrm{v}\}$, and $\mathrm{dom}(x_5) = \{\mathrm{d}\}$, and M is the DFA of Figure 1. The minimum edit distance between all feasible assignments (namely {evedd, evevd}) and the 5-letter regular language accepted by M (namely {ddddv, ddvdv, ddvev, dvddv, evddv}) is 2. However, as shown in Figure 9, the minimum weight computed by the CYK-based propagator of (Katsirelos et al. 2008, 2011) with the obtained weighted grammar is 1 (instead of 2), which is the same as the one computed in (van Hoeve et al. 2004, 2006) measuring the edit distance from word evevd to evev (in the 4-letter regular language accepted by M) through one deletion operation, hence the CYK-based propagator with the unsuitable weighted grammar underestimates the cost measure in this case.

Actually, in order to make the CYK-based propagator for the WEIGHTEDGRAMMAR constraint of (Katsirelos et al. 2008, 2011) work properly for the edit-distance based SOFTGRAMMAR constraint, we claim that two more changes are needed in addition to the one mentioned for this purpose on page 200 of (Katsirelos et al. 2011), which changes a loop control variable in order to handle ε productions.

1. Unit-weight ε productions are introduced to simulate insertion operations. Here, ε production means a production that generates ε. In order to handle these ε productions, the CYK-based propagator allows a symbol generated from another symbol in the same cell. For example, in Figure 10, there are 2 symbols, C and E, in the cell $(i = 1, j = 0)$ generated from the two insertion productions $\mathrm{C} \to \varepsilon$ and $\mathrm{E} \to \varepsilon$. Note that the example of Figure 10 has no relation to our running example of Section 4. In the cell $(i = 1, j = 2)$, there are three symbols O, D, and A generated from the three productions $\mathrm{O} \to \mathrm{CD}$, $\mathrm{D} \to \mathrm{EA}$, and $\mathrm{A} \to \mathrm{BC}$ respectively. When the CYK-based propagator computes the lower (or upper) bounds, it is crucial that these three productions are explored in a correct order: first $\mathrm{A} \to \mathrm{BC}$, then $\mathrm{D} \to \mathrm{EA}$, and finally $\mathrm{O} \to \mathrm{CD}$ (or in the opposite order), so that the lower (or upper) bounds of O, D, and A in cell $(i = 1, j = 2)$ are computed correctly. Hence all symbols in each cell must be sorted before computing the bounds.
2. Line 73 of the CYK-based propagator on page 191 of (Katsirelos et al. 2011), where the domains of the decision variables are pruned, also needs to be modified to suit the case of substitution and deletion productions, so that the domain of the decision variable x_i should not be pruned if there exists a symbol with an upper bound of 1 in cell $(i,1)$ denoting a substitution or deletion production.

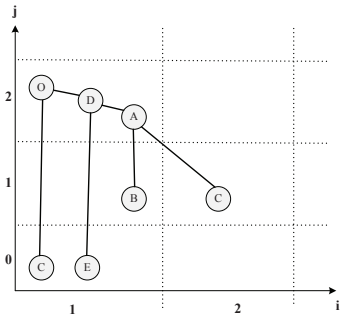

Figure 10 An example for the CYK-based propagator of (Katsirelos et al. 2011) with insertion productions that generate ε

References

Apt, K.: *Principles of Constraint Programming*. Cambridge University Press, Cambridge (2003)

Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On matrices, automata, and double counting in constraint programming. *Constraints* 18(1), 108–140 (2013).

Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) *Proceedings of CP'04*, volume 3258 of LNCS, pp. 107–122. Springer, Berlin (2004)

He, J., Flener, P., Pearson, J.: An *automaton* constraint for local search. *Fundamenta Informaticae* 107(2–3), 223–248 (2011)

Kadioğlu, S., Sellmann, M.: Grammar constraints. *Constraints* 15(1), 117–144 (2010)

Katsirelos, G., Narodytska, N., Walsh, T.: The weighted CFG constraint. In: Perron, L., Trick, M. (ed.) *Proceedings of CP-AI-OR'08*, volume 5015 of LNCS, pp. 323–327. Springer, Berlin (2008)

Katsirelos, G., Maneth, S., Narodytska, N., Walsh, T.: Restricted global grammar constraints. In: Gent, I.P. (ed.) *Proceedings of CP'09*, volume 5732 of LNCS, pp. 501–508. Springer, Berlin (2009)

Katsirelos, G., Narodytska, N., Walsh, T.: The weighted GRAMMAR constraint. *Annals of Operations Research* 184, 179–207 (2011)

Menana, J., Demassey, S.: Sequencing and counting with the multicost-regular constraint. In: van Hoeve, W.-J., Hooker, J.N. (ed.) *Proceedings of CP-AI-OR'09*, volume 5547 of LNCS, pp. 178–192. Springer, Berlin (2009)

Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) *Proceedings of CP'04*, volume 3258 of LNCS, pp. 482–495. Springer, Berlin (2004)

Pralong, B.: Implémentation de la contrainte *Regular* en *Comet*. Master's Thesis, École Polytechnique de Montréal, Canada (2007)

Quimper, C.-G., Walsh, T.: Global grammar constraints. In: Benhamou, F. (ed.) *Proceedings of CP'06*, volume 4204 of LNCS, pp. 751–755. Springer, Berlin (2006)

Ukkonen, E.: Algorithms for approximate string matching. *Information and Control* 64(1–3), 100–118 (1985)

Van Hentenryck, P., Michel, L.: *Constraint-Based Local Search*. MIT Press, Cambridge (2005)

van Hoeve, W.-J., Pesant, G., Rousseau, L.-M.: On global warming (Softening global constraints). In: *Proceedings of the 6th International Workshop on Preferences and Soft Constraints*, available at `http://www.andrew.cmu.edu/user/vanhoeve/papers/softglob.pdf` (2004)

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

Wagner, R.A., Fischer M.J.: The string-to-string correction problem. *Journal of the ACM* 21, 168–173 (1974)

Zanarini, A., Milano, M., Pesant, G.: Improved algorithm for the soft global cardinality constraint. In: Beck, J.C., Smith, B. (ed.) *Proceedings of CP-AI-OR'06*, Volume 3990 of LNCS, pp. 288–299. Springer, Berlin (2006)