# Assignment Problems in Cost Function Networks

**Guidio Sewa[1,2], David Allouche[1], Simon de Givry[1,2], George Katsirelos[2,3,4], Pierre Montalbano[5], Thomas Schiex[1,2]**

[1]MIAT, INRAE, France
[2]Université de Toulouse, ANITI, France
[3]MIA Paris-Saclay, INRAE, France
[4]Agroparistech, Université Paris-Saclay, France
[5]Université de Tours, France
guidio.sewa@inrae.fr, david.allouche@inrae.fr, simon.de-givry@inrae.fr, gkatsi@gmail.com,
pierre.montalbano@univ-tours.fr, thomas.schiex@inrae.fr

## Abstract

To efficiently solve exact discrete optimization problems, branch and bound algorithms require tight bounds. In constraint programming, for optimization, soft arc consistencies typically derive much stronger bounds than those offered by domain or bound consistencies applied to a cost variable. The reason is that soft local consistencies exchange marginal cost information between variables whereas domain consistencies rely only on shrinking domains, which is less informative. However, CP solvers equipped with soft arc consistencies have so far offered limited support for efficient processing of global constraints.

In this work, we show how we can efficiently enforce soft local consistency over the ALLDIFFERENT constraint, relying on algorithms for the Linear Assignment Problem (LAP). We implement this propagator in toulbar2, the state-of-the-art weighted CP solver exploiting soft local consistencies for bounding. We show that, equipped with this new propagator, toulbar2 outperforms state-of-the-art domain consistency-based CP as well as integer programming solvers for the Quadratic Assignment Problem and shows better performance for miniCOP instances of the 2024 XCSP competition with ALLDIFFERENT constraints.

## Introduction

Combinatorial optimization is a problem that has been attacked with techniques from many different disciplines, from inexact methods like metaheuristics and local search, to exact mathematical programming or constraint programming methods. In the context of constraint programming, there are two conflicting approaches: using a CP solver on a model using soft global constraints, or using a Weighted Constraint Satisfaction Problem (WCSP) solver on a model expressed as a graphical model such as a Cost Function Network (Cooper et al. 2010; Schiex, Fargier, and Verfaillie 1995). The latter approach is very successful in certain applications, such as resource management (Bensana, Lemaître, and Verfaillie 1999), bioinformatics (Simoncini et al. 2015; Sanchez, de Givry, and Schiex 2008), machine learning (Zanfir and Sminchisescu 2018), computer vision (Haller et al. 2022), and others (Hurley et al. 2016). The reason for this success is that WCSP solvers use *soft consistencies* to compute bounds. These soft consistencies are enforced with propagation-based algorithms which exchange not only domain pruning information, as standard CP techniques do, but also updates of the unary costs of variables. This deeper communication between variables leads to stronger lower bounds and in turn stronger pruning and improved branching heuristics. Unfortunately, the requirement for this more involved communication means that until recently, only tabular representations of constraints were practical for WCSP solvers. The exception was decomposable global constraints (Allouche et al. 2012). While there were some efforts to integrate global constraints into the WCSP framework (Lee and Leung 2009), the performance was not sufficient to address problems that required such global constraints.

Recently, Montalbano et al. (2022), proposed a method for propagating linear inequalities in WCSP. This method, based on a new consistency called $F\emptyset IC$, proved effective and led to state-of-the-art performance in some domains.

## Our Contributions

In this work, we show how to integrate ALLDIFFERENT constraints in a WCSP solver that uses soft consistencies to derive bounds. Specifically

- We show how to enforce full zero-inverse consistency ($F\emptyset IC$) on a permutation constraint, and how to reformulate a WCSP instance using the primal and dual solutions computed by a solver for the LAP.
- We show how to handle non-permutation ALLDIFFERENT constraints, with strictly more values than variables.
- We implement this algorithm in a state of the art WCSP solver and demonstrate significant performance improvements in standard benchmarks for quadratic assignment (QAPLIB) and for COP (XCSP competition).

## Related Work

ALLDIFFERENT is one of the most studied global constraints in constraint programming. There exist algorithms that enforce generalized arc consistency for both its hard version (where the unary costs are all assumed to be 0)

(Régin 1994) and its weighted version (Sellmann 2002; Claus, Cambazard, and Jost 2020). Additionally there exist algorithms for generalizations such as cost GCC (Régin 1999), where the bounds on the number of occurrences of each value can be arbitrary rather than exactly 1 or at most 1. The main difference between our work and these propagators is they can only communicate with the rest of the problem via domain prunings, *i.e.*, by removing values that may not appear in any feasible solution or any optimal solution. In our approach, communication occurs through the unary costs of variables, leading to stronger propagation and stronger bounds.

The work of Dlask and Savchynskyy (2025) is closest to our own. They showed how to integrate ALLDIFFERENT constraints in a Block-Coordinate Descent (BCD) algorithm, specifically showing how to find points on the relative interior of the polytope, which are known to yield better performance for such algorithms (Werner, Průša, and Dlask 2020). This is in contrast to our approach which is a combinatorial algorithm based on propagation. More importantly, while BCD algorithms and the related VAC (Cooper et al. 2008) have demonstrated the ability to obtain stronger bounds, they are relatively expensive and their use for exact solving has remained limited to a preprocessing step. To our knowledge, no branch-and-bound solver uses them for computing bounds during search despite efforts to do so (Nguyen et al. 2014; Trösser, De Givry, and Katsirelos 2020).

# Background

## Cost Function Networks

A Cost Function Network (CFN) $P$ is a triple $\langle V, D, C\rangle$, where $V$ is a set of variables $1, \ldots, n$, $D$ is a set of domains, where $D^i$ is a function mapping variable $i$ to its domain and $C$ is a set of *cost functions*. An assignment $I$ to a set of variables $S$ is a mapping from each variable $i \in S$ to a value in $D^i$. Let $\ell(S) = \Pi_{i \in S} D^i$ denote the set of all possible assignments for $S$. An assignment is complete if $S = V$, otherwise it is partial. The projection of $I$ to $S' \subset S$ is denoted $I[S']$. Each cost function $c_S \in C, S \subseteq V$, maps each assignment to the variables in $S$ to $\mathbb{N}_{\geq 0} \cup \{\top\}$, where $\top$ indicates infeasibility[1]. We say that $S$ is the *scope* of $c_S$. The cost of a complete assignment $I$ is $c_P(I) = \sum_{c_S \in C} c_S(I[S])$. To simplify notation, we write $c_i$ instead of $c_{\{i\}}$ for cost functions with unary scope and $c_{ij}$ instead of $c_{\{i,j\}}$ for functions with binary scope. We also assume that there always exists $c_i$ for each $i \in V$ and a cost function with empty scope, $c_\varnothing$, which represents a constant in the objective function. All costs being non-negative, $c_\varnothing$ is a lower bound for the cost of any assignment to the variables of the CFN. Finally, we assume there is at most one cost function for a given scope[2]. The Weighted Constraint Satisfaction Problem (WCSP) is the problem of finding an assignment with minimum cost. It is NP-hard (Cooper, de Givry, and Schiex 2020).

[1]It is possible to use rationals instead of integers, but dealing with integer costs makes exact computation simpler.

[2]This assumption does not necessarily hold in practice and is only used to simplify notation, so it does not otherwise affect any of the algorithms discussed.

Algorithm 1: MOVE$(S_1, S_2, I_1, \alpha)$: Move $\alpha$ units of cost between tuple $I_1$ of scope $S_1$ and all tuples $I_2$ that extend $I_1$ in scope $S_2$

**Require:** scopes $S_1 \subset S_2$; tuple $I_1 \in \ell(S_1)$; transfer cost $\alpha$

1: $c_{S_1}(I_1) \leftarrow c_{S_1}(I_1) + \alpha$
2: **for all** $I_2 \in \ell(S_2)$ such that $I_2[S_1] = I_1$ **do**
3: $\quad c_{S_2}(I_2) \leftarrow c_{S_2}(I_2) - \alpha$
4: **end for**

In order to compute lower bounds for WCSPs, we can use very fast static techniques (Dechter 1999) or slightly slower dynamic techniques based on propagation, called soft consistencies (Cooper et al. 2010; Schiex 2000). The latter have proved more effective for branch-and-bound solvers (Hurley et al. 2016), so we focus on them here.

A crucial notion for these bounds is *reformulation*.

**Definition 1** (Equivalence and Reformulation)**.** *Let $P = \langle V, D, C\rangle, P' = \langle V, D, C'\rangle$ be CFNs, then $P$ and $P'$ are* equivalent *if $c_P(I) = c_{P'}(I)$ for all $I \in \ell(V)$. If additionally, the multiset of scopes of $P$ and $P'$ are identical, i.e., $\uplus\{S \mid c_S \in C\} = \uplus\{S \mid c_S \in C'\}$, then $P$ and $P'$ are* reformulations *of each other.*

It has been shown (Kolmogorov 2006) that all such reformulations can be derived as a set of MOVE operations, an example of equivalence preserving transformation (Cooper et al. 2010).

There are three particular cases of MOVE that are of interest in this paper:

- PROJECT-C0$(S, \alpha) = move(\varnothing, S, \varnothing, \alpha)$, where $\alpha > 0$. In this case, we move cost from cost function $c_S$ directly to $c_\varnothing$, increasing the lower bound. If $|S| = 1$, we call this UNARYPROJECT.
- PROJECT$(S_1, S_2, a, \alpha) = Move(S_1, S_2, a, \alpha)$, where $|S_1| = 1, S_2 \supset S_1, \alpha > 0$, which moves cost from a non-unary cost function to the value $a$ of a unary cost function.
- EXTEND$(S_1, S_2, a, \alpha) = Move(S_1, S_2, a, -\alpha)$, where $|S_1| = 1, S_2 \supset S_1, \alpha > 0$, which moves cost from the value $a$ of a unary cost function to a non-unary cost function.

The most important soft consistencies for this work are node consistency and full $\varnothing$-inverse consistency.

**Definition 2** (Node consistency)**.** *A CFN $P$ is Node Consistent (NC) (Larrosa 2002) if for every variable $i \in V$ there exists a value $a \in D^i$ such that $c_i(a) = 0$ and for every value $b \in D^i$, $c_\varnothing + c_i(b) < \top$.*

Enforcing NC allows to eliminate values that are guaranteed to be infeasible with respect to unary cost functions, while ensuring a zero-cost local assignment exists by applying UNARYPROJECT, resulting in increased lower bound $c_\varnothing$. This inference mechanism can be generalized to non-unary cost functions through $\varnothing$-inverse consistency ($\emptyset IC$).

**Definition 3** ($\varnothing$-Inverse Consistency)**.** *A CFN is $\varnothing$-Inverse Consistent ($\emptyset IC$) (Zytnicki et al. 2009) if for every cost function $c_S \in C$ there exists $I \in \ell(S)$ such that $c_S(I) = 0$.*

Similarly to NC, the existence of a zero-cost local assignment can be guaranteed by shifting costs from every cost function $c_S \in C$ to $c_\varnothing$ using PROJECT-C0. This definition can be strengthened by integrating unary cost functions.

**Definition 4** (Full $\varnothing$-Inverse Consistency)**.** *A CFN is Full $\varnothing$-Inverse Consistent ($F\emptyset IC$) (Montalbano, de Givry, and Katsirelos 2022) if for every cost function $c_S \in C$ there exists $I \in \ell(S)$ such that $c_S(I) + \sum_{i \in S} c_i(I[\{i\}]) = 0$.*

State of the art solvers for WCSP by default enforce a variant of arc consistency, called Existential Directional Arc Consistency (De Givry et al. 2005). We do not give a full definition here for lack of space, but we note that it subsumes $F\emptyset IC$. It is, however, challenging to generalize to non-binary constraints (Lee and Leung 2009).

## Linear Programming (LP) and Linear Assignment Problem (LAP)

A linear program is a problem of the form $\min c^T x : Ax = b, x \in \mathbb{R}^n_{\geq 0}$, where $x$ is a vector of $n$ non-negative rational variables, $c$ is a vector in $\mathbb{R}^n$, $b$ is a vector in $\mathbb{R}^m$ and $A$ is a matrix in $\mathbb{R}^{n \times m}$. The problem $\max b^T y : A^T y \leq c, y \in \mathbb{R}^m$, where $y$ is a vector of $m$ rational variables, is called the *dual* problem. Linear programs can be solved in polynomial time (Vaidya 1989). Given a linear program $P$, we write $opt(P)$ for the value of its optimum, and $P'$ for the dual. For any feasible solution of $P'$, *i.e.*, for any $\hat{y}$ such that $A^T \hat{y} \leq c$, it holds that $b^T \hat{y} \leq opt(P)$. Similarly, for any feasible $\hat{x}$, $c^T \hat{x} \geq opt(P')$. The theorem of *strong duality* states that $opt(P) = opt(P')$, *i.e.*, for optimal solutions $(x^*, y^*)$, $c^T x^* = b^T y^*$. Due to the strong mathematical relationship between the primal and dual problems, analyzing the dual provides rich information about the primal. In particular, given a dual solution $\hat{y}$, the reduced cost of primal variable $x_i$ can be computed as the slack of the corresponding dual constraint: $rc^{\hat{y}}(x_i) = c_i - A_i^T \hat{y}$. In a minimization problem, if $\hat{y}$ is optimal, the reduced cost of a primal variable $x_i$ indicates by how much coefficient $c_i$ would need to decrease before the optimal solution changes. Reduced costs are always non-negative.

The Linear Assignment Problem (LAP) is the following LP:

$$\min \sum_{i,j \in [1,n]} c_{ij} x_{ij} \quad s.t. \tag{1}$$

$$\sum_{j \in [1,n]} x_{ij} = 1 \qquad \forall i \in [1,n] \tag{2}$$

$$\sum_{i \in [1,n]} x_{ij} = 1 \qquad \forall j \in [1,n] \tag{3}$$

where $n$ is an integer and $c_{ij}$ are given costs. The LAP can be seen as a graph problem. Consider a complete bipartite graph $G = V \cup U, E$, with $E = V \times U$. A perfect matching of $G$ is a set of edges $M \subseteq E$ such that exactly one edge in $M$ is adjacent to any vertex in $V$ and to any vertex in $U$. Given a function $c : E \to \mathbb{R}_{\geq 0}$, the cost of a perfect matching $M$ is $c(M) = \sum_{e \in M} c(e)$. LAP is equivalent to finding a perfect matching of minimum cost, where an edge $(i, j)$ is in the matching if and only if $x_{ij} = 1$.

When the sets $V$ and $U$ have different sizes $n < m$, the corresponding problem is called the rectangular LAP (RLAP) (Bijsterbosch and Volgenant 2010):

$$\min \sum_{\substack{i \in [1,n] \\ j \in [1,m]}} c_{ij} x_{ij} \tag{4}$$

$$s.t.$$

$$\sum_{j \in [1,m]} x_{ij} = 1 \qquad \forall i \in [1,n] \tag{5}$$

$$\sum_{i \in [1,n]} x_{ij} \leq 1 \qquad \forall j \in [1,m] \tag{6}$$

The difference here is that we do not require all vertices of the larger set to be used in an edge of the matching, hence constraints (6) are inequalities.

For both LAP and RLAP, there exist efficient, practical, well studied algorithms. LAPJV (Jonker and Volgenant 1987), a variant of the Hungarian algorithm (Kuhn 1955), is widely regarded as the most efficient exact algorithm for the LAP (Crouse 2016) and can be easily adapted for the RLAP.

## ALLDIFFERENT Constraint

The ALLDIFFERENT constraint over a set of variables $S$ requires that no value is assigned to more than one variable. One interpretation of it is that it admits maximal matchings in a bipartite graph where the set $V$ corresponds to variables, $U$ corresponds to values, and there exists an edge $(i, j)$ if $j \in D^i$. When the number of values is equal to the number of variables, ALLDIFFERENT corresponds to a LAP with a constant objective function. When there are more values than variables, it is an RLAP with a constant objective. The *weighted* ALLDIFFERENT constraint has costs associated with each variable-value. It corresponds to LAP or RLAP with variable-value costs placed on the edges.

# Soft Consistencies for ALLDIFFERENT

We show here how to enforce $F\emptyset IC$ for an ALLDIFFERENT constraint in WCSP. In order to do so, we first have to model ALLDIFFERENT as a cost function. Suppose that the scope of the ALLDIFFERENT constraint is $S$. The initial cost of any assignment $I \in \ell(S)$ is:

$$c_S(I) = \begin{cases} 0 & \text{if } I[\{i\}] \neq I[\{j\}] \, \forall i \neq j, i, j \in S \\ \top & \text{otherwise} \end{cases}$$

Enforcing $F\emptyset IC$ requires being able to minimize the full function $C_S^f = c_S + \sum_{i \in S} c_i$. In the general case, once we compute the minimum of that function, we extend the necessary cost from each unary cost function to $c_S$, so that afterwards $\min c_S = \min C_S^f$. Since $c_S$ has a non-zero minimum, we can project that cost to $c_\varnothing$ and get an increase in the lower bound. Note that we do not require the stronger $c_S = C_S^f$. It is also distinct (neither implies the other) from Strong $\varnothing IC$ (Lee and Leung 2009), which requires that all values have a support with finite cost in the conjunction of $c_S$ and all the unary cost functions.

The difficulty comes from the fact that the EXTEND and PROJECT-C0 operations may change $c_S$ so that it is no longer a hard constraint. There is no guarantee that tuples which previously had cost 0 (satisfied the constraint) continue to have cost 0 after the sequence of EXTEND and PROJECT-C0 operations. How we address this depends on whether the constraint is a permutation constraint (as many values as variables) or not (more values than variables).

When $c_S$ is a permutation constraint the problem $\min C_S^f$ corresponds exactly to solving an LAP problem, because it matches the form of a weighted ALLDIFFERENT, which we showed above is a LAP. We write $LAP(S, C)$ for the LAP corresponding to the function $C_S^f$ for an ALLDIFFERENT constraint with scope $S$ and unary cost functions $C = \{c_i \mid i \in S\}$.

An interesting feature of the LAP is that it is an LP that contains only equality constraints. Therefore, the following textbook theorem holds. We reproduce its proof from the thesis of Montalbano (2023) for illustrative purposes.

**Theorem 1.** *Let $P$ be an LP of the form $\min c^T x : Ax = b, x \geq 0$. Let $\hat{y}$ be a feasible dual solution with cost $b^T\hat{y}$ and let $rc^{\hat{y}}(x_i)$ denote the reduced cost of primal variable $x_i$ given the dual solution $y$. Then, the following LP $P'$ is equivalent to $P$: it has the same set of feasible solutions, and all feasible solutions have the same cost:*

$$P' = \begin{cases} & \min b^T\hat{y} + \sum_{i\in[1,n]} rc^{\hat{y}}(x_i)x_i \\ s.t. & \\ & Ax = b \\ & x \in \mathbb{R}^n_{\geq 0} \end{cases}$$

*Proof.* Since the constraints of both problems are identical, we only need to show that the objectives are identical, subject only to $Ax = b$:

$$b^T\hat{y} + (rc^{\hat{y}})^T x = b^T\hat{y} + (c - A^T\hat{y})^T x = c^T x + b^T\hat{y} - \hat{y}^T Ax \tag{7}$$

Finally, since $Ax = b$, we have $\hat{y}^T Ax = \hat{y}^T b = b^T\hat{y}$, so $b^T\hat{y} + (rc^{\hat{y}})^T x = c^T x$, as required. □

This implies that we can use the reduced costs of the LAP to reformulate it. Because the LAP is equivalent to ALLDIFFERENT and its objective function is linear, we can move costs from the linear objective of $c_S$ back to the corresponding unary cost functions. After these cost moves, $c_S$ is again a hard constraint, *i.e.*, all its tuples have either cost 0 or $\top$. Therefore, the algorithm for enforcing $F\emptyset IC$ on permutation constraints is as shown in Algorithm 2.

This algorithm moves maximal cost from the unary cost functions to the permutation constraint. With these costs, minimizing $c_S$ means solving the corresponding LAP. It projects the optimum of the LAP to $c_\varnothing$, then it uses reduced costs from the dual solution to compute new unary costs and projects those back to the unary cost functions. We combine the EXTEND and PROJECT operations into a single operation in line 3 that extends only the necessary $c_i(j) - rc^{y^*}(x_{ij})$. Note that because of Theorem 1 and the discussion after it, the combined EXTEND and PROJECT-C0 operations leave the ALLDIFFERENT constraint unchanged, so each of these operations is constant time (memoizing changes in unary costs only, as (Cooper et al. 2010), page 27). Its complexity is dominated by solving the LAP, which is in $O(n^3)$.

Algorithm 2: PERMUTATION-$F\emptyset IC(S)$: Enforcing $F\emptyset IC$ on a permutation constraint, *i.e.*, an ALLDIFFERENT constraint with as many values as variables.

1: $(opt, x^*, y^*) =$ SOLVE-LAP$(S, \{c_i \mid i \in S\})$
2: **for all** $i \in S, j \in D^i$ **do**
3: EXTEND$(\{i\}, S, j, c_i(j) - rc^{y^*}(x_{ij}))$
4: **end for**
5: PROJECT-C0$(S, opt)$

**Example 1.** *Consider a CFN with 3 variables with 3 values each, an* ALLDIFFERENT *constraint over them, i.e., $S = \{1, 2, 3\}$, and additionally with unary cost functions $c_1(1) = 3, c_1(3) = 2, c_2(1) = 2, c_2(3) = 5, c_3(1) = 1$, and binary cost function $c_{1,3}(2, 1) = 1, c_{1,3}(2, 3) = 1$ (we omit unary and binary tuples that have cost 0). When we enforce $F\emptyset IC$ in the* ALLDIFFERENT *constraint, we get the primal solution $\{X_1 \leftarrow 2, X_2 \leftarrow 1, X_3 \leftarrow 3\}$ with cost 2, which we project to $c_\varnothing$. The reduced costs, which we use as the new unary costs, are $c_1(1) = 1, c_1(3) = 1, c_2(3) = 4, c_3(2) = 1$, and the rest are 0. We then perform* EXTEND$(\{3\}, \{1, 3\}, 2, 1)$ *followed by* PROJECT$(\{1\}, \{1, 3\}, 2, 1)$. *This leaves all values of $X_1$ with a unary cost of at least 1, so we can perform* UNARYPROJECT$(\{1\}, 1)$, *to further increase $c_\varnothing$ to 3. This demonstrates the strength of the communication of different cost functions through marginal costs, as it takes the interaction of unary costs, the* ALLDIFFERENT *constraint, and the binary cost function to increase the lower bound.*

Consider now the case where $c_S$ is not a permutation constraint. Let $U = \cup_{i\in S} D^i$. One possible approach to propagating it is to add $|U| - |S|$ dummy variables with domain $U$, and propagate it as a permutation constraint. However, there exist applications where ALLDIFFERENT constraints have many more values than variables. Adding dummy variables makes for a larger graph, and also exposes these extra variables to the WCSP solver, making for a greater overhead.

We show next that it is possible to avoid adding extra variables. Minimizing $C_S^f$ in this case requires solving an RLAP, the variant of RLAP where the partitions of the graph have different sizes. Similarly to LAP, this LP can be solved efficiently, using variants of LAP algorithms. However, it does not satisfy the conditions of Theorem 1, so Algorithm 2 is not correct for this case, as the following example shows.

**Example 2.** *Consider an* ALLDIFFERENT *constraint with 3 variables $\{X_1, X_2, X_3\}$ and 4 values $\{1, 2, 3, 4\}$ and suppose $c_i(1) = c_i(2) = c_i(3) = 5, c_i(4) = 1$ for all $i \in [1, 2, 3]$. The RLAP has optimum 11, by using the value 4 for one variable and any of the values 1,2,3 for the other two variables. All reduced costs are 0, so if we use them to reformulate the problem, i.e., if we set them as the*

Algorithm 3: ALLDIFFERENT-$F\emptyset IC(S)$: Enforcing $F\emptyset IC$ on a general ALLDIFFERENT constraint, *i.e.*, an ALLDIFFERENT constraint with more values than variables.

```
(opt, x*, y*) = SOLVE-RLAP(S, {c_i | i ∈ S})
for all i ∈ S, j ∈ D^i do
    Δ_i(j) ← Δ_i(j) + c_i(j) − rc^{y*}(x_ij)
    EXTEND({i}, S, j, c_i(j) − rc^{y*}(x_ij))
end for
PROJECT-C0(S, opt)
Δ_∅ ← Δ_∅ + opt
```

*new unary costs of all variable/value pairs, the assignment $X_1 = 1, X_2 = 2, X_3 = 3$ will have cost 11, while its cost in the original CFN is 15. Therefore, reduced costs do not give a reformulation in the non-permutation case.*

Note, however, that it remains safe to project reduced costs onto unary cost functions. The reason for this is that reduced costs are always an underestimation of the true marginal cost, and are additive, *i.e.*, assigning two variables with non-zero reduced cost increases the lower bound by at least the sum of the two reduced costs. The only thing that we lose by going from LAP to RLAP is that the reduced costs no longer provide a reformulation.

For correctness, a propagator for a non-permutation ALLDIFFERENT has to compensate for the costs that have been moved into the constraint and which correspond to non-unary assignments. We do this in our algorithm by introducing *delta* costs, which give for each unary cost $c_i(j), i \in S, j \in D^i$, the balance of cost that has been moved from $c_i$ to $c_S$ and from $c_S$ to $c_\emptyset$. Using these costs, it is possible to reconstruct the correct cost for each assignment.

Specifically, the ALLDIFFERENT constraint has a state associated with it, which is the vector $(\Delta_1, \ldots, \Delta_n, \Delta_\emptyset)$, where each $\Delta_i$ and $\Delta_\emptyset$ are themselves cost functions.

The propagator maintains the invariant that

$$c_S(I) = \begin{cases} \left(\sum_{i \in S} \Delta_i(I[\{i\}])\right) - \Delta_\emptyset & \text{If } I \models \text{ALLDIFFERENT} \\ \top & \text{otherwise} \end{cases}$$

By updating the $\Delta_i$ and $\Delta_\emptyset$ values as costs are moved from unary cost functions, the propagator maintains this invariant, therefore no cost is lost. In Example 2, the entirety of $c_1, c_2, c_3$ will be moved to $\Delta_1, \Delta_2, \Delta_3$, so that when an assignment is evaluated, it reports the correct cost.

## Additional Considerations

**Pruning Values.** We can prune values that appear in no matching using standard techniques (Régin 1994). Following Focacci, Lodi, and Milano (1999), we also perform reduced cost-based pruning. Once we project the reduced costs to unary cost functions, NC automatically performs cost-based variable fixing. It prunes $j$ from $D^i$ if $c_\emptyset + c_i(j) \geq ub$. This is more powerful than reduced cost-based fixing in a standalone propagator for weighted ALLDIFFERENT: in that case, we prune $j$ from $D^i$ if $opt + rc_i^{y^*}(x_{ij}) \geq ub$, but $c_\emptyset \geq opt$, so this condition prunes less often. In Example 1, if there exists an incumbent with cost 7, we can prune 3 from $D^2$, because $c_\emptyset + c_2(3) = 3 + 4 \geq ub = 7$. We cannot do the same if we use the optimum of the ALLDIFFERENT constraint instead of $c_\emptyset$ because $opt + c_2(3) = 2 + 4 < ub = 7$.

**Incrementality.** In general, incrementality for this constraint is difficult, as the LAP we solve at each invocation does not change monotonically. There is however a very simple technique that eliminates some calls to the LAP solver: at each invocation, we store the primal and dual solutions. At the next invocation, if those are still feasible, there is no need to solve the LAP again. We have empirically found that this improves performance and use it by default.

# Experimental Evaluation

We implemented our approach in toulbar2, an open-source C++ exact WCSP solver. The enforced soft consistencies are EDAC (De Givry et al. 2005) for table cost functions, partial $F\emptyset IC$ for linear (Montalbano, de Givry, and Katsirelos 2022)) and ALLDIFFERENT constraints, applied at every search node of a hybrid best/depth-first branch-and-bound search method (Allouche et al. 2015). We compared different encodings of ALLDIFFERENT in toulbar2: decomposed in $\frac{n(n-1)}{2}$ binary constraints (denoted *tb2+binary*), decomposed in $m$ at-most-one constraints (denoted *tb2+KP*), using the Soft-ALLDIFFERENT flow-based implementation (Lee and Leung 2009) (denoted *tb2+salldiff*), and our implementation with LAPJV (denoted *tb2+*LAPJV). When the LAP defines a permutation ($n = m$), we can add redundant at-least-one constraints. We compared the various toulbar2 modeling approaches against Google OR-Tools CP-SAT, an open-source state-of-the-art constraint programming solver, and IBM cplex, a state-of-the-art integer programming solver.[3]

toulbar2 used the *dom/wdeg* variable ordering heuristic (Boussemart et al. 2004) with last conflict (Lecoutre et al. 2009) and EAC *support value* ordering heuristic (Cooper et al. 2010; Trösser, De Givry, and Katsirelos 2020), combined with solution phase saving (Demirović, Chu, and Stuckey 2018). Unless reported otherwise, all tests were run on a 2.5GHz Intel Xeon E5-2680, using one thread per CPU. We limit CPU-time to 1200 seconds, except for the XCSP benchmark with 1800 seconds, to match the XCSP competition. For each benchmark we compute a score with the system used in the XCSP competition. This scoring system gives 1 point per instance solved or if no other solver found a better solution, and $\frac{1}{2}$ point if the solver found an optimal solution without proving optimality but another solver proved its optimality.

## Weighted $N$-Queens Problem

The Weighted $N$-Queens problem is a variant of the $N$-Queens problem (Bezzel 1848) where each queen placement incurs a cost. The objective is to place $N$ queens on

[3] https://github.com/toulbar2/toulbar2 v1.2.1, binary branching (option *-d:*), https://github.com/google/or-tools CP-SAT v9.14 in free-search mode, cplex version 22.1.1.0 with non-premature stop parameters *EPAGAP=EPGAP=EPINT=0*. All single-threaded.

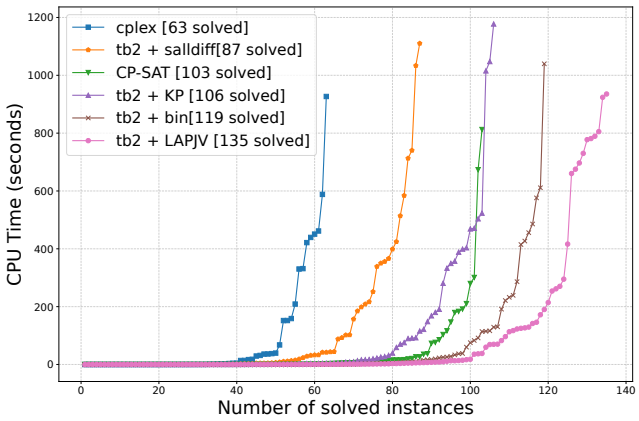


Figure 1: Weighted $N$-Queens cactus plot.

| Solver $N$ | IBM cplex | tb2 salldiff | OR-tools CP-SAT | tb2 KP | tb2 binary | tb2 LAPJV |
|---|---|---|---|---|---|---|
| 14 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 16 | 5.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 18 | 17.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 20 | 59.8 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22 | 103.2 | 13.7 | 0.0 | 0.0 | 0.0 | 0.0 |
| 24 | 113.2 | 10.9 | 4.0 | 0.6 | 0.0 | 0.0 |
| 26 | 145.3 | 35.3 | 8.4 | 4.6 | 0.1 | 0.0 |
| 28 | 173.1 | 32.2 | 10.0 | 6.8 | 1.9 | 0.0 |
| 30 | 225.8 | 39.4 | 16.3 | 9.6 | 1.5 | 0.7 |
| Score | 63.5 | 90.0 | 105.5 | 109.5 | 126.5 | 138.0 |

Table 1: Weighted $N$-Queens average optimality gaps (%) and XCSP scores (bottom line).

| *Solver* | cplex | CP-SAT | *tb2+at-least-one* | *tb2*+LAPJV |
|---|---|---|---|---|
| *Score* | 25 | 43.5 | 57.5 | **100** |
| *gap* | 16% (107) | 5.4%(121) | 5.3 % (131) | **3.9 %(131)** |

Table 2: Scores and average gap to best known solution (# of instances where a solution was returned) for QAP.

an $N \times N$ board, minimizing the total cost while ensuring that no two queens threaten each other. This is an interesting benchmark in that it isolates the performance of ALLDIFFERENT. In contrast to the unweighted version, it is a challenging problem for the solvers we evaluate. We tested 140 instances, with sizes ranging from $N = 4$ to $N = 30$ in steps of 2, with 10 instances per size, using random costs uniformly sampled in $[1, N]$. We modeled the problem using three ALLDIFFERENT constraints to enforce column, diagonal and anti-diagonal uniqueness. Costs are defined by unary cost functions. The same model was implemented in toulbar2, CP-SAT, and cplex. [4]

Figure 1 and Table 1 show that *tb2*+LAPJV significantly outperforms the other solvers in terms of execution time and scalability, finding the best-known solution in 138 out of 140 instances and proving optimality in 135 cases. Both *tb2*+LAPJV and *tb2+salldiff* handle ALLDIFFERENT without decomposition. The difference in performance between these two methods shows that enforcing $F\emptyset IC$ achieves a favorable balance between propagation strength and computational speed. Conversely, *tb2+KP* and *tb2+bin* decompose ALLDIFFERENT into multiple cost functions, resulting in weaker propagation. Still, *tb2+bin* is the second best approach, solving 106 instances. While it is competitive on the smallest instances thanks to a very fast propagation, it fails on the largest instances $N = 28$ (see Table 1). cplex reaches its limits after $N = 16$ and CP-SAT after $N = 24$.

## Quadratic Assignment Problems

The quadratic assignment problem (QAP) is a generalization of the LAP which also places costs on combinations of edges. It is a very challenging problem and even MIP solvers struggle to scale beyond the smallest instances. It is naturally modeled as a WCSP with a single permutation constraint and unary and binary cost functions to capture the costs of edges and pairs of edges, respectively.

We took the 132 smallest of the 136 instances from the QAPLIB.[5] For a problem of size $N$, we expressed the quadratic objective function as a binary Weighted CSP with $N$ variables of domain size $N$. The permutation constraint is encoded for toulbar2 and cplex as binary constraints enforcing that any pair of two variables cannot take the same value. To strengthen the propagation we add one ALLDIFFERENT constraint of arity $N$ (*tb2*+LAPJV) or $N$ redundant (generalized) linear constraints of arity $N$ to ensure that each value is assigned to at least one variable (*tb2+at-least-one*). It is encoded as an ALLDIFFERENT constraint in CP-SAT.

Within the CPU-time limit of $1,200$ seconds, *tb2+at-least-one* solved 31 instances and *tb2*+LAPJV solved 33 (see Figure 2). Although it has a theoretically stronger lower bound, CPLEX solved only 22 instances, and CP-SAT 25. For instance, *tb2*+LAPJV solved *scr20* in 337 seconds and 0.9 million search nodes. Other models in CP-SAT, toulbar2, and cplex timed out. The best (XYL) $0/1$LP approach reported in (Zhang, Beltran-Royo, and Ma 2013) solved it in 6,369s and 1.4 million nodes using an Intel Core Duo 2.80 GHz and Cplex 11.2. Considering feasibility only, *tb2*+LAPJV and *tb2+at-least-one* were able to find feasible solutions for 131 cases, whereas cplex and CP-SAT obtained solutions for only 107 and 121 instances, respectively (see Table 2). Among all methods, *tb2*+LAPJV returns the highest quality solutions with a gap of 3.9% to best known solutions, leading to a significantly higher score than those of the other approaches. The ALLDIFFERENT constraints encoded in the QAP instances are permutation constraints, allowing *tb2*+LAPJV to use the efficient Algorithm 2 without relying on the $\Delta$ costs. In contrast, the decomposition-based method *tb2+at-least-one* depends on $\Delta$ costs and is sensitive to the propagation order, yielding weaker propagation.

[4] Scripts at https://miat.inrae.fr/degivry/aaai2026supp.tgz

[5] http://coral.ise.lehigh.edu/wp-content/uploads/2014/07/qapdata.tar.gz, sizes less than 100 variables.

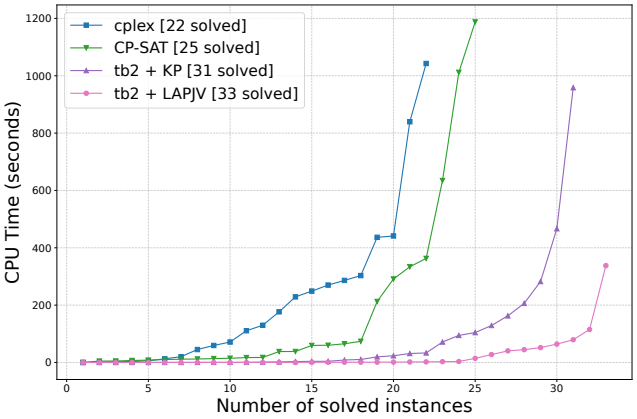


Figure 2: Quadratic Assignment Problems cactus plot.

## XCSP 2024 Mini COP Competition

We restricted to the Mini COP category of the 2024 XCSP competition and selected the 40 instances that contain one or more ALLDIFFERENT constraints.[6]

ALLDIFFERENT constraints are either decomposed into binary constraints if the scope size is less than 100, else translated into at-most-one linear constraints, one for each domain value (*tb2+KP*), or kept as a global constraint using our approach (*tb2*+LAPJV). Element constraints are decomposed into binary and ternary constraints. Sum constraints are directly converted into knapsack constraints. Constraints in extension (table) or intention (functional expression) are expressed using a dual encoding with one extra domain variable representing the allowed tuples of the original constraint and binary constraints to link this extra variable with the variables in the scope of the original constraint (Montalbano et al. 2023). The objective variable is transformed into a unary cost function the cost of which corresponds to the domain value. An objective given by a sum on $n$ variables is transformed into $n$ unary cost functions.

In 2024, 11 instances among 40 were unsolved. The largest solved instance is *Charlotte-mini-24-2* with 507 variables and $1,240$ constraints, including 121 ALLDIFFERENT. Both CP-SAT and *tb2*+LAPJV solved the greatest number of instances, *i.e.*, 22 instances. CP-SAT was the fastest in general (Fig. 3). However, *tb2*+LAPJV proved unsatisfiability for two instances that were unsolved by all the 15 competitors of the 2024 XCSP competition, including CP-SAT (*RotationPuzzle-5* and *RotationPuzzle-7*). It also improved significantly compared to *tb2+KP*, which solved 17 instances. cplex solved 15 instances and gurobi got the worst results with 6 solved instances.

Taking into account the best solutions found during this competition, we re-evaluated the score of every participant. *tb2*+LAPJV obtained the highest score for this selection of instances (previous best score was obtained by CP-SAT). A selection of the scores are presented in Table 3. *tb2*+LAPJV found 4 best solutions without proving optimality (*GolombRuler-a3v18-11, GolombRuler-a3v18-nodv-11* being solved by CP-SAT, *Pyramid-09-1300* by the `choco` solver and others, and *GolombRuler-a3v18-13* was unsolved), hence its score of $22 + 3 \times \frac{1}{2} + 1 = 24.5$.

[6]https://xcsp.org/competitions

| *Solver* | gurobi | cplex | tb2+KP | CP-SAT | tb2+LAPJV |
|---|---|---|---|---|---|
| *Score* | 6 | 15 | 17 | 23.5 | **24.5** |

Table 3: Re-evaluation of XCSP 2024 score per solver on a selection of 40 Mini COP instances with assignment problems.

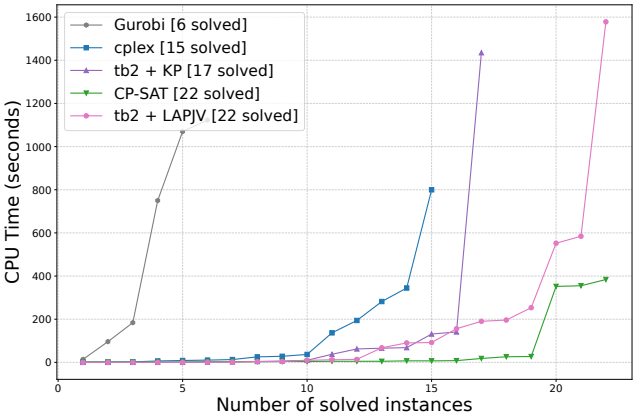


Figure 3: XCSP24 MiniCOP ALLDIFFERENT cactus plot.

## Conclusion

We have presented algorithms for propagating variants of the ALLDIFFERENT constraint in weighted CSPs. The addition of a soft consistency propagator for ALLDIFFERENT in the state-of-the-art WCSP solver toulbar2 yielded performance that surpassed that of other CP solvers like OR-Tools and of the MIP solver CPLEX in several families of instances.

Within the scope of ALLDIFFERENT, we plan to investigate techniques used for generalized arc consistency (GAC) on weighted ALLDIFFERENT (Sellmann 2002; Claus, Cambazard, and Jost 2020). Additionally, we intend to explore enforcing VAC (Cooper et al. 2010) on WCSPs with ALLDIFFERENT constraints. VAC is more expensive to enforce but gives stronger bounds and is useful when applied in a limited manner, such as during preprocessing.

More generally, the technique we used for propagating ALLDIFFERENT relied only on the reduced costs of a linear formulation of the problem. This gave a relatively straightforward way to capture reformulations. The simplicity of the use of the LP formulations means that we can apply those techniques to propagating other global constraints with ideal linear formulations (German et al. 2017), including generalizations of ALLDIFFERENT like the global cardinality constraint (GCC). It will be more challenging to come up with reformulation methods for constraints where the reasoning of propagation algorithms cannot be captured by a compact linear program, as is the case for the cumulative constraint.

## References


Allouche, D.; Bessière, C.; Boizumault, P.; De Givry, S.; Gutierrez, P.; Loudni, S.; Metivier, J.-P.; and Schiex, T. 2012. Filtering decomposable global cost functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 407–413.

Allouche, D.; de Givry, S.; Katsirelos, G.; Schiex, T.; and Zytnicki, M. 2015. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In *Proc. of CP-15*, 12–28. Cork, Ireland.

Bensana, E.; Lemaître, M.; and Verfaillie, G. 1999. Earth Observation Satellite Management. *Constraints An Int. J.*, 4(3): 293–299.

Bezzel, M. 1848. Proposal of 8-queens problem. *Berliner Schachzeitung*, 3(363): 1848.

Bijsterbosch, J.; and Volgenant, A. 2010. Solving the Rectangular assignment problem and applications. *Ann. Oper. Res.*, 181(1): 443–462.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, 146.

Claus, G.; Cambazard, H.; and Jost, V. 2020. Analysis of Reduced Costs Filtering for Alldifferent and Minimum Weight Alldifferent Global Constraints. In Giacomo, G. D.; Catalá, A.; Dilkina, B.; Milano, M.; Barro, S.; Bugarín, A.; and Lang, J., eds., *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325, 323–330. Santiago de Compostela, Spain: IOS Press.

Cooper, M.; de Givry, S.; Sanchez, M.; Schiex, T.; Zytnicki, M.; and Werner, T. 2010. Soft arc consistency revisited. *Artificial Intelligence Journal*, 174: 449–478.

Cooper, M. C.; de Givry, S.; Sánchez-Fibla, M.; Schiex, T.; and Zytnicki, M. 2008. Virtual Arc Consistency for Weighted CSP. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 253–258. AAAI Press.

Cooper, M. C.; de Givry, S.; and Schiex, T. 2020. Graphical Models: Queries, Complexity, Algorithms (Tutorial). In Paul, C.; and Bläser, M., eds., *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, 4:1–4:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Crouse, D. F. 2016. On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4): 1679–1696.

De Givry, S.; Heras, F.; Zytnicki, M.; and Larrosa, J. 2005. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *IJCAI*, volume 5, 84–89.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1): 41–85.

Demirović, E.; Chu, G.; and Stuckey, P. J. 2018. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In *International Conference on Principles and Practice of Constraint Programming*, 99–108. Springer.

Dlask, T.; and Savchynskyy, B. 2025. Relative-interior solution for the (incomplete) linear assignment problem with applications to the quadratic assignment problem: T. Dlask, B. Savchynskyy. *Annals of Mathematics and Artificial Intelligence*, 1–44.

Focacci, F.; Lodi, A.; and Milano, M. 1999. Cost-Based Domain Filtering. In Jaffar, J., ed., *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, 189–203. Springer.

German, G.; Briant, O.; Cambazard, H.; and Jost, V. 2017. Arc Consistency via Linear Programming. In Beck, J. C., ed., *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, 114–128. Springer.

Haller, S.; Feineis, L.; Hutschenreiter, L.; Bernard, F.; Rother, C.; Kainmüller, D.; Swoboda, P.; and Savchynskyy, B. 2022. A comparative study of graph matching algorithms in computer vision. In *European Conference on Computer Vision*, 636–653. Springer.

Hurley, B.; O'Sullivan, B.; Allouche, D.; Katsirelos, G.; Schiex, T.; Zytnicki, M.; and de Givry, S. 2016. Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints An Int. J.*, 21(3): 413–434.

Jonker, R.; and Volgenant, A. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38: 325–340.

Kolmogorov, V. 2006. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10): 1568–1583.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2): 83–97.

Larrosa, J. 2002. On Arc and Node Consistency in weighted CSP. In *Proc. AAAI'02*, 48–53. Edmondton, (CA).

Lecoutre, C.; Saïs, L.; Tabary, S.; and Vidal, V. 2009. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence Journal*, 173: 1592,1614.

Lee, J. H.-M.; and Leung, K. L. 2009. Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction. In *IJCAI*, volume 9, 559–565.

Montalbano, P. 2023. *Linear Constraints and Conflict-free Learning for Graphical Models*. Ph.D. thesis, Université de Toulouse.

Montalbano, P.; Allouche, D.; De Givry, S.; Katsirelos, G.; and Werner, T. 2023. Virtual Pairwise Consistency in Cost Function Networks. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 417–426. Springer.

Montalbano, P.; de Givry, S.; and Katsirelos, G. 2022. Multiple-choice knapsack constraint in graphical models. In

*International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 282–299. Springer.

Nguyen, H.; de Givry, S.; Schiex, T.; and Bessiere, C. 2014. Maintaining Virtual Arc Consistency Dynamically during Search. In *$26^{th}$ IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, 8–15. IEEE Computer Society.

Régin, J. 1994. A Filtering Algorithm for Constraints of Difference in CSPs. In Hayes-Roth, B.; and Korf, R. E., eds., *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, 362–367. AAAI Press / The MIT Press.

Régin, J. 1999. Arc Consistency for Global Cardinality Constraints with Costs. In Jaffar, J., ed., *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, 390–404. Springer.

Sanchez, M.; de Givry, S.; and Schiex, T. 2008. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1): 130–154.

Schiex, T. 2000. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *LNCS*, 411–424. Singapore.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, 631–639. Morgan Kaufmann.

Sellmann, M. 2002. An Arc-Consistency Algorithm for the Minimum Weight All Different Constraint. In Hentenryck, P. V., ed., *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, 744–749. Springer.

Simoncini, D.; Allouche, D.; de Givry, S.; Delmas, C.; Barbe, S.; and Schiex, T. 2015. Guaranteed Discrete Energy Optimization on Large Protein Design Problems. *Journal of Chemical Theory and Computation*, 11(12): 5980–5989. PMID: 26610100.

Trösser, F.; De Givry, S.; and Katsirelos, G. 2020. Relaxation-aware heuristics for exact optimization in graphical models. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 475–491. Springer.

Vaidya, P. M. 1989. Speeding-up linear programming using fast matrix multiplication. In *30th annual symposium on foundations of computer science*, 332–337. IEEE Computer Society.

Werner, T.; Průša, D.; and Dlask, T. 2020. Relative Interior Rule in Block-Coordinate Descent. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7556–7564.

Zanfir, A.; and Sminchisescu, C. 2018. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2684–2693.

Zhang, H.; Beltran-Royo, C.; and Ma, L. 2013. Solving the quadratic assignment problem by means of general purpose mixed integer linear programming solvers. *Annals of Operations Research*, 207: 261–278.

Zytnicki, M.; Gaspin, C.; de Givry, S.; and Schiex, T. 2009. Bounds Arc Consistency for Weighted CSPs. *Journal of Artificial Intelligence Research*, 35: 593–621.