A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint

Willem Jan van Hoeve

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
W.J.van.Hoeve@cwi.nl
http://homepages.cwi.nl/~wjvh/

Abstract. This paper presents an algorithm that achieves hyper-arc consistency for the soft `alldifferent` constraint. To this end, we prove and exploit the equivalence with a minimum-cost flow problem. Consistency of the constraint can be checked in $O(nm)$ time, and hyper-arc consistency is achieved in $O(m)$ time, where n is the number of variables involved and m is the sum of the cardinalities of the domains. It improves a previous method that did not ensure hyper-arc consistency.

1 Introduction

If a constraint satisfaction problem (CSP) is over-constrained, i.e. has no solution satisfying all constraints, it is natural to allow certain constraints, the soft constraints, to be violated and search for solutions that violate as few soft constraints as possible. Constraints that are not decided to be soft are hard constraints, and should always be satisfied.

Several methods have been proposed to handle over-constrained CSPs, see for instance [6, 2, 4]. In this paper, we follow the scheme proposed by Régin, Petit, Bessière and Puget [11], that is particularly useful for non-binary constraints. The idea is as follows. A cost function is assigned to each soft constraint, measuring the violation. Then the soft CSP is transformed into a constraint optimization problem (COP), where all constraints are hard, and the (weighted) sum of cost functions is minimized. This approach allows one to use specialized filtering algorithms for soft constraints, as shown by Petit, Régin and Bessière [7].

For the soft `alldifferent` constraint, an algorithm is presented in [7] that removes inconsistent values in $O(m^2 n\sqrt{n})$ time, where n is the number of variables and m the sum of the cardinalities of their domains. However, that algorithm does not ensure hyper-arc consistency. In this paper, we propose an algorithm that does ensure hyper-arc consistency and runs in $O(nm)$ time. In principle, we consider the soft `alldifferent` constraint as a minimum-cost flow problem in a particular graph. Checking the consistency can then be done in $O(nm)$ time. Thereafter, domain values are checked for consistency by an efficient shortest path computation, which takes in total $O(m)$ time.

The outline of the paper is as follows. Section 2 presents definitions related to constraint satisfaction problems. Section 3 shows a graph-theoretic analysis

of the soft `alldifferent` constraint, using flow theory. In Section 4 the filtering algorithm is presented. We conclude with a discussion in Section 5.

2 Background

We assume familiarity with the basic concepts of constraint programming. For a thorough explanation of constraint programming, see [1].

A constraint satisfaction problem (CSP) consists of a finite set of variables $\mathcal{V} = \{v_1, \ldots, v_r\}$ with finite domains $\mathcal{D} = \{D_1, \ldots, D_r\}$ such that $v_i \in D_i$ for all i, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{V}$. A constraint $C \in \mathcal{C}$ is defined as a subset of the Cartesian product of the domains of the variables that are in C. A tuple $(d_1, \ldots, d_r) \in D_1 \times \cdots \times D_r$ is a solution to a CSP if for every constraint $C \in \mathcal{C}$ on the variables $v_{i_1}, \ldots, v_{i_k}$ we have $(d_{i_1}, \ldots, d_{i_k}) \in C$. A constraint optimization problem (COP) is a CSP together with an objective function to be optimized. A solution to a COP is a solution to the corresponding CSP, that has an optimal objective function value.

Definition 1 (Hyper-arc consistency). *A constraint C on the variables x_1, $\ldots, x_k$ is called hyper-arc consistent if for each variable x_i and value $d_i \in D_i$, there exist values $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_k$ in $D_1, \ldots, D_{i-1}, D_{i+1}, \ldots, D_k$, such that $(d_1, \ldots, d_k) \in C$.*

Definition 2 (Consistent CSP). *A CSP is hyper-arc consistent if all its constraints are hyper-arc consistent. A CSP is inconsistent if it has no solution. Similarly for a COP.*

Definition 3 (Pairwise difference). *Let $x_1, \ldots, x_n$ be variables with respective finite domains $D_1, \ldots, D_n$. Then*

$$\texttt{alldifferent}(x_1, \ldots, x_n) = \{(d_1, \ldots, d_n) \mid d_i \in D_i, d_j \neq d_k \text{ for } j \neq k\}.$$

In [7], two different measures of violation for a soft constraint are presented. The first is the minimum number of variables that need to change their value in order to satisfy the constraint. For this measure, applied to the `alldifferent` constraint, [7] also contains a hyper-arc consistency algorithm. The second measure is the number of violated constraints in the binary decomposition of the constraint, if this decomposition exists. For the `alldifferent` constraint, such a decomposition does exist, namely $x_i \neq x_j$ for $i \in \{1, \ldots, n-1\}, j \in \{i+1, \ldots, n\}$. We follow this second, more refined, measure, and present it in terms of the soft `alldifferent` constraint. For $\texttt{alldifferent}(x_1, \ldots, x_n)$, let the cost of violation be defined as

$$\text{violation}(x_1, \ldots, x_n) = |\{(i, j) \mid x_i = x_j, \text{ for } i < j\}|. \tag{1}$$

Definition 4 (Soft pairwise difference). *Let $x_1, \ldots, x_n, z$ be variables with respective finite domains $D_1, \ldots, D_n, D_z$. Then*

$$\texttt{soft\_alldifferent}(x_1, \ldots, x_n, z) =$$
$$\{(d_1, \ldots, d_n, \tilde{d}) \mid d_i \in D_i, \tilde{d} \in D_z, \text{violation}(d_1, \ldots, d_n) \leq \tilde{d}\}.$$

The variable z in Definition 4 will serve as a so-called cost variable, which will be minimized during the solution process. This means that admissible tuples in Definition 4 are those instantiations of variables, such that the number of violated dis-equality constraints $d_i \neq d_j$ is not more than that of the currently best found solution, represented by $\max D_z$. At the same time, $\min D_z$ should not be less than the currently lowest possible value of violation$(x_1, \ldots, x_n)$.

An over-constrained CSP with an `alldifferent` constraint is transformed into a COP by introducing z, replacing `alldifferent` with `soft_alldifferent`, and minimizing z. This is illustrated in the following example.

Example 1. Consider the following over-constrained CSP

$$x_1 \in \{a, b\}, x_2 \in \{a, b\}, x_3 \in \{a, b\}, x_4 \in \{b, c\},$$
$$\texttt{alldifferent}(x_1, x_2, x_3, x_4).$$

We transform this CSP into

$$z \in \{0, \ldots, 6\},$$
$$x_1 \in \{a, b\}, x_2 \in \{a, b\}, x_3 \in \{a, b\}, x_4 \in \{b, c\},$$
$$\texttt{soft\_alldifferent}(x_1, x_2, x_3, x_4, z),$$
$$\texttt{minimize } z.$$

This COP is not hyper-arc consistent, as there is no support for $z < 1$. If we remove 0 from D_z, the COP is hyper-arc consistent, because there are at most 6 simultaneously violated dis-equalities. Suppose now that during the search for a solution, we have found the tuple $(x_1, x_2, x_3, x_4, z) = (a, a, b, c, 1)$, that has one violated dis-equality. Then $z \in \{1\}$ in the remaining search. As the assignment $x_4 = b$ always leads to a solution with $z \geq 2$, b can be removed from D_4. The resulting COP is hyper-arc consistent again.

One should take into account that a simplified CSP is considered in Example 1. In general, a CSP can consist of many more constraints, and also more cost-variables that together with z form an objective function to be minimized.

Throughout this paper, let $m = \sum_{i \in \{1,\ldots,n\}} |D_i|$ for variables $x_1, \ldots, x_n$.

3 Graph-Theoretic Analysis

A directed graph is a pair $G = (V, A)$ where V is a finite set of vertices V and A is a family[1] of ordered pairs from V, called arcs. A pair occurring more than once in A is called a multiple arc. For $v \in V$, let $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ denote the family of arcs entering and leaving v respectively.

A (directed) walk in G is a sequence $P = v_0, a_1, v_1, \ldots, a_k, v_k$ where $k \geq 0$, $v_0, v_1, \ldots, v_k \in V$, $a_1, \ldots, a_k \in A$ and $a_i = (v_{i-1}, v_i)$ for $i = 1, \ldots, k$. If there is no confusion, P may be denoted as $P = v_0, v_1, \ldots, v_k$. A (directed) walk is called a (directed) path if $v_0, \ldots, v_k$ are distinct. A closed (directed) walk, i.e. $v_0 = v_k$, is called a (directed) circuit if $v_1, \ldots, v_k$ are distinct.

[1] A family is a set in which elements may occur more than once.

3.1 Minimum-cost flow problem

First, we introduce the concept of a flow, following Schrijver [12, pp. 148–150].

Let $G = (V, A)$ be a directed graph and let $s, t \in V$. A function $f : A \to \mathbb{R}$ is called a flow from s to t, or an $s - t$ flow, if

$$\begin{array}{ll} (i) & f(a) \geq 0 \qquad\qquad\qquad\quad \text{for each } a \in A, \\ (ii) & f(\delta^{\text{out}}(v)) = f(\delta^{\text{in}}(v)) \text{ for each } v \in V \setminus \{s, t\}, \end{array} \tag{2}$$

where $f(S) = \sum_{a \in S} f(a)$ for all $S \subseteq A$. Property (2)(ii) ensures flow conservation, i.e. for a vertex $v \neq s, t$, the amount of flow entering v is equal to the amount of flow leaving v.

The value of an $s - t$ flow f is defined as

$$\text{value}(f) = f(\delta^{\text{out}}(s)) - f(\delta^{\text{in}}(s)).$$

In other words, the value of a flow is the net amount of flow leaving s, which can be shown to be equal to the net amount of flow entering t.

When we study flows we typically endow capacity constraints, via a "capacity" function $c : A \to \mathbb{R}_+$. We say that a flow f is under c if $f(a) \leq c(a)$ for each $a \in A$. A feasible flow is a flow under c.

We also assign costs to flows via a "cost" function $w : A \to \mathbb{R}_+$. Doing so the cost of a flow f is defined as

$$\text{cost}(f) = \sum_{a \in A} w(a) f(a).$$

A minimum-cost flow is an $s-t$ flow under c of maximum value and minimum cost. The minimum-cost flow problem is the problem of finding such a minimum-cost flow.

A minimum-cost flow can be computed using an algorithm originally due to Ford and Fulkerson [5] (we follow the description given by Schrijver [12, pp. 183–185]). It consists of successively finding shortest (with respect to the cost function) $s-t$ paths in the so-called residual graph, while maintaining an optimal flow.

Define the residual graph $G_f = (V, A_f)$ of f (with respect to c), where

$$A_f = \{a \mid a \in A, f(a) < c(a)\} \cup \{a^{-1} \mid a \in A, f(a) > 0\}.$$

Here $a^{-1} = (v, u)$ if $a = (u, v)$. We extend w to $A^{-1} = \{a^{-1} \mid a \in A\}$ by defining

$$w(a^{-1}) = -w(a)$$

for each $a \in A$.

Any directed path P in G_f gives an undirected path in $G = (V, A)$. We define $\chi^P \in \mathbb{R}^A$ by

$$\chi^P(a) = \begin{cases} 1 \text{ if } P \text{ traverses } a, \\ -1 \text{ if } P \text{ traverses } a^{-1}, \\ 0 \text{ if } P \text{ traverses neither } a \text{ nor } a^{-1}, \end{cases}$$

Algorithm 1 Minimum-cost $s-t$ flow

```
set f = 0
while termination criterion not satisfied do
  compute minimum-cost s − t path P in G_f
  if no s − t path in G_f exists then
    terminate
  else
    set ε maximal, such that 0 ≤ f + εχ^P ≤ c
    reset f = f + εχ^P
  end if
end while
```

for $a \in A$. Define the cost of a path P as $\text{cost}(P) = \sum_{a \in P} w(a)$.

Call a feasible flow extreme when it has minimum cost among all feasible flows with the same value. Then the following holds (cf. [12, Theorem 12.3 and 12.4]). Let $\mathbf{0}$ denote the all-zero vector of appropriate size.

Theorem 1. *A flow f is extreme if and only if each directed circuit of G_f has nonnegative cost.*

Theorem 2. *Let f be an extreme flow in $G = (V, A)$. Let P be a minimum-cost $s-t$ path in G_f, for some $s, t \in V$, and let $\varepsilon > 0$ be such that $f' = f + \varepsilon\chi^P$ satisfies $\mathbf{0} \leq f' \leq c$. Then f' is an extreme flow again.*

In fact, for f, P, ε and f' in Theorem 2 holds

$$\begin{aligned} \text{value}(f') &= \text{value}(f) + \varepsilon, \\ \text{cost}(f') &= \text{cost}(f) + \varepsilon \cdot \text{cost}(P). \end{aligned}$$

This means that we can find a minimum-cost $s-t$ flow in G by successively computing minimum-cost $s-t$ paths in G_f. Along such a path we increase the amount of flow to the maximum possible value ε. By Theorem 2, the last flow (of maximum value) we obtain must be extreme, and hence optimal. This is presented as Algorithm 1. Note that the cost of minimum-cost $s-t$ paths in G_f is bounded, because there are no directed circuits of negative cost in G_f. For rational capacities, Algorithm 1 terminates with a feasible $s-t$ flow of maximum value and minimum cost. Although faster algorithms exist for general minimum-cost flow problems, Algorithm 1 suffices when applied to our problem. This is because in our particular graph Algorithm 1 is faster than the algorithms for general minimum-cost flow problems.

3.2 From `soft_alldifferent` to minimum-cost flow

We transform the problem of finding a solution to the `soft_alldifferent` constraint into a minimum-cost flow problem.

Construct the directed graph $G = (V, A)$ with

$$V = \{s, t\} \cup X \cup D_X$$

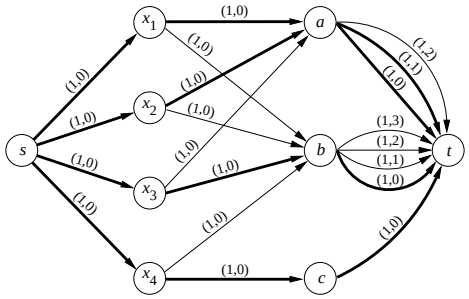

Fig. 1. Graph G for the `soft_alldifferent` constraint of Example 1. For each arc a, $(c(a), w(a))$ is given. Bold arcs indicate an optimal $s-t$ flow with cost 1.

and

$$A = A_X \cup A_s \cup A_t$$

where

$$X = \{x_1, \ldots, x_n\},$$
$$D_X = \bigcup_{i \in \{1,\ldots,n\}} D_i,$$

and

$$A_X = \{(x_i, d) \mid d \in D_i\},$$
$$A_s = \{(s, x_i) \mid i \in \{1, \ldots, n\}\},$$
$$A_t = \{(d, t) \mid d \in D_i, i \in \{1, \ldots, n\}\}.$$

Note that A_t contains parallel arcs if two or more variables share a domain value. If there are k parallel arcs (d,t) between some $d \in D_X$ and t, we distinguish them by numbering the arcs as $(d,t)_0, (d,t)_1, \ldots, (d,t)_{k-1}$ in a fixed but arbitrary way.

To each arc $a \in A$, we assign a capacity $c(a) = 1$ and a cost $w(a)$. If $a \in A_s \cup A_X$, then $w(a) = 0$. If $a \in A_t$, so $a = (d,t)_i$ for some $d \in D_X$ and integer i, the value of $w(a) = i$.

In Figure 1, the graph G for the `soft_alldifferent` constraint in Example 1 is depicted. For each arc a, $(c(a), w(a))$ is given.

Theorem 3. *An integer flow f that is a solution to the minimum-cost flow problem in G corresponds to an instantiation of variables $x_1, \ldots, x_n$ in* `soft_alldifferent`$(x_1, \ldots, x_n, z)$*, minimizing* violation$(x_1, \ldots, x_n)$*.*

Proof. For an integer flow f in G, $f(a) = 1$ if arc a is used, and $f(a) = 0$ otherwise. An arc $a = (x_i, d) \in A_X$ with $f(a) = 1$ corresponds to the instantiation $x_i = d$. By construction, every solution f to the minimum-cost flow problem in G has value$(f) = n$. Thus a solution corresponds to assigning a value to each variable x_i, $i \in \{1, \ldots, n\}$.

The cost function $w(a_i) = i$ for k parallel arcs $a_0, \ldots, a_{k-1} \in A_t$ corresponds to counting the number of violations caused by assigning $i+1$ variables to a particular value. Namely, for these parallel arcs, a minimum-cost $s-t$ path in G_f uses the arc with lowest cost first. Using arc a_i (the $(i+1)$st

arc) causes a "violation" with the i previously used arcs. Thus, for a feasible flow f, which corresponds to an assignment of $x_1, \ldots, x_n$, $\sum_{a \in A} w(a)f(a)$ measures exactly violation$(x_1, \ldots, x_n)$. Hence, a minimum-cost flow minimizes violation$(x_1, \ldots, x_n)$. □

Consider again the graph G in Figure 1. A bold arc a in G denotes $f(a) = 1$. This particular flow f has value$(f) = 4$ and cost$(f) = 1$. Indeed, the only violation is $x_1 = a = x_2$.

Next we describe the behaviour of Algorithm 1 to compute a minimum-cost flow in G. We need to compute a sequence of minimum-cost $s - t$ paths in G_f, maintaining extreme intermediate flows. Note that along each minimum-cost $s-t$ path in G_f we can increase the flow by a maximum of $\varepsilon = 1$. Hence all extreme flows in G are integer. By construction, there are exactly n such paths, each containing one arc in A_s (in fact, the paths may as well be computed starting from the vertices x_i instead of s, using only arcs in A_X and A_t). Further, each minimum-cost $s - t$ path contains exactly one arc in A_t. Namely, consider a minimum-cost path P using multiple arcs in A_t. Then P consists of an $s-t$ path with one arc in A_s, followed by a $t-t$ path. If the $t-t$ path has cost 0, we may omit this part, and use only the $s-t$ path with one arc in A_s. If the $t-t$ path, which is a circuit, has negative cost, it contradicts Theorem 1. Effectively, it means that the $t - t$ path could have been used to improve the preceding intermediate solution, thus contradicting the extremity of that solution. To conclude, the minimum-cost paths we need to compute use exactly one arc in A_s and one arc in A_t. It follows that these paths can be computed in $O(m)$ time, and the total time complexity for finding a maximum flow of minimum cost in G is $O(nm)$. Hence it follows, by Theorem 3, that consistency of the `soft_alldifferent` constraint can be checked in $O(nm)$ time.

4 The Filtering Algorithm

The following theorem identifies hyper-arc consistent domain values for the `soft_alldifferent` constraint. For an arc a of G, let G^a arise from G by enforcing $f(a) = 1$ for every flow f in G.

Theorem 4. *The constraint* `soft_alldifferent`$(x_1, \ldots, x_n, z)$ *is hyper-arc consistent if and only if*

- (i) *for all all arcs $a \in A_X$ a minimum-cost flow of maximum value in G^a has cost at most* $\max D_z$,
- (ii) *all values in D_z are not smaller than the cost of a minimum-cost flow of maximum value in G.*

Proof. Enforcing $f(a) = 1$ for arc $a = (x_i, d)$ corresponds to assigning $x_i = d$. The result follows from Definition 1 and Theorem 3. Namely, property (i) checks consistency for all domain values in $D_1, \ldots, D_n$. Property (ii) checks consistency of the domain values of D_z. □

Algorithm 2 Naive hyper-arc consistency

```
set minimum = ∞
for x_i ∈ X do
  for d ∈ D_i do
    compute minimum-cost s − t flow f in G^a where a = (x_i, d)
    if cost(f) > max D_z then
      remove d from D_i
    end if
    if cost(f) < minimum then
      set minimum = cost(f)
    end if
  end for
end for
if min D_z < minimum then
  set min D_z = minimum
end if
```

Using Theorem 4, we can construct an algorithm that enforces hyper-arc consistency for the `soft_alldifferent` constraint, presented as Algorithm 2. For all variables $x_i \in X$, the algorithm scans all domain values $d \in D_i$, and checks whether there exists a minimum-cost $s-t$ flow in G^a, where $a = (x_i, d)$, of maximum value with cost at most $\max D_z$. If such a flow does not exist, then, by Theorem 4, d is removed from D_i. Finally, we remove all values of D_z which are smaller than the cost of a minimum-cost flow in G. The time complexity of Algorithm 2 is $O(m^2 n)$.

We can construct a more efficient filtering algorithm, however. It is presented as Algorithm 3, and makes use of the following theorem. We follow the notation introduced in Section 3.1.

Theorem 5. *Let f be an extreme flow of maximum value in G. Let $a = (x_i, d) \in A_X$ and P a minimum-cost $d - x_i$ path in G_f. Let $f^\star$ be an extreme flow of maximum value in G^a. Then* $\text{cost}(f^\star) = \text{cost}(f) + \text{cost}(P)$.

Proof. Either $f(a) = 1$ or $f(a) = 0$. In case $f(a) = 1$, $f^\star(a) = 1$, $P = d, x_i$, $\text{cost}(P) = 0$ and we are done. In case $f(a) = 0$, first note that there exists a $d - x_i$ path in G_f. Namely, there is exactly one $d' \in D_i$ for which $f((x_i, d')) = 1$, which allows the path d, t, d', x_i. Let P be a minimum-cost $d - x_i$ path in G_f. Together with arc (x_i, d) P forms a circuit C. The directed circuit C in G_f gives an undirected circuit in G^a. For all $b \in A$, define flow $f^\star$ in G^a as follows:

$$f^\star(b) = \begin{cases} 0 & \text{if } b^{-1} \in C \\ 1 & \text{if } b \in C \\ f(b) & \text{else.} \end{cases}$$

It is easy to check that $f^\star$ is again a flow of maximum value.

Because f is extreme, we may assume that P enters and leaves t only once, say via arcs b_{in} and b_{out} respectively (where $b_{\text{in}} = (d, t)$). It follows that $\text{cost}(P) =$

Algorithm 3 More efficient hyper-arc consistency

```
compute minimum-cost flow f in G
if cost(f) > max D_z then
  return INCONSISTENT
end if
if min D_z < cost(f) then
  set min D_z = cost(f)
end if
for a = (x_i, d) with f(a) = 0 do
  compute minimum-cost d − x_i path P in G_f
  if cost(f) + cost(P) > max D_z then
    remove d from D_i
  end if
end for
```

$w(b_{\text{in}}) - w(b_{\text{out}})$. From Theorem 1 we know that $\text{cost}(P) \geq 0$. Similarly,

$$\begin{aligned}\text{cost}(f^\star) &= \textstyle\sum_{b \in A} f^\star(b) w(b) \\ &= \textstyle\sum_{b \in A} f(b) w(b) + w(b_{\text{in}}) - w(b_{\text{out}}) \\ &= \text{cost}(f) + \text{cost}(P)\end{aligned}$$

It remains to show that $f^\star$ is extreme in G^a. Suppose not, i.e. there exists a flow g in G^a with maximum value and $\text{cost}(g) < \text{cost}(f^\star)$. As $\text{cost}(f^\star) = \text{cost}(f) + \text{cost}(P)$ and $\text{cost}(P) \geq 0$, there are two possibilities. The first is that $\text{cost}(g) < \text{cost}(f)$, which is not possible because f is extreme. The second is that there exists an $x_i - d$ path P' in G_f with $\text{cost}(P') < \text{cost}(P)$ which also leads to a contradiction because P is a minimum-cost path. Hence $f^\star$ is extreme. □

Algorithm 3 first computes a minimum-cost flow f in G. This takes $O(nm)$ time, as we have seen in Section 3.2. If $\text{cost}(f) > \max D_z$, we know that the `soft_alldifferent` constraint is inconsistent. If this is not the case, we update $\min D_z$. Next, we scan all arcs $a = (x_i, d)$ for which $f(a) = 0$. For each of these arcs, we compute a minimum-cost $d - x_i$ path P in G_f. By Theorem 5 and Theorem 4, we remove d from D_i if $\text{cost}(f) + \text{cost}(P) > \max D_z$. This can be done efficiently, as shown by the following theorem.

Theorem 6. *Let* `soft_alldifferent`$(x_1, \ldots, x_n, z)$ *be consistent and* f *an integer minimum-cost flow in* G*. Then* `soft_alldifferent`$(x_1, \ldots, x_n, z)$ *can be made hyper-arc consistent in* $O(m)$ *time.*

Proof. The complexity of the filtering algorithm depends on the computation of the minimum-cost $d - x_i$ paths inG_f for arcs (x_i, d). We make use of the fact that only arcs $a \in A_t$ contribute to the cost of such path.

Consider the strongly connected components[2] of the graph $\tilde{G}_f$ which is a copy of G_f where s and t and all their incident arcs are removed. Let P be a minimum-cost $d - x_i$ path P in G_f. If P is equal to d, x_i then $f(x_i, d) = 1$ and $\text{cost}(P) = 0$.

[2] A strongly connected component in a directed graph $G = (V, A)$ is a subset of vertices $S \subseteq V$ such that there exists a directed $u - v$ path in G for all $u, v \in S$.

Otherwise, either x_i and d are in the same strongly connected component of $\tilde{G}_f$, or not. In case they are in the same strongly connected component, P can avoid t in G_f, and $\text{cost}(P) = 0$. In case x_i and d are in different strongly connected components, P must visit t, and we do the following.

Split t into two vertices t^{in} and t^{out} such that $\delta^{\text{in}}(t^{\text{in}}) = \delta^{\text{in}}(t)$, $\delta^{\text{out}}(t^{\text{in}}) = \varnothing$, and $\delta^{\text{in}}(t^{\text{out}}) = \varnothing$, $\delta^{\text{out}}(t^{\text{out}}) = \delta^{\text{out}}(t)$. For every vertex $v \in X \cup D_X$ we can compute the minimum-cost path from v to t^{in} and from t^{out} to v in total $O(m)$ time.

The strongly connected components of $\tilde{G}_f$ can be computed in $O(n + m)$ time, following Tarjan [14]. Hence the total time complexity of achieving hyper-arc consistency is $O(m)$, as $n < m$. □

The proof of Theorem 6 applies to any constraint whose graph representation resembles G and has only costs on arcs from D_X to t. For all such constraints that are consistent, hyper-arc consistency can be achieved in $O(m)$ time. Note that this is equal to the complexity of achieving hyper-arc consistency on these constraints if no costs are involved.

5 Conclusion and Discussion

We have presented an algorithm that checks consistency of the `soft_alldifferent` constraint on n variables in $O(nm)$ time and achieves hyper-arc consistency in $O(m)$ time, where m is the sum of the cardinalities of the domains. A previous method for removing domain values that are inconsistent with the `soft_alldifferent` constraint did not ensure hyper-arc consistency [7]. Moreover, that method has a time complexity of $O(m^2 n\sqrt{n})$. Hence our algorithm improves on this in terms of quality as well as time complexity.

The `soft_alldifferent` constraint is related to the standard `alldifferent` constraint [8] and the minimum weight `alldifferent` constraint [3]. The minimum weight `alldifferent` constraint is a particular instance of the global cardinality constraint with costs [9, 10]. For that constraint, hyper-arc consistency can be achieved in $O(n(m+d\log d))$ time, where d is the cardinality of the union of all domains [9, 10, 13]. It is achieved by finding n shortest paths, each taking $O(m + d\log d)$ time to compute. Although our algorithm has a similar flavour, the underlying graphs have a different cost structure. We improve the efficiency by exploiting the cost structure of our particular graph when computing the shortest paths. Our result can be applied to other constraints with a similar graph representation and cost structure.

Acknowledgements

Many thanks to Bert Gerards and Lex Schrijver for valuable comments. Thanks also go to Sebastian Brand and Jean-Charles Régin for fruitful discussion.

References

1. K.R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
2. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2):201–236, 1997.
3. Y. Caseau and F. Laburthe. Solving Various Weighted Matching Problems with Constraints. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 17–31. Springer, 1997.
4. M.C. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 2004. To appear.
5. L.R. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.
6. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.
7. T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. In T. Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *LNCS*, pages 451–463. Springer, 2001.
8. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 362–367, 1994.
9. J.-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. In J. Jaffar, editor, *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer, 1999.
10. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.
11. J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An Original Constraint Based Approach for Solving over Constrained Problems. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *LNCS*, pages 543–548. Springer, 2000.
12. A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.
13. M. Sellmann. An Arc-Consistency Algorithm for the Minimum Weight All Different Constraint. In P. Van Hentenryck, editor, *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *LNCS*, pages 744–749. Springer, 2002.
14. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.