

Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction*

J. H. M. Lee and K. L. Leung

Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{jlee,klleung}@cse.cuhk.edu.hk

Abstract

Powerful consistency techniques, such as AC* and FDAC*, have been developed for Weighted Constraint Satisfaction Problems (WCSPs) to reduce the space in solution search, but are restricted to only unary and binary constraints. On the other hand, van Hoeve *et al.* developed efficient graph-based algorithms for handling soft constraints as classical constraint optimization problems. We prove that naively incorporating van Hoeve's method into the WCSP framework can enforce a strong form of \emptyset -Inverse Consistency, which can prune infeasible values and deduce good lower bound estimates. We further show how Van Hoeve's method can be modified so as to handle cost projection and extension to maintain the stronger AC* and FDAC* generalized for non-binary constraints. Using the soft `allDifferent` constraint as a testbed, preliminary results demonstrate that our proposal gives improvements up to an order of magnitude both in terms of time and pruning.

1 Introduction

The task at hand is on how to relax or weaken some of the hard constraints in an over-constrained problem so as to obtain useful partial solutions. Weighted constraint satisfaction [Schiex *et al.*, 1995] is a framework for handling such tasks. While the basic technique for solving weighted constraint satisfaction problems (WCSPs) relies on a form of branch-and-bound search, various consistency notions and techniques [Larrosa and Schiex, 2003; 2004; Sanchez *et al.*, 2008] for unary, binary, and ternary constraints have been developed to help prune the search space. Higher arity constraints have to be either first converted to their binary counterparts or activated only after enough variables are instantiated during search. The lack of efficient handling of non-binary constraints in WCSP systems greatly restricts the ap-

plicability of WCSP techniques to complex real-life problems.

Incorporating arbitrary soft n -nary constraints into WCSP can be difficult since the costs have to be represented extensionally and maintained in an n -dimensional table, incurring time and space overheads. Soft global constraints are non-binary constraints with semantics. In particular, the cost structure of flow-based soft global constraints [van Hoeve *et al.*, 2006] can be formulated as a flow network, allowing the computation of the minimum cost of the soft global constraints using minimum cost flow algorithm. This is useful in estimating the lower bound of the current search path. We show that a naive incorporation of flow-based soft global constraints into WCSP would result in a strong form of the \emptyset -inverse consistency [Zytnicki *et al.*, 2006], which is still relatively weak in terms of lower bound estimation and pruning. The question becomes whether we can *achieve stronger consistencies, the generalized versions of AC* [Larrosa and Schiex, 2004] and FDAC* [Larrosa and Schiex, 2003], for non-binary constraints efficiently.* Consistency algorithms for AC* and FDAC* involve three main operations: (a) computing the minimum cost of the constraint when a variable x is fixed with value v , (b) projecting the minimum cost of a constraint to the unary constraint for x at value v , and (c) extending the unary cost to the non-unary constraints. These operations allow cost movement among constraints and shifting of cost to the C_\emptyset constraint, resulting in higher lower bound and also domain prunings. Part (a) is readily handled by the minimum cost flow (MCF) algorithm. We show how the MCF algorithm and the corresponding flow networks can be adapted for parts (b) and (c) so as to perform projection and extension in polynomial time and space complexity. Using the soft `allDifferent` constraint as a testbed, we demonstrate the advantages of the stronger consistencies over naive incorporation.

2 WCSP

The weighted CSP (WCSP) framework extends classical constraint satisfaction by associating costs to the tuples of variable assignments. A WCSP [Schiex *et al.*, 1995] is a tuple (X, D, C, k) . X is a set of variables $\{x_1, x_2, \dots, x_n\}$ ordered by their indices. D is a set of domains $D(x_i)$ for $x_i \in X$. Each x_i can only be assigned one value in its corresponding domain. An assignment on a set of variables can

*We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by grants CUHK413207 and CUHK413808 from the Research Grants Council of Hong Kong SAR.

be represented by a tuple ℓ . We denote $\ell[x_i]$ as the value assigned to x_i , and $\ell[S]$ as the tuple formed from the assignment on a subset of variables S . C is a set of constraints, each C_S of which represents a function mapping tuples corresponding to assignments on S to a cost valuation structure $V(k) = ([0 \dots k], \oplus, \leq)$. The structure $V(k)$ contains a set of integers $[0 \dots k]$ with standard integer ordering \leq . Addition \oplus is defined by $a \oplus b = \min(k, a + b)$, and subtraction \ominus is defined by $a \ominus b = a - b$ if $a \neq k$ and $k \ominus a = k$ for any a .

Without loss of generality, C_S can always be defined (initially with all tuples mapping to zero) for all $S \subseteq X$. The cost of a tuple ℓ corresponding to an assignment on X is defined as:

$$\text{cost}(\ell) = C_\emptyset \oplus \bigoplus_{C_S \in C} C_S(\ell[S]),$$

where C_\emptyset is a null constraint that denotes the lower bound of costs of all possible tuples. A tuple ℓ corresponding to an assignment on X is *feasible* if $\text{cost}(\ell) < k$, and is a *solution* of a WCSP if ℓ has the minimum cost among all feasible tuples.

A *soft global constraint* C_S on variables S has a particular semantics and *can have more than one cost measure*. Where necessary, we sometimes give also a *separate* cost function μ in case C_S has more than one such function. For simplicity, we assume that when we write $C_S(\ell)$ or $\mu(\ell)$, ℓ is always a feasible assignment to S .

WCSPs are solved with basic branch-and-bound search augmented with consistency techniques which prune infeasible values from variable domains and push lower bound estimates into C_\emptyset . Common consistency notions and techniques [Larrosa and Schiex, 2003; 2004; Sanchez *et al.*, 2008] include NC*, AC*, and FDAC*, but are designed for unary to ternary constraints only.

A variable x_i is NC* if (1) each value $v \in D(x_i)$ satisfies $C_{x_i}(v) \oplus C_\emptyset < k$ and (2) there exists a value $v' \in D(x_i)$ such that $C_{x_i}(v') = 0$. A WCSP is NC* iff all variables are NC*. Algorithm 1 enforces NC*. Function `unaryProject()` projects costs from unary constraints to C_\emptyset by simple arithmetic operations, and `pruneVal()` removes infeasible values from domains.

3 Enforcing \emptyset IC and Strong \emptyset IC

In this section, we explain how van Hoesve's method of using minimum cost flow can be adapted for WCSPs to enforce a strong form of \emptyset -inverse consistency [Zytnicki *et al.*, 2006]. Given a connected flow network $G(V, E, w, c, s, t)$, where V are the vertices, E are the edges, and each edge $e \in E$ has a weight w_e and a capacity c_e . A flow f from a source s to a sink t of a value α in G is defined as a mapping from E to \mathbb{R} such that :

- $\sum_{(s,u) \in E} f_{su} = \sum_{(u,t) \in E} f_{ut} = \alpha$;
- $\sum_{(u,v) \in E} f_{uv} = \sum_{(v,u) \in E} f_{vu} \forall v \in V \setminus \{s, t\}$;
- $0 \leq f_e \leq c_e \forall e \in E$.

If α is not defined, α is the maximum value of all flows in G . The cost of a flow f is defined as $\sum_{e \in E} w_e f_e$. A soft global constraint C_S with cost function μ is *flow-based* if μ allows for a representation in a flow network G so that the flow with

Procedure `enforceNC*()`

```

1 | foreach  $x_i \in X$  do unaryProject( $x_i$ );
2 | foreach  $x_i \in X$  do pruneVal( $x_i$ );

```

Procedure `unaryProject`(x_i)

```

3 |  $\alpha := k$ ;
4 | foreach  $v \in D(x_i)$  do
5 |   | if  $\alpha > C_{x_i}(v)$  then  $\alpha := C_{x_i}(v)$ ;
6 |  $C_\emptyset := C_\emptyset \oplus \alpha$ ;
7 | foreach  $v \in D(x_i)$  do  $C_{x_i}(v) := C_{x_i}(v) \ominus \alpha$ ;
8 | return  $\alpha > 0$ ;

```

Function `pruneVal`(x_i):Boolean

```

9 |  $flag := \text{false}$ ;
10 | foreach  $v \in D(x_i)$  do
11 |   | if  $C_{x_i}(v) \oplus C_\emptyset = k$  then
12 |     |  $D(x_i) := D(x_i) \setminus \{v\}$ ;
13 |     |  $flag := \text{true}$ ;
14 | return  $flag$ ;

```

Algorithm 1: Enforce NC* on a WCSP

minimum cost in G corresponds to the tuple mapping to the minimum cost in C_S . Van Hoesve *et al.* [2006] demonstrate his framework on the soft versions of the `allDifferent`, `gcc`, `regular`, and `same` constraints.

We use the soft `allDifferent` constraint with the μ_{dec} cost measure [Petit *et al.*, 2001] to illustrate the concepts. Given an assignment tuple ℓ on variables S , $\mu_{dec}(\ell) = |\{(i, j) | i < j \wedge \ell[x_i] = \ell[x_j] \wedge x_i, x_j \in S\}|$, which stands for the number of pairs of variables sharing the same value. We can construct a flow network $G(V, E, w, c, s, t)$ as follows [van Hoesve *et al.*, 2006]. The network consists of $|S| + |D| + 2$ nodes, where $|D|$ is the size of the union of all variable domains in S . Each variable and value have an associated node, with two more nodes s and t . The network contains three sets of edges:

- $(s, x_i) \in E$ for each $x_i \in X$ with zero weight and unit capacity;
- $(x_i, v) \in E$ for each $v \in D(x_i)$ with zero weight and unit capacity;
- $(v, t)_i \in E$, for each $i = 1, \dots, d_v$, where d_v is the number of variables containing v . Each edge $(v, t)_i$ has a unit capacity and a weight of $i - 1$.

For example, if $X = \{x_1, x_2, x_3, x_4\}$ with $D(x_1) = \{a, c\}$, $D(x_2) = \{b, d\}$, $D(x_3) = D(x_4) = \{a, d\}$ and `allDifferent`(X), the network is constructed as shown in Figure 1. Only non-zero weights are shown in the network. All edges assume unit capacity. The minimum cost of the feasible flow in G with value $|X|$ is $\min\{\mu_{dec}(\ell)\}$. To compute $\min\{\mu_{dec}(\ell) | \ell[x] = v\}$, the minimum cost flow simply enforces $f_{xv} = 1$. For example, Figure 1 shows a flow (highlighted by thickened edges) of minimum cost when $f_{x_1 a} = 1$. Regin [2002] and Van Hoesve *et al.* [2006] proved that such an enforcement can be derived from an existing flow

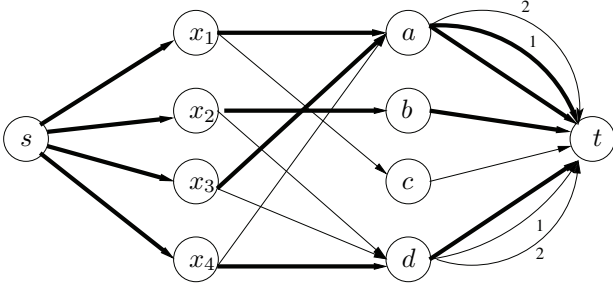


Figure 1: A flow network for `allDifferent`. The thick edges give the minimum cost flow when $x_1 = a$.

by constructing a residual network from G and the existing flow, and finding the minimum cost cycle containing (x, v) in the residual network. This can be found by using the single source shortest path algorithm.

We now define $\emptyset\text{IC}$ [Zytnicki *et al.*, 2006] and strong $\emptyset\text{IC}$ for WCSPs, the enforcement of which can benefit from van Hoeve’s method. A constraint C_S is $\emptyset\text{IC}$ if there exists a tuple ℓ corresponding to a feasible assignment with $C_S(\ell) = 0$. A WCSP is $\emptyset\text{IC}$ iff all constraints are $\emptyset\text{IC}$.

For example, Figure 2(a) shows a WCSP which is not $\emptyset\text{IC}$. No matter which values are assigned to the variables, C_{x_1, x_2}

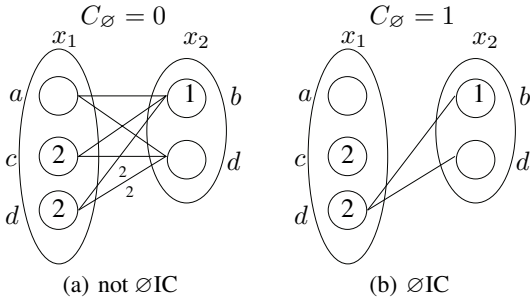


Figure 2: Two equivalent WCSPs with $k = 4$

returns a cost of at least 1. To enforce $\emptyset\text{IC}$, a cost of 1 is projected directly from C_{x_1, x_2} to C_{\emptyset} by reducing the cost of each tuple by 1 and increase C_{\emptyset} by 1. The resultant WCSP is shown in Figure 2(b).

Procedure `enforceEmptySetIC()` in Algorithm 2 enforces $\emptyset\text{IC}$ for a WCSP by enforcing $\emptyset\text{IC}$ on each constraints. Reducing the cost of each tuple can be expensive. An implementation trick is to use a zero-initialized variable z_S to store the cost reduced so far due to projection from C_S to C_{\emptyset} . If a tuple ℓ queries its cost from C_S , the result is $C_S(\ell) \ominus z_S$.

In general, the algorithm is exponential even with the implementation trick since exponential number of tuples have to be examined at line 5. However, minimum cost flow computation allows for a polynomial time algorithm for flow-based soft global constraints, such as `allDifferent` with μ_{dec} .

Enforcing $\emptyset\text{IC}$ only increases C_{\emptyset} . We observe, for example in Figure 2(b), that the value $d \in D(x_1)$ cannot be part of any solution. The tuple associated with $x_1 = d$ has a cost at least 4: 1 from C_{\emptyset} , 2 from C_{x_1} , and 1 from C_{x_1, x_2} . Extra conditions can be added to strengthen $\emptyset\text{IC}$ to allow also

Function `enforceEmptySetIC():Boolean`

```

1  flag := true;
2  foreach  $C_S \in C$  do
3    flag := flag  $\vee$  enforceEmptySetIC( $C_S$ );
4  return flag;
```

Function `enforceEmptySetIC(C_S):Boolean`

```

5   $\alpha := \min\{C_S(\ell)\}$ ;
6  foreach tuple  $\ell$  do  $C_S(\ell) := C_S(\ell) \ominus \alpha$ ;
7   $C_{\emptyset} := C_{\emptyset} \oplus \alpha$ ;
8  return  $\alpha > 0$ 
```

Algorithm 2: Enforcing $\emptyset\text{IC}$ on a WCSP

domain reduction. A non-unary constraint C_S is *strong* $\emptyset\text{IC}$ if:

- C_S is $\emptyset\text{IC}$, and;
- for all values $v \in D(x)$ with $x \in S$, $C_{\emptyset} \oplus C_x(v) \oplus \min\{C_S(\ell) | t[x] = v\} < k$.

A WCSP is *strong* $\emptyset\text{IC}$ iff all constraints are strong $\emptyset\text{IC}$.

For example, the WCSP in Figure 2(b) is not strong $\emptyset\text{IC}$, but removing the value d from $D(x_1)$ makes it so. Procedure `enforceStrongEmptySetIC()` in Algorithm 3 enforces strong $\emptyset\text{IC}$, based on the `W-AC*3()` Algorithm [Larrosa and Schiex, 2004]. The algorithm maintains a propagation queue Q (implemented as a set) of variables. Constraints involving variables in Q are potentially not strong $\emptyset\text{IC}$. Function `pop()` removes an arbitrary available variable from Q in constant time.

Procedure `enforceStrongEmptySetIC()`

```

1   $Q := X$ ;
2  while  $Q \neq \emptyset$  do
3     $x_u := \text{pop}(Q)$ ;
4    foreach  $C_S$  s.t.  $\{x_u\} \subset S$  do
5      foreach  $x_i \in S \setminus \{x_u\}$  do
6        flag := removeInfeasible( $C_S, x_i$ );
7        if flag then  $Q := Q \cup \{x_i\}$ ;
8      if enforceEmptySetIC() then
9         $Q := X$ ;
```

Function `removeInfeasible(C_S, x_i):Boolean`

```

10 flag := false;
11 foreach  $v \in D(x_i)$  do
12    $\alpha := \min\{C_S(\ell) | \ell[x_i] = v\}$ ;
13   if  $C_{\emptyset} \oplus C_{x_i}(v) \oplus \alpha = k$  then
14      $D(x_i) := D(x_i) \setminus \{v\}$ ;
15     flag := true;
16 return flag;
```

Algorithm 3: Enforcing strong $\emptyset\text{IC}$ of a WCSP

Procedure `enforceStrong \emptyset IC()` in Algorithm 3 must terminate, the proof of which is similar to those of Larrosa and Schiex's Theorems 12 and 21 [2004]. Suppose `removeInfeasible()` and `enforce \emptyset IC()` have a time complexity of $O(f_{strong})$ and $O(f_{\emptyset})$ respectively, the complexity can be stated as follows.

Theorem 1 *Procedure `enforceStrong \emptyset IC()` has a time complexity of $O(s_{max}d(s_{max}ef_{strong} + f_{\emptyset}))$, where e is the number of non-unary constraints, s_{max} is the maximum arity of the constraints, n is the number of variables, and d is the maximum domain size. Thus, `enforceStrong \emptyset IC()` must terminate.*

Proof: In each iteration of the while loop, line 6 will be executed $O(s_{max}e)$ times. Each variable is pushed into Q at most $O(d)$ times due to line 7 (each time $D(x_u)$ is modified); thus the while loop will be executed $O(s_{max}d)$ times. Therefore, the complexity of procedure `enforceStrong \emptyset IC()` is $O(s_{max}d(s_{max}ef_{strong} + f_{\emptyset}))$, and it must terminate. ■

Again, `enforceStrong \emptyset IC()` requires exponential complexity since line 12 is exponential in general. However, line 12 can be computed in polynomial time for flow-based soft global constraints. The following result is a consequence of Regin's Lemma 1 [2002] and van Hove *et al.*'s Theorem 1 [2006].

Theorem 2 *If C_S is a flow-based soft global constraint, `removeInfeasible()` has a time complexity of $O(K + d \cdot SP)$, where $O(K)$ and $O(SP)$ are the time complexity to find the minimum cost flow and single source shortest path respectively, and d is the maximum domain size.*

Given a network $G(V, E, w, c, s, t)$. A typical $O(K)$ is $O(|V|^2|E|)$ if the successive shortest path algorithm is used, and a typical $O(SP)$ is $O(|V||E|)$ if a label correcting algorithm, like the Bellman-Ford algorithm, is used [Ahuja *et al.*, 2005].

Due to space limitation, we cannot give the details of the reasoning that the ‘‘Soft as Hard’’ approach [Petit *et al.*, 2001] is slightly weaker than enforcing strong \emptyset IC together with NC*, which is still relatively weak in terms of the deduced lower bound and pruning. Stronger consistencies for soft global constraints are desirable.

4 Projection in GAC*

We specialize the definition of GAC in Cooper *et al.* [2004] for WCSP. A variable $x_i \in S$ is *generalized arc consistent star (GAC*)* with respect to a non-unary constraint C_S if:

- x_i is NC*, and;
- for each value $v_i \in D(x_i)$, there exists values $v_j \in D(x_j)$ for all $j \neq i$ and $x_j \in S$ so that they form a tuple ℓ with $C_S(\ell) = 0$. $\{v_j\}$ is a *simple support* of v_i with respect to C_S .

A WCSP is GAC* iff all variables are GAC* with respect to all constraints. Notice that GAC* collapses to AC* for binary constraints [Larrosa and Schiex, 2004] and AC for ternary constraints [Sanchez *et al.*, 2008].

Procedure `enforceGAC*()` in Algorithm 4 enforces GAC* for a WCSP and is based on the `W-AC*3()` Algorithm [Larrosa and Schiex, 2004]. Algorithm 4 must termi-

Procedure `enforceGAC*()`

```

1  Q := X;
2  while Q ≠ ∅ do
3    x_u := pop(Q);
4    flag := false;
5    foreach C_S s.t. {x_u} ⊂ S do
6      foreach x_i ∈ S \ {x_u} do
7        flag := flag ∨ findSupport(C_S, x_i);
8        if pruneVal(x_i) then Q := Q ∪ {x_i};
9    if flag then
10   foreach x_i ∈ X do
11     if pruneVal(x_i) then Q := Q ∪ {x_i};

```

Function `findSupport(C_S, x_i):Boolean`

```

12  flag := false;
13  foreach v ∈ D(x_i) do
14    α := min{C_S(ℓ) | ℓ[x_i] = v};
15    if C_{x_i}(v) = 0 ∧ α > 0 then flag := true;
16    C_{x_i}(v) := C_{x_i}(v) ⊕ α;
17    foreach tuple ℓ with ℓ[x_i] = v do
18      C_S(ℓ) := C_S(ℓ) ⊖ α;
19  unaryProject(x_i);
20  return flag;

```

Algorithm 4: Enforcing GAC* for a WCSP

nate, the proof of which is similar to that of Theorem 1. By replacing $O(f_{strong})$ and $O(f_{\emptyset})$ by $O(f_{GAC})$ (the complexity of `findSupport()`) and $O(nd)$ (n times the complexity of `pruneVal()`) respectively, the complexity of Algorithm 4 can be stated as follows.

Theorem 3 *Procedure `enforceGAC*()` has a time complexity of $O(s_{max}d(es_{max}f_{GAC} + nd))$, where n , d , e , and s_{max} are as defined in Theorem 1. Thus, `enforceGAC*()` must terminate.*

Again, Algorithm4 requires exponential time complexity since function `findSupport()` is exponential. The time complexity of `findSupport()` is determined by two operations: minimum cost computation (line 14) and cost projection (lines 16 to 18). Line 14 computes the minimum cost of C_S when $x_i = v$. Line 16 projects the cost to the unary constraint C_{x_i} , which is a simple arithmetic operation. Lines 17 and 18 update the cost of all tuples corresponding to $x_i = v$. In general, this two sub-procedures require exponential time complexity, which can be reduced for flow-based soft global constraints. Van Hove's method can be applied similarly to line 14 as in Section 3. Lines 17 to 18 modify the cost function of the soft (global) constraint C_S . Before we give our method, we state the conditions under which our method is applicable.

A soft global constraint C_S with cost function μ is *projection-safe* if

- the soft global constraint C_S with cost function μ is flow-based, and has the corresponding flow network

$G(V, E, w, c, s, t)$,

- there is a one-one correspondence between every flow f of G and a complete variable assignment tuple ℓ for C_S , and
- there exists an injection from an assignment $x_i = v$ to $\bar{e} \in E$ such that whenever $\ell[x_i] = v$ for some tuple ℓ , $f_{\bar{e}} = 1$ in the flow f corresponding to ℓ ; whenever $\ell[x_i] \neq v$, $f_{\bar{e}} = 0$.

Given a projection-safe soft global constraint C_S with cost function μ defined above. Suppose a cost of α is projected from C_S to C_{x_i} associated with $x_i = v$, resulting in a new cost function μ' . In other words, $\mu'(\ell) = \mu(\ell) \ominus \alpha$ if $\ell[x_i] = v$; otherwise $\mu'(\ell) = \mu(\ell)$. We construct the corresponding flow network of C_S with cost function μ' as $G'(V, E, w', c, s, t)$, where $w'_e = w_e \ominus \alpha$ if e is the edge corresponding to $x_i = v$; otherwise $w'_e = w_e$.

We use again the `allDifferent` constraint with μ_{dec} as an example. Figure 1 shows the corresponding flow network and the flow representing $(x_1, x_2, x_3, x_4) = (a, b, a, d)$ with cost 1. If a cost of 1 is projected from the constraint to C_{x_1} associated with $x_1 = a$, a new network can be constructed by decreasing the weight $w_{x_1 a}$ of the edge (x_1, a) from 0 to -1 , as shown in Figure 3. The new cost of the flow in the network is now 0, which corresponds to the cost of the tuple (a, b, a, d) after projection.

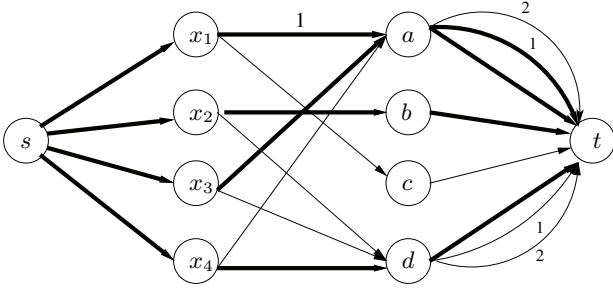


Figure 3: The flow network corresponding to `allDifferent` after projection.

The soundness and closure of our method are guaranteed by the following theorem.

Theorem 4 Suppose C_S is a soft global constraint with cost function μ is projection safe, a cost of α associated with $x_i = v$ is projected from C_S to C_{x_i} , resulting in a new cost function μ' .

- (Soundness) If f is a minimum cost flow of $G'(V, E, w', c, s, t)$, then $\sum_{e \in E} w'_e f_e = \min\{\mu'(\ell)\}$.
- (Closure) C_S with cost function μ' is projection safe.

Proof: Projection-safety implies that $\sum_{e \in E} w'_e f_e = \sum_{e \in E} w_e f_e \ominus \alpha f_{\bar{e}} = \min\{\mu(\ell)\} \ominus \alpha f_{\bar{e}} = \min\{\mu'(\ell)\}$, where \bar{e} is the edge corresponding to $x_i = v$. This concludes soundness.

In addition, C_S with μ' is flow-based with $G'(V, E, w', c, s, t)$ as the corresponding flow network. Since the topology of $G'(V, E, w', c, s, t)$ is the same as that of $G(V, E, w, c, s, t)$, C_S with μ' is projection-safe. ■

We state without proof that the majority of the flow-based global constraints [van Hoeve *et al.*, 2006] are projection-safe so that `GAC*` can be enforced on them in polynomial time.

Theorem 5 The following flow-based soft global constraints are projection-safe.

- `allDifferent` with either μ_{var} or μ_{dec} ;
- `gcc` with either μ_{var} or μ_{val} ;
- `same` with μ_{var} ;

Unfortunately, the `regular` constraint with either μ_{var} or μ_{edit} [van Hoeve *et al.*, 2006] and the soft `SEQUENCE` constraint [Maher *et al.*, 2008] are not projection-safe since they do not satisfy the third requirement.

Again, the complexity of enforcing `GAC*` for a variable with respect to the projection-safe soft global constraints follows from van Hoeve *et al.*'s Theorem 1.

Theorem 6 If C_S is projection safe, `findSupport()` has a time complexity of $O(K + d \cdot SP)$, where K and SP are as defined in Theorem 2.

Based on `FDAC*` [Larrosa and Schiex, 2003], even stronger consistency can be defined but its enforcement involves an extension operator, which is the reverse of projection and the focus of the next section.

5 Extension in `FDGAC*`

Suppose variables are ordered by their indices. A variable $x_i \in S$ is *directional generalized arc consistent star* (`DGAC*`) with respect to a non-unary constraint C_S if:

- x_i is `NC*`, and;
- for each value $v_i \in D(x_i)$, there exists values $v_j \in D(x_j)$ for all $j \neq i$ and $x_j \in S$ so that they form a tuple ℓ with $C_S(\ell) \oplus \bigoplus_{j>i \wedge x_j \in S} C_{x_j}(v_j) = 0$. $\{v_j\}$ is a *full support* of v_i with respect to C_S .

A `WCSP` is *full directional generalized arc consistent star* (`FDGAC*`) if all variables are `DGAC*` and `GAC*` with respect to all non-unary constraints. When the constraints are binary, `FDGAC*` collapses to `FDAC*` [Larrosa and Schiex, 2004]. When the constraints are binary and ternary, however, `FDGAC*` differs slightly from `FDAC` [Sanchez *et al.*, 2008]. `FDGAC*` requires full supports with only zero unary costs, while `FDAC` [Sanchez *et al.*, 2008] requires full supports with not only zero unary but also zero binary costs.

Based on the `FDAC*()` Algorithm [Larrosa and Schiex, 2003], procedure `enforceFDGAC*()` in Algorithm 5 enforces `FDGAC*`. Q and R store variables which are potentially not `GAC*` and not `DGAC*` respectively. Function `popMax()` always removes the variable with the largest index from R in constant time. Procedure `enforceFDGAC*()` in Algorithm 5 must terminate, the proof of which is similar to those of Larrosa *et al.*'s Theorems 3 and 4 [2003]. Suppose `findFullSupport()` and `findSupport()` are of order $O(f_{DGAC})$ and $O(f_{GAC})$ respectively, the complexity of procedure `enforceFDGAC*()` can be stated as follows.

```

Procedure enforceFDGAC*()
1  R := Q := X;
2  while R ≠ ∅ ∨ Q ≠ ∅ do
3    while Q ≠ ∅ do
4      xu := pop(Q);
5      flag := false;
6      foreach CS s.t. {xu} ⊂ S do
7        foreach xi ∈ S \ {xu} do
8          if findSupport(CS, xi) then
9            R := R ∪ {xi};
10           flag := true;
11         if flag then
12           foreach xi ∈ X s.t. pruneVal(xi) do
13             Q := Q ∪ {xi};
14       while R ≠ ∅ do
15         xu := popMax(R);
16         if pruneVal(xu) then Q := Q ∪ {xu};
17         foreach CS s.t. {xu} ⊂ S do
18           for i = u - 1 DownTo 1 s.t. xi ∈ S do
19             if findFullSupport(CS, xi) then
20               R := R ∪ {xi};
21         foreach xi ∈ X s.t. pruneVal(xi) do
22           Q := Q ∪ {xi};

```

```

Function findFullSupport(CS, xi):Boolean
23 foreach j > i and xj ∈ S do
24   foreach v ∈ D(xj) do
25     foreach tuple ℓ with ℓ[xj] = v do
26       CS(ℓ) := CS(ℓ) ⊕ Cxj(vj);
27       Cxj(vj) := 0;
28   flag := findSupport(CS, xi);
29 foreach j > i and xj ∈ S do findSupport(CS, xj);
30 unaryProject(xi);
31 return flag;

```

Algorithm 5: Enforcing FDGAC* on a WCSP

Theorem 7 *enforceFDGAC*() has a time complexity of $O(s_{max}^2 ed(n f_{DGAC} + f_{GAC}) + n^2 d^2)$, where n , d , e , and s_{max} are defined in Theorem 1. Thus, *enforceFDGAC*() must terminate.**

Again, the complexity can be exponential due to `findSupport()` and `findFullSupport()`. In the following, we focus the discussion on `findFullSupport()`. The first part (lines 23 to 27) performs *extension*, a reversal of projection, to push all the unary costs back to C_S . By the time we execute line 28, all unary costs are 0, and enforcing GAC* for x_i achieves the second requirement of DGAC*. Line 29 re-instates GAC* for all variables x_j , where $j > i$. Note that the success in line 28 guarantees that $C_{x_j}(v_j) = 0$ if v_j appears in a tuple ℓ which makes $C_S(\ell) = 0$.

The key idea to performing extension properly is similar to that of projection: the method is applicable to a projection-safe soft global constraint C_S with cost function μ . Suppose now we want to extend a cost of α associated with $x_i = v$

from C_{x_i} to C_S resulting in a new cost function μ'' . In other words, $\mu''(\ell) = \mu(\ell) \oplus \alpha$ if $\ell[x_i] = v$; otherwise $\mu''(\ell) = \mu(\ell)$. We construct the corresponding flow network of C_S with cost function μ'' as $G''(V, E, w'', c, s, t)$, where $w''_e = w_e \oplus \alpha$ if e is the edge corresponding to $x_i = v$; otherwise $w''_e = w_e$.

Similarly, extension is both sound and closed.

Theorem 8 *Suppose C_S is a projection safe soft global constraint with cost function μ , and a cost of α associated with $x_i = v$ is extended from C_{x_i} to C_S , resulting in a new cost function μ'' .*

- (Soundness) *If f is a minimum cost flow of $G''(V, E, w'', c, s, t)$, then $\sum_{e \in E} w''_e f_e = \min\{\mu''(\ell)\}$.*
- (Closure) *C_S with cost function μ'' is projection safe.*

The complexity result again follows from van Hoeve *et al.*'s Theorem 1 [2006].

Theorem 9 *If C_S is a projection-safe soft global constraint, `findFullSupport()` has a time complexity of $O(K + s_{max} d \cdot SP)$, where K and SP are as defined in Theorem 2.*

Last but not least, we state the relative strength of the consistencies concerned. Given two consistencies β and γ , β is *stronger* than γ ($\beta \geq \gamma$) if a WCSP P is γ whenever P is β .

Theorem 10 *FDGAC* ≥ GAC* ≥ strong ∅IC ≥ GAC in “Soft as Hard” Approaches.*

6 Experimental Results

To demonstrate the efficiency of our proposals, we have implemented strong ∅IC, GAC*, and FDGAC* for the soft `allDifferent` constraint with the μ_{dec} and μ_{var} cost functions in ToulBar2¹. Our benchmark instances are based on a softened version of the all-interval series problem (CSPLib Prob007). This problem contains mainly `allDifferent` constraints, unconcerning us from other possible external factors and focusing on evaluating the efficiency of our proposed algorithms. Such a benchmark also allows us to study the scaling behavior of our algorithms. The original problem of order n is to find a series $\{x_1, \dots, x_n\}$ such that it is a permutation of $\{0, \dots, n-1\}$ and the adjacent differences $d_i = |x_i - x_{i+1}|$, $i = \{1, \dots, n-1\}$ are distinct. To model its softened version as a WCSP, we use $\{x_i\}$ and $\{d_i\}$ as variables with domains $\{0, \dots, n-1\}$. Two `allDifferent` constraints are placed on $\{x_i\}$ and $\{d_i\}$ respectively. Ternary table constraints are used to enforce $d_i = |x_i - x_{i+1}|$. Besides, random unary constraints are placed on $\{x_i\}$, assigning random costs to each assignment ranging from 0 to 9.

During the experiment, variables $\{x_i\}$ are first assigned in lexicographic order, followed by $\{d_i\}$ in the same order. Value assignments start with the value with minimum unary cost first. The test is conducted in a Dell Optiplex 280 with an Intel P4 3.2GHz CPU and 2GB RAM. The average runtime and number of backtracks of five instances are measured for each value of n with no initial upper bound. Entries are marked with a “*” if the average runtime exceeds the limit of 1 hour.

¹<http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro>

n	Strong \emptyset IC		GAC*		FDGAC*	
	Time(s)	Backtracks	Time(s)	Backtracks	Time(s)	Backtracks
14	9.44	2923.2	4.42	541.4	2.77	160.2
15	15.52	3028.2	6.24	498.4	2.77	105.8
16	71.17	12700.0	20.39	1615.0	10.96	312.8
17	63.54	8180.0	28.15	1768.0	8.21	262.6
18	481.01	57805.0	124.64	6199.6	16.74	427.4
19	*	*	253.18	12186.0	94.87	1708.8
20	3480.34	161152.8	307.24	13440.6	52.69	717.2
21	*	*	*	*	1167.64	9726.2
22	*	*	*	*	2972.06	22942.8
23	*	*	*	*	2321.53	23785.0
24	*	*	*	*	2130.71	27440.5
25	*	*	*	*	3341.50	46228.0

(a) μ_{dec}

n	Strong \emptyset IC		GAC*		FDGAC*	
	Time(s)	Backtracks	Time(s)	Backtracks	Time(s)	Backtracks
14	6.75	2122.8	3.16	332.0	1.12	101.4
15	6.86	1465.4	4.05	289.0	0.99	65.0
16	50.06	9862.2	16.19	1213.6	3.79	197.6
17	50.41	7294.6	27.51	1739.4	5.16	277.2
18	397.86	49902.0	134.65	6340.0	11.42	496.8
19	*	*	263.75	11881.8	68.51	2227.2
20	*	*	316.48	12797.4	36.00	845.0
21	*	*	*	*	470.83	8945.0

(b) μ_{var}

Figure 4: The time in seconds and the number of backtracks in solving softened all-interval series instances by enforcing different consistencies on the soft `allDifferent` constraints with μ_{dec} (top) and μ_{var} (bottom).

We give the results for `allDifferent` with μ_{dec} and μ_{var} in Figure 4, which agrees well with the theoretical comparison of the three consistencies. This demonstrates that minimum cost flow computation is an efficient method for enforcing the consistencies of projection-safe soft global constraints. Despite a higher complexity, FDGAC*, the strongest consistency both in terms of pruning and lower bound reasoning, is the clear winner bettering strong \emptyset IC by one to two orders of magnitude, while GAC* comes in a clear second. In the best case, enforcing FDGAC* can remove 18 times more search nodes than enforcing GAC*, and 220 times more than enforcing strong \emptyset IC. Last but not least, we note that, without strong \emptyset IC, Toulbar2 delays the propagation of n -ary constraints until only two variables restricted by the constraints are not yet assigned. It is impractical to solve the benchmark even with a small value of n .

7 Conclusion

Global constraints are one of the keys for modeling and solving complex real-life problems. To the best of our knowledge, this is the first success report of global constraints in WCSP solvers with practical efficiency. Our techniques make it possible to enforce generalized versions of existing consistencies exploiting specifically characteristics of WCSPs.

Immediate future work includes studies of the implementation of more projection-safe soft global constraints, feasibility of other forms of consistencies, experiments on a wider variety of benchmarks. It is also interesting to investigate if there are other forms of projection-safety.

References

- [Ahuja *et al.*, 2005] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall/Pearson, 2005.
- [Cooper and Schiex, 2004] M. Cooper and T. Schiex. Arc Consistency for Soft Constraints. *Artificial Intelligence*, 154:199–227, 2004.
- [Larrosa and Schiex, 2003] J. Larrosa and T. Schiex. In the Quest of the Best Form of Local Consistency for Weighted CSP. In *Proceedings of IJCAI'2003*, pages 239–244, 2003.
- [Larrosa and Schiex, 2004] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc Consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
- [Maher *et al.*, 2008] M. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *Proceedings of CP'2008*, pages 159–174, 2008.
- [Petit *et al.*, 2001] T. Petit, J.-C. Regin, and C. Bessiere. Specific Filtering Algorithm for Over-Constrained Problems. In *Proceedings of CP'2001*, pages 451–463, 2001.
- [Regin, 2002] J.-C. Regin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.
- [Sanchez *et al.*, 2008] M. Sanchez, S. de Givry, and T. Schiex. Mendelian Error Detection in Complex Pedigrees using Weighted Constraint Satisfaction Techniques. *Constraints*, 13(1):130–154, 2008.
- [Schiex *et al.*, 1995] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of IJCAI'1995*, pages 631–637, 1995.
- [van Hoeve *et al.*, 2006] W.J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming: Flow-based Soft Global Constraints. *J. Heuristics*, 12(4-5):347–373, 2006.
- [Zytnicki *et al.*, 2006] Matthias Zytnicki, Christine Gaspin, and Thomas Schiex. A New Local Consistency for Weighted CSP Dedicated to Long Domains. In *Proceedings of SAC'2006*, pages 394–398, 2006.