Maintaining Consistency of Dynamic Cardinality Constraints with Costs

Waldemar Kocjan[1], Per Kreuger[2], Björn Lisper[1]

[1] Mälardalen University, Västerås, Sweden
{waldemar.kocjan,bjorn.lisper}@mdh.se
[2] Swedish Institute of Computer Science, Kista, Sweden
piak@sics.se

Abstract. This paper introduce a novel method for maintaining consistency of cardinality constraints in context of dynamic constraint satisfaction. The presented method adopt sensitivity analysis for feasible and minimum cost flows underlying all cardinality constraints. Moreover, we extend a dynamic all-pairs shortest path algorithm to support weight changes, insertion and deletion of multiple edges and show how this algorithm can be used to maintain consistency of values with the modified constraint.

1 Introduction

Within constraint programming we have seen the development a number of methods that adopt and extend assignment models known from optimization and operation research for constraint consistency and filtering. Global constraints which deal with the cardinality of the variables and values in a problem are typical examples of such methods.

To this class of constraints belongs, e.g., the constraints of difference [14] and the global cardinality constraint [15], as well as their weighted versions ([4, 9] resp. [16, 17]). These models have recently [12] been extended to set constraint satisfaction problems by the introduction of the *symmetric cardinality constraint* and its weighted version [10].

Like most of the methods used in constraint programming, those mentioned above deal mainly with problems which demand only a single solution, and they assume completely known and persistent problem parameters. Unfortunately, this is seldom the case in real life problems. Many assignments problems need to be re-computed due to changes to, e.g., the costs of assignments or in the bounds of individual variables, and even after adding and removing variables.

Constraint Programming problems in which parameters change over time are usually solved by recomputation from scratch ([5, 11]). Some other methods focus on modifying the current solution ([3, 19]). The only method which handles dynamic versions of global constraints is described in [2]. However, this method handles only monotonic changes, and relies on backtracking in the context of Constraint Logic Programming.

In this paper we introduce a method for maintaining consistency of cardinality constraints in the context of dynamic constraint satisfaction. The methods presented here are based on sensitivity analysis of the feasible and minimum cost flows underlying every cardinality constraint.

This paper is organized as follows. In Section 2 we describe some basic concepts in flow theory and constraint satisfaction. Section 3 gives a brief description the symmetric cardinality constraint with costs, and presents methods for checking the consistency of the constraint and for filtering the domains of the constrained variables. Section 4 introduces basic notions of sensitivity analysis, and the following sections (5 − 7 describe how sensitivity analysis can be used to maintain consistency of a cardinality constraint as well as for maintaining the optimality of an underlying minimum cost flow in the value network of a cardinality constraint. Section 10, finally, concludes the paper and discusses future work.

2 Preliminaries

2.1 Graphs

Definitions in this section follows the presentation in[1].

A *directed graph* $G = (X, E)$ consists of a set of nodes (vertices) X and arcs (edges) E, where every arc $(u, v) \in E$ is an ordered pair of distinct nodes.

An arc (u, v) connects node u with node v, i.e. in directed graph it is an arc oriented from node u to node v. A *path* in a graph G from v_1 to v_k is a sequence of nodes $[v_1, \ldots, v_k]$ such that each (v_i, v_{i+1}) is an arc for $i \in [1, \ldots, k-1]$. The path is *simple* if all its nodes are distinct.

2.2 Network Flows

Definition 1. *A* network $G = (X, E)$ *is a directed graph, in which each arc* $e \in E$ *is associated with two non-negative integers* l_e *and* u_e *representing the lower and upper bounds, respectively, on the flow on* e.

The upper bound on the flow on an arc is also referred to as the *capacity* of the arc.

A flow f_e on an arc e represents the amount of commodity that the arc accommodates. More formally:

Definition 2. *A* flow f *in a network* (X, E) *is a function that assigns to each arc* $(i, j) \in E$ *a value* f_{ij} *in such way that for each node* $p \in X$,

$$\sum_{(i,p)\in E} f_{ip} = \sum_{(p,j)\in E} f_{pj}$$

This property is known as a *conservation law* and states that for each node in G, the incoming amount of flow of some commodity equals the amount of that commodity leaving that node.

Definition 3. *A flow f in a network (X, E) is* feasible *if, for each $e \in E$, it holds that $l_e \leq f_e \leq u_e$.*

Associate furthermore with each arc $(i, j) \in E$ a *cost* c_{ij} that denotes the cost per unit of flow on the arc. For any flow f in G, we define

$$cost(f) = \sum_{(i,j) \in E} c_{ij} * f_{ij} \tag{1}$$

A *minimum cost flow problem* is the problem of finding a feasible flow in G with minimal cost.

In this paper we assume that all the parameters (c_{ij}, l_e and u_e) of a flow problem are integral, which guarantees the existence of an integral minimal solution to the problem. (For the integrality property of minimum cost flow problems see, e.g., [1, p. 318]).

The *residual graph* $R(f)$ is a graph representing the utilization and remaining capacity in the flow graph with respect to a flow f.

Definition 4. *Given a flow f from s to t in the network G, the* residual graph *for f, denoted $R(f)$, has the same set of nodes as G. The arc set of $R(f)$ is defined as follows. For all arcs (i, j) in G,*

– *if $f_{ij} < u_{ij}$ then (i, j) is an arc of $R(f)$ with residual capacity $r_{ij} = u_{ij} - f_{ij}$, and cost c_{ij}*
– *if $f_{ij} > l_{ij}$ then (j, i) is an arc of $R(f)$ with residual capacity $r_{ji} = f_{ij} - l_{ij}$ and cost $-c_{ij}$*

Moreover, the *potential function* and the *reduced costs* are defined as follows.

Definition 5. *A* potential function *is a function π which associates with each node $i \in G$ a number $\pi(i)$, which is referred to as a node potential. With respect to the node potentials, the* reduced cost c^{π}_{ij} *of an arc (i, j) in $R(f)$ is defined by $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$.*

For minimum cost flows node potentials are typically defined by $\pi(i) = -d$, where d is a shortest path distance from a given source to i.

2.3 Constraint Satisfaction

A *constraint satisfaction problem (CSP)* is a triple (X, D, C) where X is a finite set of variables $\{x_1, \ldots, x_n\}$, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains, where D_i is a set of values for the variable $x_i \in X$, and $C = \{C_1, \ldots, C_n\}$ is a set of constraints between variables [18].

The symmetric cardinality constraint used in this paper is defined in context of the *set constraint satisfaction problem (sCSP)*, which differs from CSP by the fact that each $x_i \in X$ is assigned a *subset* of elements in D_i [12].

Let $\mathcal{P}$ be a (set) constraint satisfaction problem. A *dynamic constraint satisfaction problem $DCSP = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$* [6] is a sequence of (s)CSPs such that any $\mathcal{P}_i$ differs from the consecutive $\mathcal{P}_{i+1}$ by a set of added and a set of deleted constraints.

Note, that a dynamic constraint satisfaction problem is a sequence of static problems. Any constraint can be added to or deleted from the problem after the solution to the previous problem was found.

3 The Symmetric Cardinality Constraint with Costs

In this section we will briefly describe the symmetric cardinality constraint with costs, which we will use later to illustrate the sensitivity analysis. A detailed description of this constraint can be found in [10].

Definition 6. *A symmetric cardinality constraint with cost is a constraint C over a set of variables $X(C)$ and a set of values obtained from the domains of these variables:*

$$D(X(C)) = \bigcup_{x \in X(C)} D_x$$

The constraint associates with each value $v_j \in D(X(C))$ two non-negative integers l_{v_j} and u_{v_j}, with each variable $x_i \in X(C)$ two other non-negative integers l_{x_i} and u_{x_i} and with the occurrence of a value $v_j \in D_{x_i}$ in a set assigned to a variable x_i a cost $cost(x_i, v_j)$. Furthermore it associates a fixed integer limit H on the total cost of the constraint as defined below.

1. $\forall i \, (l_{x_i} \leq \#(x_i, P) \leq u_{x_i})$
2. $\forall j \left(l_{v_j} \leq \#(v_j, C, P) \leq u_{v_j}\right)$.
3. $\sum_{i=1}^{|X(C)|} \sum_{v_j \in D_{x_i}} cost(x_i, v_j) \leq H$

where $\#(x_i, P)$ is the cardinality of the set assigned to x_i by P and $\#(v_j, C, P)$ the number of variables to which v_j is assigned by P.

The symmetric cardinality constraint with costs extends the symmetric cardinality constraint of [12] with a cost associated with each assignment. The constraint can also be seen as an extension of the global cardinality constraint with costs [17] into the set constraint satisfaction setting.

For each global cardinality constraint with costs, there is an equivalent symmetric cardinality constraint with costs where the cardinality of each constraint variable is restricted to be in the interval $[1, 1]$. Similarly, for a constraint of difference with costs there is a symmetric cardinality constraint with costs where the bounds of each variable are restricted to $[1, 1]$, the bounds of each value is restricted to $[0, 1]$, and $H = \infty$.

The first step of verifying the consistency of a symmetric cardinality constraint with costs C is to build its value network $N(C)$.

Definition 7. *A value network of a symmetric cardinality constraint is a network obtained by*

1. *Adding a node for each variable $x_i \in X(C)$, a node for each value in $v_j \in D(X(C))$, a source node s and a sink node t.*
2. *Adding an arc (s, v_j) from s to each node v_j and bounding the flow on it to be between l_{v_i} and u_{v_i} at zero cost $c_{sv_i} = 0$.*
3. *Adding an arc from each node $v_i \in D_{x_i}$ to x_i, and bounding the flow on each such arc to be between 0 and 1 at a cost equal to that of the occurrence of v_i in x_i.*
4. *Adding an arc (x_i, t) from each node x_i to t and bounding the flow on it to be between l_{x_i} and u_{x_i} at zero cost $c_{x_it} = 0$.*
5. *Adding an arc from t to s and bounding the flow on this arc to be between 0 and ∞ at a zero cost $c_{t,s} = 0$.*

The following theorem relates consistency of a symmetric cardinality constraint to a flow in its value network.

Theorem 1. *A symmetric cardinality constraint with costs C is consistent if and only if there exists a feasible flow f^C in the value network $N(C)$ such that $cost(f^C) \leq H$.*

Proof. See [10] for proof. □

In practice, computing consistency of a symmetric cardinality constraint with costs C is done by computing a minimum cost flow in $N(C)$. Several algorithms can be used. For the state of the art in computing minimum cost flow, see [1].

In this paper we assume that minimum cost flows are computed using the *successive shortest path algorithm*. Briefly, the successive shortest path algorithm searches for the shortest path from node j in an arc (i, j), whose lower bound on the flow is not satisfied, iteratively to every other node in the graph. After each iteration, the algorithm augments the flow on (i, j) along the shortest path from j to i and updates the residual graph relative to the newly computed flow. Moreover, the potential of each node k, initially set to 0, is updated by subtracting its distance from j. For a detailed description see [1], pp. 320-324.

The best known algorithm for computing shortest path is Dijkstra's algorithm with Fibonacci heaps implementation. Using this algorithm, the worst time complexity for verifying the consistency of C becomes equal to

$$O\left(\left(\sum_{i=1}^{|X|} l_{x_i} + \sum_{j=1}^{|D|} l_{v_j}\right) \times (m + n \ log \ n)\right)$$

where m is the number of edges and n the number of nodes in the value network. Dijkstra's algorithm requires that the arcs in the network have non-negative costs, which can be ensured by transforming original costs into reduced costs (see [10] for details).

A minimum cost flow implies that the reduced cost of any arc in the graph is non-negative ([1], p. 308-309, see also Theorem 3 in Section 4). Given a minimum cost flow f^C in the value network of $N(C)$, we can compute the consistency of each value v_j in x_i by computing the shortest path from x_i to v_j in the residual graph of f^C, and checking if there exists a feasible flow containing (v_j, x_i) which cost is lower than or equal to H.

4 Sensitivity Analysis

The purpose of sensitivity analysis is to determine changes in the solution resulting from the changes in the problem parameters. Here, we are interested in a sensitivity analysis of the *consistency* of a symmetric cardinality constraint with costs, as well as a sensitivity analysis of a *minimum cost flow* in the value network of the constraint. The results apply also to global cardinality constraints with costs and constraints of difference with costs, since these are instances of the symmetric cardinality constraint with costs.

Let $\mathcal{C} = \{C_1, \dots, C_n\}$ be a dynamic constraint satisfaction problem consisting of a sequence of symmetric cardinality constraint with costs. For each $C_i \in \mathcal{C}$ we will denote its value network by $N(C_i)$, the flow in is network by f^{C_i}, and the residual graph of its flow by $R(f^{C_i})$.

Recall from Section 3 that a symmetric cardinality constraint with costs C_i is consistent iff there exists a feasible flow f^{C_i} in $N(C_i)$ such that $cost(f^{C_i}) \leq H$, where H is a non-negative integer. Moreover, a flow is feasible if it satisfies the lower and upper bound constraints of each arc in the graph. Consequently, a sensitivity analysis for consistency of a symmetric cardinality constraint with costs is an analysis of how changes to the parameters of a problem influence the feasibility of the flow in the value network of the constraint.

Computing a feasible flow is performed by augmenting and reducing a flow on arcs in a graph. Given a feasible flow we can compute a new one using the following theorem, due to J.-L. Laurière [13] .

Theorem 2. *Let f be a feasible flow in a network N and $R(f)$ be a residual graph for f. If there is a simple path p from n to m in $R(f) - \{(n, m)\}$ then we can obtain a new feasible flow f' in N such that $f^{'}_{mn} > f_{mn}$. Such a path is referred to as an augmenting path.*

Moreover, if there is a simple path p from m to n in $R(f) - \{(m, n)\}$ then we can obtain a new feasible flow f' in N such that $f^{'}_{mn} < f_{mn}$. Such a path is referred to as a reducing path.

If no augmenting path for f_{mn} exists then f_{mn} is a *maximum flow* through (m, n). Similarly, if there is no reducing path for f_{mn} then f_{mn} is a *minimum cost flow* through (m, n).

Performing sensitivity analysis of consistency of a cardinality constraint requires information about the value network of the constraint and the feasible flow found during the computation. To make this information accessible we can store the residual graph resulting from computing a feasible flow in the value network of the constraint. Let n denote a number of nodes in the graph and m number of edges. Storing information about the residual graph requires at most $O(n + m)$ space, where $n = |X| + |D| + 2$ and m in the worst case is equal to $\sum_{i=1}^{|X|} |D_{x_i}| + 2|D| + 2|X| + 2$.

In many situations it is interesting to maintain the minimum cost flow in the value network of a cardinality constraint. This is the case when a cardinality constraint is a subject to an objective function which minimizes the cost of the occurrence of all values, or when we are interested in maintaining the consistency of each value with the constraint. In the second case, maintaining a minimum cost flow makes it possible to use more powerful algorithms for computing shortest paths in the graph.

There are several methods for verifying the optimality of a flow in a graph (see [1], p. 306–315). Here we will use the *reduced cost optimality condition*.

Theorem 3. ***(Reduced Cost Optimality Condition).*** *A feasible flow f is a minimum cost flow if and only if some set of node potentials π satisfy the following reduced cost optimality conditions:*

$$c^{\pi}_{ij} \geq 0 \textit{ for every arc } (i,j) \textit{ in } R(f) \tag{2}$$

The reduced cost c^{π}_{ij} of an arc (i,j) indicates that augmenting or reducing a flow through (i,j) with 1 unit of commodity will change the total cost of the flow in a graph with c^{π}_{ij} (see [1], p. 43–44).Consequently, if $c^{\pi}_{ij} < 0$ then augmenting a flow through (i,j) might reduce the total cost of the flow in the graph with c^{π}_{ij}. Thus, if there exists a path p from j to i in $R(f) - \{(j,i)\}$ then, by Theorem 2, we can obtain a new feasible flow f' by augmenting a flow along p. Moreover, if $d_{ji} < -c^{\pi}_{ij}$ then $cost(f') < cost(f)$. If p is a shortest path from j to i then $d_{ji} + c^{\pi}_{ij}$ is a greatest possible reduction of $cost(f)$.

If no reducing path p for (i,j) exists then, by Theorem 2, there is no feasible flow in the graph containing (i,j) thus (i,j) is not contained in a minimum cost flow. Similarly, if $d_{ji} \geq -c^{\pi}_{ij}$ then augmenting a flow through (i,j) does not reduce the total cost of f. In these cases the reduced costs optimality conditions are maintained by adjusting potential of i with c^{π}_{ij}. See [1] p. 339 for justification and pp. 320-324 on relation between node potentials and minimum cost flow.

To perform a sensitivity analysis based on Theorem 3 we need additional information about the previously computed node potentials for each node of the graph. Storing this information requires an additional $|X| + |D| + 2$ space.

In the following Sections 5 – 8 we show how the sensitivity analysis can be used to maintain the consistency of a cardinality constraint with costs and a minimum cost flow in the value network of the constraint.

5 Changes to the Cost

By Definition 7 a change to the cost of an appearance of a value v_i in x_i corresponds to a change of the cost associated with arc (v_i, x_i) in the value network of a constraint.

Let C_i, C_{i+1} be two consecutive cardinality constraints with costs and let $cost(C_i, P)$ denote $\sum_{i=1}^{|X(C)|} \sum_{v_j \in D_{x_i}} cost(x_i, v_j)$ assigned by P in C_i. Assume that C_i is consistent and that C_{i+1} differs from C_i by a change in the cost of the occurrence of value v_i in x_i. By Definition 6 if C_i is consistent then there exists an assignment P which satisfies cardinality and occurrence restrictions, and such that $cost(C_i, P) \leq H$.

Proposition 1. *The modified constraint C_{i+1} is inconsistent if and only if the cost of an occurrence of any value v_j in x_i such that $f^{C_i}_{v_j x_i} = 1$* increases *with $k > H - cost(C_i, P)$ and there is no path from v_j to x_i in $R(f^{C_i}) - \{(v_j, x_i)\}$ of length $d_{v_j x_i} \leq H - cost(C_i, P) - cost(x_i, v_j)$.*

Proof. By Theorem 1 v_j appears in a set assigned to x_i by P if there exists, corresponding to P, a feasible flow in the value network of C_i in which $f^{C_i}_{v_j x_i} = 1$. Thus, by Definition 6, if $f^{C_i}_{v_j x_i} = 0$ then $cost(C_{i+1}, P) = cost(C_i, P)$ and the modified constraint is consistent.

Moreover, if $f^{C_i}_{v_j x_i} = 1$, and the cost of an appearance of v_j in x_i decreases with any $k \leq cost(x_i, v_j)$, then $cost(C_{i+1}, P) \leq (C_i, P) \leq H$. Thus, the modified constraint is consistent. Since, by Definition 6, $cost(x_i, v_j)$ is non-negative it can be decreased at most with $k = cost(x_i, v_j)$.

Trivially, If $f^{C_i}_{v_j x_i} = 1$ and $cost(x_i, v_j)$ increases with $k \leq H - cost(C_i, P)$ then $cost(C_{i+1}, P)$is lower than or equal to H, thus the modified constraint is consistent.

If $cost(x_i, v_j)$ increases with $k > H - cost(C_i, P)$, then the sum of the costs assigned by P is greater than H. However if there exists a simple path from v_i to x_i of length $d_{v_j x_i}$ in $R(f^{C_i}) - \{(v_j, x_i)\}$ then, by Theorem 2, a new feasible flow in the value network of the constraint can be obtained by reducing the flow through (v_j, x_i). Moreover if $d_{v_j x_i} \leq H - cost(C_i, P) - cost(x_i, v_j)$ then the cost of the obtained flow is lower than H. Thus, by Theorem 1, the modified constraint is consistent.

On the other hand, if no such path exists then the flow through (v_j, x_i) can not be reduced, which implies that there is no assignment P, which satisfies Property 3 of Definition 6. Similarly, if the length of the path $d_{v_j x_i} > H - cost(C_i, P) - cost(x_i, v_j)$, then there is no flow in the value network of the modified constraint whose cost is lower than or equal to H. Thus by Theorem 1, there exists no assignment which satisfies Property 3 of the Definition 6, which indicates inconsistency of the modified constraint. □

This theorem gives us a method for verifying the consistency of a cardinality constraint with dynamically changed costs. However, in some situations it is interesting to maintain the minimum cost flow in the value network of a constraint. To maintain such a flow we use the sensitivity analysis as described in Section 4.

First, note that changing the cost of the occurrence of a value with k units changes the cost of the corresponding arc with the same number of units (Definition 6). By Theorem 3, when decreasing the cost of the occurrence of a value v_j in x_i we are concerned with recomputing the minimum cost flow in the value network of a cardinality constraint only if the reduced cost of the corresponding arc becomes lower than 0.

Proposition 2. *Let f^{C_i} be a minimum cost flow in the value network of $N(C_i)$.*

1. *If $f^{C_i}_{v_j x_i} = 1$ then f^{C_i} is a minimum cost flow in the value network of the modified constraint.*

2. *If $f^{C_i}_{v_j x_i} = 0$ then a minimum cost flow in the value network of the modified constraint is obtained by augmenting a flow through (v_j, x_i) along the shortest path from x_i to v_j in $R(f^{C_i}) - \{(x_i, v_j)\}$ if the length $d_{x_i v_j}$ of this path is less than $k - d_{v_j x_i}$.*

Proof. If $f^{C_i}_{v_j x_i} = 0$ and there is no path from x_i to v_j in $R(f^{C_i}) - \{(x_i, v_j)\}$ then, by Theorem 2 there exists no feasible flow in the value network of the constraint which contains (v_j, x_i). If such a path exists, then redirecting the flow along this path will reduce the total cost of the flow if and only if $d_{x_i v_j} + d_{v_j x_i} - k < 0$, that is: $d_{x_i v_j} < k - d_{v_j x_i}$.

If $f^{C_i}_{v_j x_i} = 1$ then, by Definition 4 and 7, there exists an arc (x_i, v_j) in $R(f^{C_i})$ such that $c_{x_i v_j} = -c_{v_j x_i}$. If $cost(x_i, v_j)$ decreases then the cost $c_{x_i v_j}$ increases and by Definition 5 also $c^{\pi}_{x_i v_j}$ increases. Since $c^{\pi}_{x_i v_j}$ is non-negative for the original problem it is positive for the modified problem. Thus, by Theorem 3, f^{C_i} is the minimum cost flow in the value network of the modified constraint. □

In the case when there is no new minimum cost flow which can be computed according to Proposition 2, the reduced cost optimality conditions are maintained by increasing potential of v_j with k.

Decreasing the cost of an arc (v_j, x_i) where $f^{C_i}_{v_j x_i} = 0$, in a way that forces rerouting of the flow through (v_j, x_i), changes the total cost of the flow with $d_{x_i v_j} + c^{\pi}_{v_j x_i} - k$ where $c^{\pi}_{v_j x_i} - k < 0$. If decreasing the cost of (v_j, x_i) with k units, when $f^{C_i}_{v_j x_i} = 1$, then the total cost of the flow will also decrease with k.

The following proposition forms the basis for a method for recomputing a minimum cost flow, in the value network of a constraint where the cost of the occurrence of a value in a subset increases.

Proposition 3. *Let f^{C_i} be a minimum cost flow in the value network of $N(C_i)$ and let the cost of the occurrence of a value v_j in a subset x_i increase with k units.*

1. *If $f^{C_i}_{v_j x_i} = 0$ then f^{C_i} is the minimum cost flow for also the modified constraint.*
2. *If $f^{C_i}_{v_j x_i} = 1$, and the shortest path from v_j to x_i in $R(f^{C_i}) - \{(v_j, x_i)\}$ has length $d_{v_j x_i} < c_{v_j x_i} + k$, then a new minimum cost flow is obtained by reducing the flow through (v_j, x_i) along this path. If no such path exists, or if the length of the shortest path $d_{v_j x_i} > c_{v_j x_i} + k$, then f^{C_i} is a minimum cost flow for the modified problem.*

Proof. If $f^{C_i}_{v_j x_i} = 0$ and f^{C_i} is a minimum cost flow then $c^{\pi}_{v_j x_i} \geq 0$ and $c^{\pi}_{v_j x_i} + k > 0$ which satisfies the reduced costs optimality conditions of Theorem 3.

If $f^{C_i}_{v_j x_i} = 1$ and there is no path from v_j to x_i in $R(f^{C_i}) - \{(v_j, x_i)\}$ then, by Theorem 2, the flow through (v_j, x_i) can not be reduced without violating its feasibility. If such a path exists, but its length is greater then or equal to $c_{v_j x_i} + k$, then the cost of the flow obtained by redirecting the flow along this path is greater than or equal to $cost(f^{C_i}) + k$; thus, the obtained flow is not a minimum cost flow. On the other hand, if a path which satisfies this properties exists, then a new flow can be obtained by redirecting the flow from (v_j, x_i) along this path. Moreover, if this path is a shortest path from v_j to x_i, then the obtained flow is a minimum cost flow in the value network of the modified constraint. $\square$

In the case when $f^{C_i}_{v_j x_i} = 1$ and $f^{C_{i+1}} = f^{C_i}$, the cost of $f^{C_{i+1}}$ is equal to $cost(f^{C_i}) + k$. Such case requires decreasing the potential of x_i with k in order to satisfy reduced cost optimality conditions. When the flow through (v_j, x_i) is reduced, $cost(f^{C_{i+1}}) = cost(f^{C_i}) + d_{v_j x_i} - c_{x_i v_j}$.

Theorem 4. *If the cost of an occurrence of value v_j in x_i is changed, then the consistency of the cardinality constraint can be checked, and the minimum cost flow in its value network maintained in $O(m + log\ n)$ time, where $m = \sum_{i=1}^{|X|} |D_{x_i}| + |X| + |D| + 2$ and $n = |X| + |D| + 2$.*

Proof. By Proposition 1, verifying the consistency of a cardinality constraint modified by changing the cost of the occurrence of a value in a subset assigned to a variable yields in the worst case one iteration of the shortest path algorithm. Similarly, by Proposition 2 and 3, restoring the minimum cost flow in the value network of the modified constraint is achieved by one iteration of the same algorithm.

Dijkstra's algorithm, implemented with Fibonacci heaps, has time complexity $O(m + n\ log\ n)$. In the residual graph of a flow in the value network of a cardinality constraint $m = \sum_{i=1}^{|X|} |D_{x_i}| + |X| + |D| + 2$ and $n = |X| + |D| + 2$ which proves the theorem. $\square$

Note that recomputing the consistency of a cardinality constraint from scratch using the successive shortest path algorithm, which is one of the best algorithms for computing a minimum cost flow (see [1] for comparison) requires time $O((\sum_{i=1}^{|X|} l_{x_i} + \sum_{j=1}^{|D|} l_{v_j}) \times (m + n\ log\ n))$.

6 Changes to Cardinality Bounds

By Definition 7 a change to a lower or upper bound of the cardinality of a variable x_i corresponds to a change of the lower and upper bounds imposed on the flow from x_i to t. Similarly, a change of the cardinality bound for a value v_j corresponds to a change of the bounds imposed on the flow through (s, v_j) in the value network of a cardinality constraint.

First assume that the lower bound of the cardinality of x_i decreases with k units. Trivially, since C_i is consistent then the flow f^{C_i} is feasible. Moreover f^{C_i} also satisfies the lower bound on the flow through (x_i, t) in the value network of the modified constraint. Thus, by Theorem 1, the modified constraint is consistent.

Furthermore, if f^{C_i} is a minimum cost flow in $N(C_i)$ and the residual capacity $r_{tx_i} > 0$, then f^{C_i} satisfies the reduced costs optimality conditions for (x_i, t) in

$R(f^{C_i})$. By Theorem 3 f^{C_i} is the minimum cost flow in the value network of the modified constraint.

However, if r_{tx_i} in $R(f^{C_i})$ is equal to 0, decreasing the lower bound of x_i will introduce a new arc (t, x_i) in the residual graph of f^{C_i} with capacity k and with the reduced cost $c^{\pi}_{tx_i}$ (Definition 4). Such a situation corresponds to changing a "virtual" reduced cost of the introduced arc with $\pm c^{\pi}_{tx_i}$, and a new minimum cost flow can be computed using the methods described in Section 5.

Similarly, if C_i is consistent and the upper bound of the cardinality of x_i increases, then the modified constraint is consistent. As in the case above, a new minimum cost flow in the value graph of the modified constraint is computed only if increasing this bound introduces a new arc (x_i, t) into the residual graph of f^{C_i}. Again, recomputing the minimum cost flow is done by methods described in Section 5.

Consider now the case where the lower bound of x_i increases with k units:

Proposition 4. *The modified constraint is inconsistent if and only if there is no sequence of $k - r_{t,x_i}$ successive paths from t to x_i in $R(f^{C_i}) - \{(t, x_i)\}$ such that $\sum_{i=1}^{k-r_{t,x_i}} d^i_{tx_i} \leq H - cost(f^{C_i})$.*

Proof. If $k \leq r_{tx_i}$ then the flow through (x_i, t) is feasible also for the modified problem.

If there exist $k - r_{t,x_i}$ such paths then, by Theorem 2 there exists a feasible flow in the value network of the modified constraint. Moreover, if $\sum_{iter=1}^{k-r_{tx_i}} d^{iter}_{tx_i} \leq H - cost(f^{C_i})$ then the cost of obtained flow is lower than or equal to H. Thus, by Theorem 1 the modified constraint is consistent. □

Furthermore, modifying the lower bound of the cardinality of x_i with $k \leq r_{tx_i}$ does not influence the reduced cost of (t, x_i). Thus, if f^{C_i} is a minimum cost flow in $N(C_i)$, then it is also the minimum cost flow in the value network of the modified constraint. If $k > r_{tx_i}$, then, if each one of the $k - r_{tx_i}$ successive paths is a shortest path, then at each iteration f^{C_i} will be augmented with a cheapest flow. Consequently the obtained flow will be the minimum cost flow in the value network of the modified constraint.

When a constraint is modified by *decreasing* the upper bound of the cardinality of a variable, then the consistency of the new constraint can be checked in the same way as when the lower bound is increased. The proof is similar to the proof of Proposition 4. Even the minimum cost flow in the value network of the modified constraint is maintained by computing $k - r_{x_i t}$ shortest paths from x_i to t in $R(f^{C_i}) - \{(x_i, t)\}$.

The rules described above apply by symmetry to the changes in bounds restraining the number of occurrence of each value. For each variable v_j a change to the bounds of its occurrence corresponds to the bounds imposed on the flow through respective (s, v_j) (see Definition 7).

Theorem 5. *The consistency of a cardinality constraint C_i, modified by changing the bounds of the cardinality of a variable or the bounds of an occurrence of* *a value with $\pm k$ units, can be done in $O(|k| \times (m + n\ log\ n)$ time. The same goes for maintaining the minimum cost flow in the value network of the modified constraint.*

Proof. As shown above, verifying and recomputing the consistency of a cardinality constraint, modified by changing the bounds of the cardinality of a variable by $\pm k$, requires at most k iterations of the shortest path algorithm. The same number of iterations of the algorithm is required when modifying the number of occurrences of a value.

Moreover, it is shown that if f^{C_i} is a minimum cost flow in $N(C_i)$, then by augmenting and reducing flows along each shortest path a new minimum cost flow is obtained. □

7 Adding and Removing Variables and Values

Removing a variable x_i can be seen as reducing the bounds of its cardinality to the interval $[0, 0]$. This corresponds to modifying the bounds imposed on the flow through (x_i, t) to be in the interval $[0, 0]$, as well as modifying the bounds of each (v_j, x_i) such that $v_j \in D_{x_i}$ is in the same interval. In this case, verifying consistency of the modified constraint as well as computing a new minimum cost flow is done in the same way as for decreasing the upper bound of the cardinality of a variable (see Section 6).

Moreover, it can be proved that due to properties of the value network of a cardinality constraint, reducing the flow through (x_i, t) to 0 will reduce the flow through each arc (v_j, x_i) to 0.

Adding a variable x_i with the cardinality $[l_{x_i}, u_{x_i}]$ and domain D_{x_i} can be seen as modifying the cardinality bounds from $[0, 0]$ to $[l_{x_i}, u_{x_i}]$, which in turns modifies the bounds imposed on the flow through (x_i, t) and bounds of the flow on each (v_j, x_i) such that $v_j \in D_{x_i}$ to $[0, 1]$ and its cost to $cost(x_i, v_j)$. In this case verifying consistency of the modified constraint, as well as recomputing the minimum cost flow in the value network of the modified constraint, is done in the same way as for increasing the lower bound of the cardinality of a variable from 0 to l_{x_i} units.

Note however, that if $l_{x_i} = 0$ we need to verify that each of the arcs added to the value graph satisfies the reduced costs optimality conditions. If the reduced cost of any arc violates those conditions then the new minimum cost flow is computed by the methods described in Theorem 2.

Removing and adding a value from/to a constraint corresponds to decreasing and increasing, respectively, the bounds imposed on the occurrence of such a value. Consequently, checking consistency of the modified constraint, as well as maintaining a minimum cost flow in its value network, is done by methods corresponding to the ones described in Section 6.

Theorem 6. *If a cardinality constraint with costs C_i is modified by removing a variable or value, then recomputing consistency of C_i and maintaining a minimum cost flow in $N(C_i)$ requires $O(\#(z, P) \times (m+n\ log\ n))$ time, where $\#(z, P)$*

is a cardinality of the set to a removed variable assigned by P in C_i or number of occurrence of a removed value in sets assigned by P in C_i.

Recomputing consistency of a cardinality constraint with costs C_i and a minimum cost flow in $N(C_i)$ modified by adding a variable or a value requires $O(u_z \times (m + n\ log\ n))$ time, where u_z is the upper bound of the cardinality of the introduced variable or the upper bound of the number of occurrence of the introduced value.

Proof. By the correspondence between the removal of a variable and restraining the upper bound on the flow through (x_i, t), since the new upper bound of the flow through (x_i, t) equals 0, all the flow through (x_i, t) has to be reduced. By Theorem 1 the amount of flow through (x_i, t) corresponds to the cardinality of the set assigned to x_i by P in C_i.

By the same correspondences we can prove that removing a value v_j requires a number of iterations of the shortest path algorithm equal to the number of occurrences of v_j in the sets assigned to the variables of C_i by P.

Consider now the case of adding a variable. In the worst case augmenting the flow on each (v_j, x_i) reduces the total cost of the flow in the value network of the modified constraints. However, by Definition 7 the amount of flow through (x_i, t) is limited by the upper bound of the cardinality of x_i.

Similarly, the amount of flow which can be augmented to (s, v_j), where v_j is an added value, is limited by the upper bound of the number of occurrences of v_j. □

As mentioned in Section 3 recomputing a symmetric cardinality constraint from scratch requires $O((\sum_{i=1}^{|X|} l_{x_i} + \sum_{j=1}^{|D|} l_{v_j}) \times (m + n\ log\ n))$ time. It is clear that as long as $\sum_{i=1}^{|X|} l_{x_i} + \sum_{j=1}^{|D|} l_{v_j}) - \#(x,P) > 0$, where x is a variable removed from the problem, we will have better time complexity than for computing the consistency of the constraint from scratch. The same applies to the case of removing a value from a constraint.

Moreover, if the amount of flow, required in the value network of a symmetric cardinality constraint with costs, is greater than the upper bound of the introduced variable or value, the time required to recompute the consistency of such a constraint, by the means presented in this section, will be shorter than the time for recomputing the modified constraint from scratch. The same applies to recomputing the minimum cost flow in the value network of the modified constraint.

For global cardinality constraints with costs and constraints of difference with costs, the methods presented here will always have better time complexity.

8 Changes to the Global Cost Limit

Changing the global cost limit H does not have an influence neither on the minimum cost flow in the value network of the constraint nor on the shortest path distances between any pair of nodes in the network.

The value of H can decrease with at most $H - cost(f^{C_i})$ units without violating consistency of C_i. If C_i is consistent and H increases with any number of units then C_{i+1} is consistent even for the modified problem.

9 Filtering

Given a consistent symmetric cardinality constraint with costs C_i we can compute the consistency of each value v_i in the domain of a variable x_i by establishing if there exists a feasible flow f^{C_i} in the value network $N(C_i)$ which contains (v_i, x_i). Typically, it is done by establishing if a minimum cost flow in $N(C_i)$ involves a flow through (v_i, x_i) or if there exists a shortest path from x_i to v_i in $R(f^{C_i}) - \{(x_i, v_i)\}$ of length $d_{x_i v_i} \leq H - cost(f^{C_i})$ [10].

Although, it is not required by definition of the constraint to compute minimum cost flow in the value network to verify consistency of the constraint, it has a great practical impact on the filtering phase. If the computed flow f^{C_i} in $N(C_i)$ is not a minimum cost flow then the residual graph of f^{C_i} will contain arcs with negative costs (see Theorem 3). In such case verifying the consistency of each value with the constraint would require less efficient algorithms.

Assuming that the minimum cost flow $f^{C_{i+1}}$ is maintained using the sensitivity analysis methods described in Sections 4 – 8 we can recompute the consistency of each value v_i in domain of x_i, as in case of a static cardinality constraint, i.e. by $|X|$ iterations of Dijkstra's algorithm. This gives overall time complexity for filtering the constraint $O(|X|*(m+n\ log\ n))$, where $m = |X|+|D|+\sum_{i=1}^{|X|} D_{x_i}+2$ and $n = |X| + |D| + 2$.

Note that the all-pairs shortest algorithm by Demetrescu, Italiano and Emiliozzi ([7], see even [8]) computes shortest paths between all nodes in a graph in $O(m + n^2 log\ n)$, where m is bounded by n^2. Moreover, the algorithm supports queries about shortest paths between nodes and their lengths in $O(1)$. Assuming that the information about the all-pairs shortest paths in a residual graph of a minimum cost flow in the value network of C_i has been stored, we can use this information during sensitivity analysis of a cardinality constraint with changed costs of occurrence of a value in a subset of a variable. Moreover, the stored information can be used to filter domain of variables in a cardinality constraint with changed global cost limit H.

Demetrescu and Italiano even introduce a dynamic version of the all-pairs shortest path algorithm [8], which recomputes all-pairs shortest paths in $O(n^2 log^2 n)$ for edge insertion, deletion or a edge weight change. Nevertheless, to use this algorithm for our purpose we need to support addition, deletion and weight change of multiple edges.

Dynamic shortest path for symmetric cardinality constraint

The update operation in the algorithm by Demetrescu and Italiano ([8]) propagates changes in the path distances for each changed weight in a graph. Nevertheless, changes to the residual graph in a symmetric cardinality constraint with costs can be more complex than that. Consider for example a situation where a change to the cost of the occurrence of some value in a subset assigned to a variable causes redirection of the flow in the value network of the constraint along a path p consisting of $|p|$ edges. Since any redirection of the flow would mean deleting a (q, r) edge and inserting respective (r, q) edge each of the mentioned algorithms would perform $2 * |p|$ iterations. Moreover, assume that (q, r) and (u, v) are i'th resp. j'th arcs in p and that $i < j$. After replacing (q, r) by (r, q) each of described algorithms would update all shortest paths even those which includes (u, v). Since (u, v) would be replaced at later iteration this would cause a lot of unnecessary computation.

In the following sections we show how the algorithm by Demetrescu and Italiano can be modified to support weight changes for the multiple edges.

Basic Definitions and Notation Let $G = (V, E, w)$ be a directed graph with non-negative edge weights, where V denote a set of nodes, E is a set of edges and w a weight. Let w_{uv} be a weight of $(u, v) \in E$ and p_{xy} a path from $x \in V$ to $y \in V$. The length of the path p_{xy} is denoted by d_{xy} and is the sum of the weights of edges in p_{xy}.

Moreover, let $l(xy)$ be a path p_{xb} such that $p_{xy} = p_{xb} \cdot (b, y)$, i.e. p_{xy} is the concatenation of path p_{xb} and an edge (b, y). Similarly, let $r(xy)$ denote a path p_{ay} such that $p_{xy} = (x, a) \cdot p_{ay}$.

Definition 8. *A path p_{xy} is **uniform** in G if every proper subpath of p_{xy} is a shortest path in G.*

By Definition 8 we can verify that if p_{xy} is a uniform path in G then each $l(p_{xy})$ and $r(p_{xy})$ are the shortest paths in G. Any path which contains a single edge is a uniform path.

Definition 9. *A path p_{xy} is a **zombie** in G at time t if it is not a shortest path but used to be a shortest path at a time $t' < t$, and none of its edges have been updated in the time interval $[t', t]$.*

Moreover, **a historical shortest path** is a path which is either a shortest path or a zombie.

Definition 10. *A path p_{xy} is **potentially uniform** in G at time t if every proper subpath of p_{xy} is a historical shortest path at that time.*

Data Structures The following data structures are maintained for each pair of nodes $x \in V$ and $y \in V$.

- the distance $d_{xy} \geq 0$ of the edge (x, y) or $+\infty$ if no such edge exists
- the time t_{xy} when the edge (x, y) was last updated
- a priority queue $P_{xy} = \{p_{xy} : p_{xy}$ is potentially uniform in $G\}$, where each item $p_{xy} \in P_{xy}$ has a priority equal to the length of p_{xy}.

– $P^*_{xy} = \{p_{xy} : p_{xy}$ is a historical shortest path in $G\}$ maintained as a set

For each path $p_{xy} \in P_{xy}$ the following data structures are maintained.

– d_{xy}, the length of the path from x to y
– $L(p_{xy}) = \{p_{x'y} = (x', x) \cdot p_{xy} : p_{x'y}$ is potentially uniform in G
– $L^*(p_{xy}) = \{p_{x'y} = (x', x) \cdot p_{xy} : p_{x'y}$ is historically shortest path in G
– $R(p_{xy}) = \{p_{xy'} = p_{xy} \cdot (y, y') : p_{xy'}$ is potentially uniform in G
– $R * (p_{xy}) = \{p_{xy'} = p_{xy} \cdot (y, y') : p_{xy'}$ is historically shortest path in G

Finally, a counter *time* keeps the number of performed `update` operations.

The Modified Algorithm To avoid unnecessary computation during the `update` procedure we modify the Demetrescu-Italiano algorithm by allowing weight changes to multiple edges. Both arc insertion and deletion are treated as in the original algorithm: deletion by changing the weight of an arc to $+\infty$ and insertion by changing the weight from $+\infty$ to the actual weight.

Allowing changes to multiple arcs require, in first place, modifying the `cleanup` procedure by allowing as a parameter to the procedure a list of edges with changed weight. Furthermore each modified edge (u, v) such that $(u, v) \in P_{uv}$ is added to the priority queue Q. After that `cleanup` procedure continues as in Algorithm 1.

Note that each redirection of the flow between variable and value nodes will result in deleting one edge and inserting an edge with opposite direction.

Similarly, we pass to `fixup` a list of modified edges. Moreover, during the Phase 1 we parse the list of modified arcs and update respective paths with new edges. Phase 2 and Phase 3 of the algorithm continue in the same way as in Algorithm 1.

Note also that the Demetrescu-Italiano algorithm, even in its modified version, recomputes all-pairs shortest paths in graphs with only non-negative edge weights. Thus this computation need to be perform using reduced costs instead of the original costs.

Theorem 7. *The modified Demetrescu-Italiano algorithm which allows weight changes to multiple edges computes recomputes all-pair shortest paths in $O(n^2 log^2 n)$. Moreover, `path` and `distance` queries are performed in optimal time.*

Proof. First, querying procedures are the same as in the original algorithm. They are proved in [8] to be optimal.

Furthermore, the `cleanup` operation removes in the worst case at most $O(n^2)$ affected paths from the sets of potentially uniform and historical shortest paths.

Phase 1 of the `fixup` procedure involves an operation of inserting each modified edge into the priority queue. The insertion operation in a priority queue can be performed in $O(log\ n)$ time. The number of edges in the residual graph is at most n^2: thus, in the worst case, when all of existing edges are removed and replaced by edges in the opposite direction the complexity of Phase 1 is

Algorithm 1 The original Demetrescu-Italiano algorithm for computing all-pairs shortest paths in fully dynamic graphs

```
update(start-node $u$, end-node $v$, new wedge $w$)
  1. $time \leftarrow time + 1$
  2. $time_{uv} \leftarrow time$
  3. unsmoothed-update(u,v,w)
  4. for each edge $(x, y) : time - time_{xy} = 2^{\lceil log_2(time - time_{x,y} \rceil)}$ do
  5.   unsmoothed-update($x$,$y$,$w_{xy}$)
unsmoothed-update($u$,$v$,$w$)
  1. cleanup($u$,$v$)
  2. fixup($u$,$v$,$w$)
cleanup($u$,$v$)
  1. if $(u, v)$ is potentially uniform then
  2.     $Q \leftarrow \{(u, v)\}$
  3.     while $Q \neq \emptyset$ do
  4.         extract any path $p_{xy}$ from $Q$
  5.         remove $p_{xy}$ from $P_{xy}$
  6.         if $p_{xy} \in P^*_{xy}$ then
  7.             remove $p_{xy}$ from $P^*_{xy}$, $L^*(r(p_{xy}))$ and $R^*(l(p_{xy}))$
  8.         add paths in $L(p_{xy})$ and paths in $R(p_{xy})$ to $Q$
fixup($u$, $v$, $w$)
  1. $d_{uv} \leftarrow w$                                                  {Phase 1}
  2. if $w < +\infty$ then
  3.     $d_{uv} \leftarrow w$; $l((u, v)) \leftarrow p_{uu}$; $r((u, v)) \leftarrow p_{vv}$
  4.     add $(u, v)$ to $P_{uv}$, $L(p_{uv})$, $R(p_{uv})$
  5. $H \leftarrow \emptyset$                                                  {Phase 2}
  6. for each $(x, y)$ do
  7.      add $p_{xy} \in P_{xy}$ with minimum $d_{xy}$ to $H$
  8. while $H \neq \emptyset$ do                                                  {Phase 3}
  9.     extract $p_{xy}$ from $H$ with minimum $d_{xy}$
  10.     if $p_{xy}$ is the first extract path for pair $(x, y)$ then
  11.         if $p_{xy} \notin P^*_{xy}$ then
  12.             add $p_{xy}$ to $P^*_{xy}$, $L^*(r(p_{xy}))$ and $R^*(l(p_{xy}))$
  13.             for each $p_{x'b} \in L^*(l(p_{xy}))$ do
  14.                 $p_{x'y} \leftarrow (x', x) \cdot p_{xy}$
  15.                 $d_{x'y} \leftarrow d_{x'x} + d_{xy}$
  16.                 $l(p_{x'y}) \leftarrow p_{x'b}$ ; $r(p_{x'y}) \leftarrow p_{xy}$
  17.                 add $p_{x'y}$ to $P_{x'y}$, $L(p_{xy})$, $R(p_{x'b})$ and $H$
  18.             for each $p_{ay'} \in R^*(r(p_{xy}))$ do
  19.                 $p_{xy'} \leftarrow p_{xy} \cdot (y, y')$
  20.                 $d_{xy'} \leftarrow d_{xy} + d_{yy'}$
  21.                 $l(p_{xy'}) \leftarrow p_{xy}$ ; $r(p_{xy'} \leftarrow p_{ay'}$
  22.                 add $p_{xy'}$ to $P_{xy'}$, $L(p_{ay'})$, $R(p_{xy})$ and $H$
distance($x$,$y$)
  1. if $P_{xy} = \emptyset$ return $+\infty$
  2. else return minimum distance path in $P_{xy}$
path($x$,$y$)
  1. if $P_{xy} = \emptyset$ then return $\emptyset$
  2. else return shortest path in $P_{xy}$
```

$O(2n^2\ log\ n) = O(n^2\ log\ n)$, which is the same as for recomputing all-pairs shortest paths from scratch.

Phase 2 and 3 of `fixup` have the same time complexity as in the original algorithm. Extracting a minimum from the priority queue P_{xy} is performed in a constant time. Similarly, inserting of an element into H, where H is a Fibonacci heap, require a constant amortized time. Thus inserting $O(n^2)$ elements require $O(n^2)$ amortized time.

In Phase 3 a path p_{xy} is processed if it is not already in P^*_{xy} before the update. This implies that this phase requires $O(n^2\ log\ n)$ time for extracting $O(n^2)$ pairs initially in H, plus $O(p) = O(n^2\ log\ n)$ amortized time for inserting p new potentially uniform paths. The time required by the `fixup` procedure is then in the worst cases $O(2n^2\ log\ n + n^2 + n^2\ log\ n) = O(n^2\ log\ n)$.

Finally, since `update` cause as much as $O(log\ n)$ `unsmoothed-update` operations the overall complexity of the algorithm is $O(n^2\ log\ n\ log\ n) = O(n^2\ log^2 n)$.

$\square$

The modified Demetrescu-Italiano algorithm can be used in a straightforward way to query paths during performing the sensitivity analysis in case of changed cost of the occurrence of a value in a subset assigned to variable. Note however, that shortest paths between the sink and a variable node as well as paths between source and a value node, which are necessary while performing sensitivity analysis in case of changed variable and value bounds and even in case of adding and removing variables and values, has to be queried on a subgraph which does not contain the direct edge between the mentioned nodes. In this case computing a single source shortest path is still faster than maintaining all-pairs shortest paths.

Nevertheless, assuming that sensitivity analysis was already performed and shortest paths in the residual graph are properly maintained, we can perform the filtering phase by querying path lengths from each variable node x_i to each value node v_i such that $f^{C_{i+1}}_{v_i x_i} = 0$. The complexity of this operation is bounded by $O(\sum_{i=1}^{|X|} |D_{x_i})$. In practice, the number of queries can be reduced further by limiting queries only to values, whose consistency with the constraint is influenced directly by the change in the problem, e.g. considering only values inconsistent with C_i when the cost of the flow in $N(C_{i+1})$ decreases etc.

Finally, the overall worst case time complexity for the filtering phase using the modified Demetrescu-Italiano algorithm is $O(n^2\ log^2 n + n^2) = O(n^2\ log^2 n)$ which can be compared with $O(n * (m + n\ log\ n)) = O(n * (n^2 + n\ log\ n))$ for filtering phase, which repeatedly computes single source shortest path from each x_i to each $v_i \in D_{x_i}$. However, advantages and drawbacks of both methods require further investigations.

10 Conclusions and Future Work

In this paper we have introduced the concept of sensitivity analysis for cardinality constraints and show how sensitivity analysis of feasible and minimum cost flows can be adopted to maintain the consistency and minimality of cardinality constraints. We have also analyzed the performance of the methods introduced and show that sensitivity analysis used in this context outperforms recomputing cardinality constraint from the scratch. Analysis of this method is given here on all types changes to a cardinality constraint.

Moreover, we adopt the dynamic all-pairs shortest paths algorithm to handle weight changes on multiple edges. We discuss how the modified algorithm can be used to maintain consistency of all values with a cardinality constraint. Furthermore, we analyze the complexity of the modified all-pairs shortest path algorithm and compare it with the complexity of applying repeatedly single shortest path.

Finally, we give directions for further investigation of the algorithm.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[2] R. Barták. Dynamic global constraints in constraint logic programming. Technical Report ITI Series 2001-018, 2001.

[3] R. Barták, T. Müller, and H. Rudová. A new approach to modelling and solving minimal perturbation problems. In *Recent Advances in Constraints*, LNAI 3010, pages 233–249. Springer Verlag, 2004.

[4] Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. In Gert Smolka, editor, *Proceedings Third International Conference on Principles and Practice of Constraint Programming*, pages 17–31. Springer-Verlag LNCS 1330, 1997.

[5] A.J. Davenport and J.Ch. Beck. Managing uncertainty in scheduling: a survey. Preprint, 2000.

[6] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of AAAI-88*, pages 37–42, 1988.

[7] C. Demetrescu, S. Emiliozzi, and G.F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. url: http://www.dis.uniroma1.it/$\tilde{}$demetres/.

[8] C. Demetrescu and G.F. Italiano. A new approach to dynamic all pairs shortest paths. In ACM, editor, *Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing, San Diego, CA, USA, June 9–11, 2003*, pages 159–166, New York, NY, USA, 2003. ACM Press.

[9] F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *Principles and Practice of Constraint Programming*, pages 189–203, 1999.

[10] W. Kocjan. Symmetric cardinality constraint with costs. Preprint, 2004.

[11] W. Kocjan. Dynamic scheduling. state of the art report. Technical Report T2002-28, Swedish Institute of Computer Science, November 12, 2002.

[12] W. Kocjan and P. Kreuger. Filtering methods for symmetric cardinality constraint. In *Proc. 1th Int. Conf. CPAIOR*, pages 200–208, 2004.

[13] J.-L. Laurière. A language and a program for stating and for solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.

[14] J.-Ch. Régin. A filtering algorithm for constraints of difference in csps. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

[15] J.-Ch. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 209–215, Menlo Park, August 4–8 1996. AAAI Press / MIT Press.

[16] J.-Ch. Régin. Arc consistency for global cardinality constraints with costs. In *Principles and Practice of Constraint Programming (CP-99)*, pages 390–404, 1999.

[17] J.-Ch. Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, (7):387–405, 2002.

[18] E. Tsang. *Foundation of Constraint Satisfaction*. Academic Press, 1993.

[19] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1*, pages 307–312, Menlo Park, CA, USA, July 31–August 4 1994. AAAI Press.