An $O(n \log n)$ Bound Consistency Algorithm for the Conjunction of an $alldifferent$ and an *Inequality between a Sum of Variables and a Constant*, and its Generalization

Nicolas Beldiceanu[1] and **Mats Carlsson**[2] and **Thierry Petit**[3] and **Jean-Charles Régin**[4]

Abstract. This paper gives an $O(n \log n)$ bound-consistency filtering algorithm for the conjunction $alldifferent(V_0, V_1, \ldots, V_{n-1}) \wedge f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq cst$, $(V_0, V_1, \ldots, V_{n-1}, cst \in \mathbb{N}^+)$, where $(\mathbb{N}, \oplus)$ is a commutative group, f is a unary function, and both $\oplus$ and f are monotone increasing. This complexity is equal to the complexity of the bound-consistency algorithm of the $alldifferent$ constraint.

1 Introduction

Since the early days of constraint programming it has been part of the folklore to try to capture the interaction of two constraints in order to perform more deduction. This was for instance done in [1] for a linear constraint for which all variables should be assigned distinct values.[5] In this context, a better evaluation of the minimum and maximum values of a linear term was suggested, since assigning all variables to their minimum (resp. maximum) value leads to a poor bound which totally ignores the $alldifferent$ constraint. More recently, it has been quoted that such patterns can be directly captured by a global constraint such as $global\_cardinality\_with\_costs$ [2] (see the Usage slot of this constraint in [3]). However using $global\_cardinality\_with\_costs$ for this purpose is not memory and time effective (i.e., we need to introduce a cost matrix and the worst case time complexity of the algorithm is $O(n(m + n \log n))$ where n is number of variables, and m the sum of domain sizes). Motivated by these facts, this paper provides a generic bound-consistency (i.e., a filtering algorithm ensures *bound-consistency* for a given constraint $\mathcal{C}$ if and only if for every variable V of $\mathcal{C}$ there exists at least one solution for $\mathcal{C}$ such that V can be assigned to its minimum value $\underline{V}$ (resp. maximum value $\overline{V}$) and every other variable U of $\mathcal{C}$ is assigned to a value in $[\underline{U}, \overline{U}]$ [4]) filtering algorithm for the conjunction $alldifferent(V_0, V_1, \ldots, V_{n-1}) \wedge f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq cst$ $(V_0, V_1, \ldots, V_{n-1}, cst \in \mathbb{N}^+)$, where:

- V_i $(0 \leq i < n)$ is a variable taking its value in a given fixed interval $[\underline{V_i}, \overline{V_i}]$.
- $alldifferent(V_0, V_1, \ldots, V_{n-1})$ is a constraint enforcing variables $V_0, V_1, \ldots, V_{n-1}$ to be assigned distinct integer values.
- $(\mathbb{N}, \oplus)$ is a commutative group, where in addition $\oplus$ is monotone increasing, $(\forall\, p, q_1, q_2 \in \mathbb{N}^+ : q_1 > q_2 \Rightarrow p \oplus q_1 > p \oplus q_2)$. $\oplus^{-1}$ denotes the inverse operation.
- f is a monotone increasing function ($q_1 > q_2 \Rightarrow f(q_1) > f(q_2)$).

Example 1 *As an illustrative example, consider ten variables* $V_0, V_1, \ldots, V_9$ *which respectively take integer values in intervals* $[1,8]$, $[2,5]$, $[3,4]$, $[3,4]$, $[2,5]$, $[1,16]$, $[7,12]$, $[7,16]$, $[9,16]$, $[12,16]$. *Assume that, given these ten variables, we have the following conjunction of constraints:*

1. $alldifferent(V_0, V_1, \ldots, V_9) \wedge \sum_{i=0}^{9} V_i^2 \leq 500$,
2. $alldifferent(V_0, V_1, \ldots, V_9) \wedge \prod_{i=0}^{9} V_i \leq 4717500$.

For each of these conjunctions, a bound-consistency algorithm would respectively narrow[6] *the domains to:*

1. $V_0 \in [1,8]$, $V_1 \in [2,5]$, $V_2 \in [3,4]$, $V_3 \in [3,4]$, $V_4 \in [2,5]$, $V_5 \in [1,\mathbf{10}]$, $V_6 \in [7,\mathbf{11}]$, $V_7 \in [7,\mathbf{11}]$, $V_8 \in [9,\mathbf{11}]$, $V_9 \in [12,\mathbf{14}]$.
2. $V_0 \in [1,\mathbf{6}]$, $V_1 \in [2,5]$, $V_2 \in [3,4]$, $V_3 \in [3,4]$, $V_4 \in [2,5]$, $V_5 \in [1,\mathbf{10}]$, $V_6 \in [7,\mathbf{8}]$, $V_7 \in [7,\mathbf{8}]$, $V_8 \in [9,\mathbf{9}]$, $V_9 \in [12,\mathbf{13}]$. *The details leading to this pruning will be given in Figure 1.*

The main question addressed by this paper is how such filtering can be done *efficiently* with a *generic* algorithm that is parametrized by a binary operation $\oplus$ and a monotone increasing function f. Section 2 provides an $O(n \log n)$ bound-consistency filtering algorithm for such a pattern, where n is the number of variables. Motivated by the common pattern of combining $alldifferent$ with multiple arithmetic constraints, Section 3 introduces the $alldifferent\_arith$ constraint. It shows how to reuse and enhance the filtering algorithm introduced in Section 2 in order to partially take into account initial holes as well as fixed variables. Finally, Section 4 evaluates $alldifferent\_arith$.

2 Bound Consistency for a Conjunction of an $alldifferent$ and a *linear inequality* Constraints

Assuming each variable has no holes in its domain, this section provides:

1. A priority rule for computing the minimum cost matching for the special case where $\oplus$ is the sum operator and f the identity function.
2. A discussion how the same priority rule can be used when $(\mathbb{N}, \oplus)$ is a commutative group, f is a unary function, and both $\oplus$ and f are monotone increasing.
3. An $O(n \log n)$ algorithm implementing this priority rule.

[1] Mines de Nantes, France, email: Nicolas.Beldiceanu@mines-nantes.fr
[2] SICS, Sweden, email: Mats.Carlsson@sics.se
[3] Mines de Nantes, France, email: Thierry.Petit@mines-nantes.fr
[4] 3S, CNRS, University of Nice-Sophia Antipolis, email: Jean-Charles.Regin@unice.fr

[5] Note that the same problem arises also when the linear constraint is replaced by a more general arithmetic constraint.

[6] Domain reductions are shown in bold.

4. An $O(n \log n)$ filtering algorithm that achieves bound-consistency for a conjunction of an *alldifferent*$(V_0, V_1, \ldots, V_{n-1})$ constraint and an *arithmetic inequality* constraint of the form $\oplus_{i \in \mathcal{I}, \mathcal{I} \subseteq \{0,1,\ldots,n-1\}} f(V_i) \leq cst$.

2.1 Minimum Cost Matching

This section provides a priority rule for computing a variable-value assignment using distinct values and minimizing the sum of all the variables (i.e., a *minimum cost matching*). Values are traversed in increasing order, a value being assigned to the still unassigned variable with the smallest maximum. Groups of consecutive values that do not belong to any variable are skipped.

Priority Rule. We consider the variable-value graph $G = ((\mathcal{V}, \cup_D(\mathcal{V})), E)$ where $\mathcal{V} = \{V_0, \ldots, V_{n-1}\}$ is a set of variables, $\cup_D(\mathcal{V})$ the union of domains of variables in $\mathcal{V}$, and E a set of edges. An edge e belongs to E iff the three following conditions hold: (1) one extremity of e is a variable $V_i \in \mathcal{V}$, and (2) the other extremity of e is a value $v \in \cup_D(\mathcal{V})$, and (3) the value v is in the domain of V_i.

Note that our variable-value graph is convex since each domain consists of one single interval of consecutive values. Taking advantage of convexity usually allows to get a better complexity, for instance for a maximum matching in a bipartite graph; see [5].

Definition 1 (var-perfect matching) *Given a variable-value graph $G = ((\mathcal{V}, \cup_D(\mathcal{V})), E)$, a* var-perfect matching *is a subset M of E of size $|\mathcal{V}| = n$ such that there is no pair of edges in M having an extremity in common.*

Definition 2 *Given a subset $\mathcal{X}$ of variables $\mathcal{V} = \{V_0, \ldots, V_{n-1}\}$ and an integer value $prev$, we define:*

- $cur(\mathcal{X}, prev) = \min_{v \in \cup_D(\mathcal{X}) \wedge v > prev}(v)$*, i.e., the smallest value in $\cup_D(\mathcal{X})$ greater than $prev$.*
- $\mathcal{X}_{cur}(\mathcal{X}, prev) = \{V_i \in \mathcal{X}$ *s.t.* $cur(\mathcal{X}, prev) \in D(V_i)\}$*, i.e., the set of variables in $\mathcal{X}$ having value $cur(\mathcal{X}, prev)$ in their domains.*
- $\mathcal{X}_{cur}^{\max}(\mathcal{X}, prev) = \{V_j \in \mathcal{X}_{cur}(\mathcal{X}, prev)$ *s.t.* $\overline{V_j} = \min_{V_i \in \mathcal{X}_{cur}}(\overline{V_i})\}$*, the subset of variables in $\mathcal{X}_{cur}(\mathcal{X}, prev)$ with the smallest maximum value.*
- $V_{cur}(\mathcal{X}, prev) = V_j$, $V_j \in \mathcal{X}_{cur}^{\max}(\mathcal{X}, prev)$, *s.t.* $j = \min_{V_i \in \mathcal{X}_{cur}^{\max}(\mathcal{X}, prev)}(i)$*, the smallest index of the variables in $\mathcal{X}_{cur}^{\max}(\mathcal{X}, prev)$.*

Notation 1 *Given a subset $M \subseteq E$ and value $v \in \cup_D(\mathcal{V})$, we note $v \dot{\in} M$ iff v is the extremity of at least one edge in M.*

Without loss of generality, the next theorem assumes that there exists a var-perfect matching in G. Its existence can be checked in practice by using the polynomial feasibility condition of *alldifferent*$(V_0, V_1, \ldots, V_{n-1})$.

Theorem 1 *Given a variable-value graph $G = ((\mathcal{V}, \cup_D(\mathcal{V})), E)$ such that there exists at least one var-perfect matching, the minimum value of $\sum_{v_j \dot{\in} M} v_j$ among all maximum matchings M of G can be obtained by the following inductive function h taking three parameters, (i) a set of variables $\mathcal{X}$ initialized to $\mathcal{V} = \{V_0, \ldots, V_{n-1}\}$, (ii) a set of edges M, initially empty, and (iii) a value $prev$ initialized to $\min(\cup_D(\mathcal{V})) - 1$:*

- *If* $\mathcal{X} = \emptyset$ *then* $h(\mathcal{X}, M, prev) = 0$.
- *Otherwise:*

$$h(\mathcal{X}, M, prev) = cur(\mathcal{X}, prev) +$$
$$h(\mathcal{X} \setminus \{V_{cur}(\mathcal{X}, prev)\},\ M \cup \{(V_{cur}(\mathcal{X}, prev), cur(\mathcal{X}, prev))\},$$
$$cur(\mathcal{X}, prev))$$

Proof 1 *We prove by induction that (p1) the current set M is a matching, (p2) M is such that there exists a var-perfect matching M' of G such that $M \subseteq M'$ and M' minimizes $\sum_{v_j \dot{\in} M'} v_j$, (p3) h is the sum of values in M, (p4) $prev$ is the largest value extremity of an edge in M. Initially at step $k = 0$, $M = \emptyset$ and $h = 0$, the four properties p1, p2, p3 and p4 are obviously true. We now assume that the four properties are true for any $|M| = k$ $(k < n)$ and prove that they remain true for $|M| = k + 1$. Before updating the parameters of h, $prev$ equals the largest value extremity of an edge in M. Thus, by Definition 2, $cur(\mathcal{X}, prev)$ is the smallest possible value for a variable in $\mathcal{X}$ greater than values that are extremities of some edges in M: adding $\{(V_{cur}(\mathcal{X}, prev), cur(\mathcal{X}, prev))\}$ to M preserves the fact that M is a matching (so p1 is satisfied) and setting $prev$ to $Cur(X, prev)$ satisfies p4. By construction, adding $cur(\mathcal{X}, prev)$ to h leads to satisfaction of Property (p3). With respect to Property (p2), by Definition 2 we know that $V_{cur}(\mathcal{X}, \underline{prev})$ is the variable minimizing the size of interval $[cur(\mathcal{X}, prev), \overline{(V_i)}]$ among all the variables $V_i \in \mathcal{X}$ such that $cur(\mathcal{X}, prev)$ can be assigned to V_i. Consider the integer $p \geq 0$ such that $cur(\mathcal{X}, prev) = \overline{V_{cur}(\mathcal{X}, prev)} - p$. If $p = 0$, all V_i's in $\mathcal{X}_{cur}(\mathcal{X}, prev)$ except $V_{cur}(\mathcal{X}, prev)$ have a maximum value in their domain greater that $cur(\mathcal{X}, prev)$ since there exists at least one var-perfect matching in G. If $p = 1$ the existence of a var-perfect matching in G guarantees that there is at most one variable $V'_{cur} \neq V_{cur}$ such that $\overline{V'_{cur}} = \overline{V_{cur}}$. In this case, adding V_{cur} or V'_{cur} is equivalent, while adding any other variable would lead to a contradiction with Property (p2) at the next step (in the other case, selecting the variable with the smallest interval does not decrease the number of possible extended matchings). This reasoning can be generalized by recurrence to any p. Thus, selecting V_{cur} guarantees that the number of var-perfect matchings M' of G such that $M \subseteq M'$ and M' minimizes $\sum_{v_j \dot{\in} M'} v_j$ is strictly positive.* □

2.2 Validity of the Priority Rule: General Case

Given:

1. $(\mathbb{N}, \oplus)$ a commutative group where $\oplus$ is monotone increasing,
2. f a monotone increasing function,
3. a set of integer variables $\mathcal{V} = \{V_0, V_1, \ldots, V_{n-1}\}$ subject to *alldifferent*$(V_0, V_1, \ldots, V_{n-1})$,

a matching containing all variables $V_0, V_1, \ldots, V_{n-1}$ minimizing $f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1})$ can be obtained, when it exists, by using the priority rule introduced in Theorem 1. First note that, given a permutation σ of $\{0, 1, \ldots, n-1\}$, since $(\mathbb{N}, \oplus)$ is a commutative group, we have that $\oplus_{i \in \{0,1,\ldots,n-1\}} f(V_i) = \oplus_{i \in \{0,1,\ldots,n-1\}} f(V_{\sigma(i)})$. Now since both $\oplus$ and f are monotone increasing, the values minimizing $\oplus_{i \in \{0,1,\ldots,n-1\}} V_i$ also minimize $\oplus_{i \in \{0,1,\ldots,n-1\}} f(V_i)$.

2.3 Implementing the Priority Rule

Alg. 1 provides an implementation of the priority rule described in Theorem 1, which achieves a time complexity of $O(n \log n)$ by using a heap for incrementally maintaining the set of candidate variables *for which the minimum value is less than or equal to the maximum of (1) the previously matched value plus one, and (2) the minimum value of the not yet matched variables*. Variables are extracted from this heap by increasing maximum value.

```
 1: function min_cost_matching($n$, $V_{0..n-1}$, $cst$, $\oplus$, $e_\oplus$, $f$) : int
      $n$           : number of variables of the alldifferent,
      $V_{0..n-1}$  : variables that must be assigned distinct values,
      $cst$         : maximum allowed minimum cost of the matching,
      $\oplus$      : aggregation operator,
      $e_\oplus$    : neutral element of the aggregation operator $\oplus$,
      $f$           : monotone increasing function applied to each variable of $V_{0..n-1}$.
 2: $V_{s_0}, V_{s_1}, \ldots, V_{s_{n-1}} \leftarrow V_0, V_1, \ldots, V_{n-1}$ sorted by increasing minimum value
 3: $h \leftarrow$ empty heap of indices of var. sorted by increasing maximum value of var.
 4: $i \leftarrow 0$; $o^\star \leftarrow e_\oplus$; $minval \leftarrow \underline{V_{s_0}} - 1$;
 5: for $j = 0$ to $n-1$ do
 6:     $minval \leftarrow \max(minval + 1, \underline{V_{s_j}})$
 7:     while $i < n \wedge \underline{V_{s_i}} \leq minval$ do insert $s_i$ in $h$; $i \leftarrow i+1$ end while
 8:     $ind_j \leftarrow$ extract variable index with smallest maximum value from $h$
 9:     $o^\star \leftarrow o^\star \oplus f(minval)$
10:     if $o^\star > cst \vee minval > \overline{V_{ind_j}}$ then return $cst + 1$ end if
11: return $o^\star$
```

Algorithm 1: return the minimum cost of the matching if it exists and is less than or equal to cst, return $cst + 1$ otherwise.

2.4 Filtering Algorithm

Definition 3 *Given $\mathcal{V}$ a set of variables, a* Hall interval *is an interval $[l, u]$ of values such that there is a set $\mathcal{V}_{[l,u]} \subseteq \mathcal{V}$ of cardinality $u - l + 1$ whose domains are contained in $[l, u]$.*

Blocks of a Minimum Cost Matching. The bound-consistency filtering algorithm of $alldifferent$ [6, 7, 8] adjusts the minimum and maximum values of the variables of $\mathcal{V} \setminus \mathcal{V}_{[l,u]}$ with respect to a Hall interval $[l, u]$. Given a minimum cost matching, consisting of a sequence of variable-value pairs, computed by Alg. 1, and assuming the filtering wrt. Hall intervals was already done, this section introduces the notion of block of variables. The intuition behind is that variables of the same block are in fact equivalent wrt. the filtering related to the arithmetic constraint $f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq cst$. The notion of block will permit evaluating the new cost of a minimum matching under the hypothesis that a variable is assigned another value, without computing from scratch a new minimum cost matching. Before defining a block, let us first introduce some notation and let us recall the property achieved by the bound-consistency algorithm of $alldifferent$. This property will be needed to show that our filtering algorithm reaches the fix point in one single step. After applying bound-consistency on the variables of an $alldifferent$ constraint, Property 1 holds:

Property 1 *Given a Hall interval $[l, u]$, for any variable whose range intersects $[l, u]$ without being included in $[l, u]$, its minimum value (resp. maximum value) is located before (resp. after) the Hall interval.*

Notation 2 *Let ind_k $(0 \leq k < n)$ denote the index of the k^{th} variable selected by Alg. 1. Let v_k* denote the value assigned to variable *V_{ind_k} by Alg. 1.*

Definition 4 *A* block *is a set of consecutive variable-value pairs $(V_{ind_p}, v_p), \ldots, (V_{ind_q}, v_q)$ $(p \leq q)$ that were computed by Alg. 1 such that the three following properties are satisfied:*

i) *(V_{ind_p}, v_p) is the* beginning of the block, *that is $\forall i \in [p, q]$: $\underline{V_{ind_i}} \geq v_p$.*

ii) *(V_{ind_q}, v_q) is the* end of the block *if all variables after V_{ind_q} have their minimum value greater than v_q,*

iii) *$\nexists j \in [p+1, q]$ such that $\forall i \in [j, q]$: $\underline{V_{ind_i}} \geq v_j$.*

Intuitively this definition means that *i)* the minimum value of the variables matched in a block $\mathcal{B}$ is greater than or equal to the minimum value that can be assigned to the variables of the block $\mathcal{B}$, that *ii)* that the block is maximum, and *iii)* that there is no other included sub-block.

Let m denote the number of blocks of the minimum cost matching computed by Alg. 1. low_b and up_b respectively denote the first and last variables of block b in $ind_{0..n-1}$. $first_b$ and $last_b$ respectively denote the smallest and largest matched values of block b.

Example 2 *The sequence of (variable,value) pairs corresponding to the minimum cost matching of the example in the introduction is $(V_0, 1)$, $(V_1, 2)$, $(V_2, 3)$, $(V_3, 4)$, $(V_4, 5)$, $(V_5, 6)$, $(V_6, 7)$, $(V_7, 8)$, $(V_8, 9)$, $(V_9, 12)$. The sequence is decomposed into the following blocks:*

BLOCK 0: $(V_0, 1)$, $(V_1, 2)$, $(V_2, 3)$, $(V_3, 4)$, $(V_4, 5)$, $(V_5, 6)$ *(i.e., $low_0 = 0$, $up_0 = 5$, $first_0 = 1$, $last_0 = 6$); V_0 is not the end of block 0 since $\underline{V_5} = 1$, but V_5 is the end of block 0 since $\underline{V_6}, \underline{V_7}, \underline{V_8}, \underline{V_9}$ are all greater than 6, the value matched to V_5),*

BLOCK 1: $(V_6, 7)$, $(V_7, 8)$ *(i.e., $low_1 = 6$, $up_1 = 7$, $first_1 = 7$, $last_1 = 8$),*

BLOCK 2: $(V_8, 9)$ *(i.e., $low_2 = 8$, $up_2 = 8$, $first_2 = 9$, $last_2 = 9$),*

BLOCK 3: $(V_9, 12)$ *(i.e., $low_3 = 9$, $up_3 = 9$, $first_3 = 12$, $last_3 = 12$).*

Theorem 2 shows how to directly compute the cost of the minimum matching under the hypothesis that we reassign a variable to a value that is different from the one that was assigned in the original minimum cost matching by Alg. 1.

Notation 3 *Given a value v, let $next_v$ denote the smallest unmatched value greater than or equal to v. Given a block b $(0 \leq b < m)$ and a value v that is not matched to a variable of block b, let $h_b(v)$ denote the minimum cost of the matching under the hypothesis that a variable of block b is assigned to value v.*

Theorem 2 *Given as computed by Alg. 1: a matching M of minimum cost $o^\star$, a variable V_{ind_i} $(0 \leq i < n)$ belonging to block b, and its matched value v_i, the minimum cost of the matching under the assumption that V_{ind_i} is assigned a value u_i different from v_i (assuming $alldifferent$ has at least one solution with $V_{ind_i} = u_i$) is:*

1. *If u_i is a value that belongs to the block containing variable V_{ind_i}, the cost is left unchanged, i.e. is equal to $o^\star$.*
2. *Otherwise, the new cost $h_b(u_i)$ is computed by first subtracting from $o^\star$ the largest matched value of block b, and then adding the smallest unmatched value, $next_{u_i}$, greater than or equal to u_i, i.e., $h_b(u_i) = \left(o^\star \oplus^{-1} f(last_b)\right) \oplus f(next_{u_i})$.*

Proof 2 *The proof is done in two steps: (1) after removing variable V_{ind_i} (i.e., unassigning variable V_{ind_i} from value v_i, the new minimum cost matching M' can be obtained by compressing the block b containing value v_i in such a way that the largest value of block b becomes unmatched (i.e., by using $\oplus^{-1}$ for removing from M the contribution of $last_b$), (2) after reintroducing variable V_{ind_i} and assigning it to value u_i the new minimum cost matching M'' can be obtained by using $\oplus$ for adding to M' the contribution of the smallest unmatched value greater than or equal to u_i.* **Step (1).** *Let b be the block of consecutive variable-value pairs containing value v_i. If v_i is the largest value of block b we are done, i.e., the largest value $last_b$ of block b is now unmatched. Otherwise, by Condition (iii) of the definition of a block (see Definition 4), we know that there exists at least one variable $V_{ind_{i'}}$ of block b, that was matched to a value v'_i greater than value v_i, that could possibly be matched to value v_i. So we match $V_{ind_{i'}}$ to v_i and continue in a similar way reorganizing block b until the largest value of block b becomes unmatched.* **Step (2).** *First, assume that u_i corresponds to an unmatched value of the matching obtained at the end of Step 1 (i.e., the matching obtained after unassigning variable V_{ind_i}). Then we are done, since we can directly match V_{ind_i} to u_i. Second, assume that u_i corresponds to a matched value that belongs to a block b' distinct from b. Since, by hypothesis, $alldifferent$ has a solution where V_{ind_i} is assigned value u_i, and since, by hypothesis, no variable of $alldifferent$ has a hole in its domain, we know that we necessarily will have to use an unmatched value that is greater than the largest value $last_{b'}$ of block b'. Since we want to minimize the cost of the new matching, we take the smallest unmatched value.* $\square$

Filtering wrt. a Maximum Cost. The filtering algorithm consists of three phases: (1) It performs bound-consistency on $alldifferent(V_0, V_1, \dots, V_{n-1})$ alone, using a standard bound-consistency algorithm.(2) It computes the minimum cost $o^\star$ of the matching and fails if this cost is greater than cst. This is achieved by using Alg. 1. (3) Finally, Alg. 2 removes those values v from the domain of the variables of block b ($0 \leq b < m$) such that $h_b(v) > cst$.

In order to adjust the maximum value of each variable of a block b ($0 \leq b < m$), we need to identify the largest value v such that $h_b(v) \leq cst$. We look for the largest value v such that $\left(o^\star \oplus^{-1} f(last_b)\right) \oplus f(next_v) \leq cst$. By isolating $next_v$ we get $next_v \leq f^{-1}\left(cst \oplus^{-1}\left(o^\star \oplus^{-1} f(last_b)\right)\right)$. This is tantamount to finding the largest value v such that $next_v$ is less than or equal to a given threshold and can be done in total $O(m)$ time over the blocks.

Example 3 *Figure 1 provides the minimum cost function associated with the last example introduced in the introduction. We use the block information provided in Example 2. The filtering wrt. the different blocks is:*

- *For the block $b = 3$ we have $t = \left\lfloor \frac{4717500}{\frac{4354560}{12}} \right\rfloor = 13$ (t is unmatched) and we adjust the maximum value of V_9 to $max = 13$.*
- *For the block $b = 2$ we have $t = \left\lfloor \frac{4717500}{\frac{4354560}{9}} \right\rfloor = 9$ (t is unmatched) and we adjust the maximum value of V_8 to $max = 9$.*
- *For the block $b = 1$ we have $t = \left\lfloor \frac{4717500}{\frac{4354560}{8}} \right\rfloor = 8$ (t is the last matched value of block 1) and we adjust the maximum values of V_6, V_7 to $max = 8$.*
- *For the block $b = 0$ we have $t = \left\lfloor \frac{4717500}{\frac{4354560}{6}} \right\rfloor = 6$ (t is the last matched value of block 0) and we adjust the maximum values of V_0, V_1, V_2, V_3, V_4, V_5 to $max = 6$.*

```
procedure min_cost_matching_filter($n$, $m$, $V_{0..n-1}$, $cst$, $o^\star$, $\oplus$, $f$, $ind_{0..n-1}$, $low_{0..m-1}$, $up_{0..m-1}$, $first_{0..m-1}$, $last_{0..m-1}$)
  $n$              : number of variables,
  $m$              : number of blocks of the minimum cost matching,
  $V_{0..n-1}$     : variables that must be assigned distinct values,
  $cst$            : maximum allowed min. cost of the matching,
  $o^\star$        : min. cost of the matching returned by Alg. 1,
  $\oplus$         : aggregation operator,
  $f$              : monotone increasing function applied to each variable of $V_{0..n-1}$,
  $ind_{0..n-1}$   : variable indices in the order they are considered by the priority rule,
  $low_{0..m-1}$   : first variable of a block in $ind_{0..n-1}$,
  $up_{0..m-1}$    : last variable of a block in $ind_{0..n-1}$,
  $first_{0..m-1}$ : first matched value of the variables of a block,
  $last_{0..m-1}$  : last matched value of the variables of a block.
$i \leftarrow m - 1$   // each iteration prunes the maximum value of all variables of block $b$ ($0 \leq b < m$)
for $b = m - 1$ downto $0$ do
    $found \leftarrow$ false; $t \leftarrow \lfloor f^{-1}\left(cst \oplus^{-1}\left(o^\star \oplus^{-1} f(last_b)\right)\right)\rfloor$;
    while $\neg found$ do
        if $(i = m - 1 \wedge t > last_i) \vee (i > 0 \wedge last_{i-1} < t \wedge t < first_i)$ then
            $found \leftarrow$ true; $max \leftarrow t$;
        else if $i > b \wedge last_{i-1} < first_i - 1 \wedge t \geq first_i$ then
            $found \leftarrow$ true; $max \leftarrow first_i - 1$;
        else if $i = b$ then
            $found \leftarrow$ true; $max \leftarrow last_i$;
        else
            $i \leftarrow i - 1$
    for $j = low_b$ to $up_b$ do
        adjust maximum of $V_{ind_j}$ to $max$
```

Algorithm 2: third step for achieving bound-consistency for the conjunction.

Theorem 3 *Bound-consistency for the conjunction $alldifferent(V_0, V_1, \dots, V_{n-1}) \wedge f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq cst$ is directly obtained after applying the three phases of the filtering algorithm (i.e., applying bound-consistency for $alldifferent$ (in $O(n \log n)$), computing a minimum cost matching, filtering wrt. a maximum cost) one time, in $O(n \log n)$ time complexity.*

Proof 3 *From Theorem 2 and Phase 3 of the algorithm, any bound v of a variable in $V_0, V_1, \dots, V_{n-1}$ satisfies $h_b(v) \leq cst$. We prove that these three phase are complete (i.e., we do not need to recall any filtering algorithm). We show that filtering wrt. a maximum cost (1) neither removes all solutions from $alldifferent$, (2) nor causes the bound-consistency filtering of $alldifferent$ to perform more filtering. (1) follows from the fact that Alg. 1 already computes a solution for the conjunction. Now for proving (2), we distinguish three cases:*

1. *If filtering wrt. a maximum cost decreases the maximum value of a variable* to an unmatched value *(i.e., a value that does not belong to any block), then no new Hall interval is created; since the bound-consistency filtering of $alldifferent$ is linked to Hall intervals, no further filtering can occur from such domain reductions.*
2. *If filtering wrt. a maximum cost decreases the maximum value of a variable* to a matched value and does not create any new Hall interval, *then again no further filtering can occur.*
3. *If filtering wrt. a maximum cost decreases the maximum value of a variable V* to a matched value v and creates a new Hall interval, *then we have the following situation: V and v belong to the same block b, and v is its largest value; furthermore, variables of block b had their maximum values decreased to the largest value of block b. We successively show that this new Hall interval cannot change the minimum or maximum value of any variable: (i) By construc-*

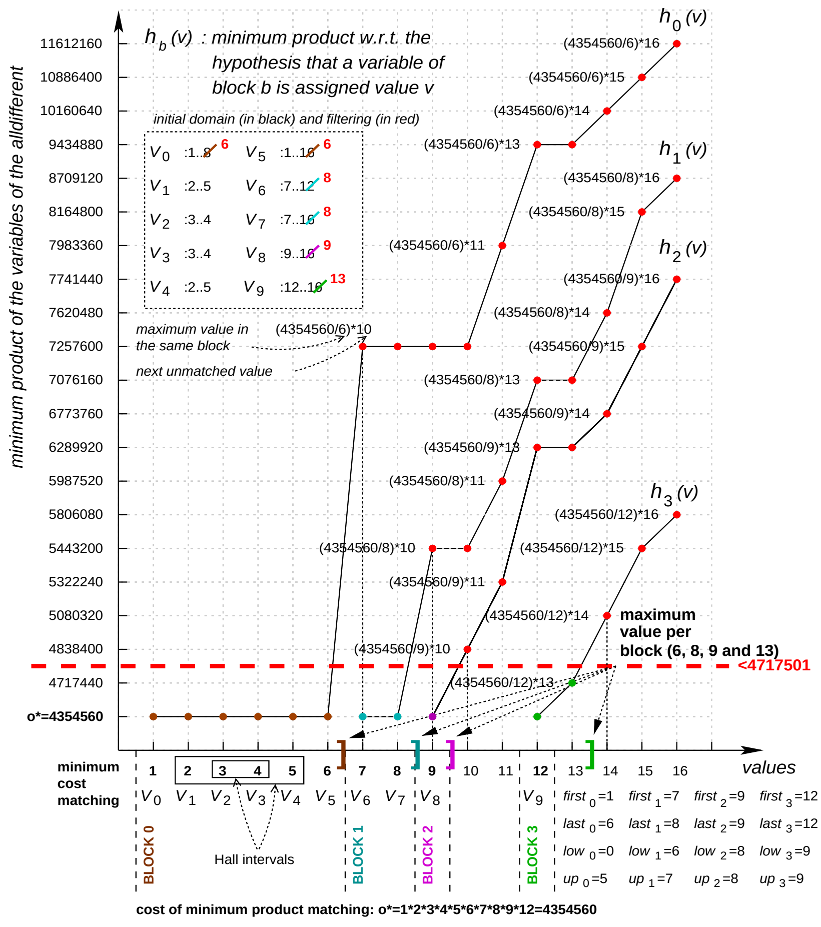

Figure 1. Minimum cost functions $h_b(v)$ $(0 \leq b < 4)$ associated with the 4 blocks of the minimum cost matching of $V_0 \in [1,8] \wedge V_1 \in [2,5] \wedge V_2 \in [3,4] \wedge V_3 \in [3,4] \wedge V_4 \in [2,5] \wedge V_5 \in [1,16] \wedge V_6 \in [7,12] \wedge V_7 \in [7,16] \wedge V_8 \in [9,16] \wedge V_9 \in [12,16] \wedge \mathit{alldifferent}(V_0, V_1, \ldots, V_9) \wedge \prod_{i=0}^{9} V_i \leq 4717500$; matched values of the minimum cost matching are shown in bold on the *values* axis; red points of $h_b(v)$ $(0 \leq b \leq 3)$ are infeasible points wrt. the limit 4717500; first_b and last_b $(0 \leq b \leq 3)$ denote the first and last matched values of block b, while low_b and up_b $(0 \leq b \leq 3)$ denote the index of the first and last variables of block b; consequently, the maximum value of variables of blocks 0, 1, 2 and 3 are respectively set to 6, 8, 9 and 13 (note that assigning 13 to V_9 is still feasible since it leads to a cost of ${}^{4354560}/_{12} \times 13 = 4717440 \leq 4717500$).

tion of the blocks, the variables whose minimum value may belong to this new Hall interval necessarily belong to the same block b. Pruning their minimum value to a value that is greater than the largest value of block b would lead to a contradiction, since the maximum value of the variables of block b were decreased to the largest value of block b. (ii) After filtering wrt. the maximum cost, no variable of a block b' preceding block b can have its maximum value decreased to a value of block b. Consequently, even if a new Hall interval was created in block b, it cannot contain the maximum values (obtained after applying line 15 of Alg. 2) of the variables belonging to blocks that are located before b.

Phases 1 and 2 have a time complexity in $O(n \log n)$, while Phase 3 has a $O(n)$ complexity. □

3 The *alldifferent_arith* Constraint

Rather than just providing a global constraint for directly handling the conjunction $\mathit{alldifferent}(V_0, V_1, \ldots, V_{n-1}) \wedge f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq \mathit{cst}$, we came up with the following global constraint $\mathit{alldifferent\_arith}(\mathcal{V}, \mathcal{C})$, where $\mathcal{V}$ is a set of variables $\{V_0, V_1, \ldots, V_{n-1}\}$ that must be assigned distinct values in $\mathbb{N}^+$, and $\mathcal{C}$ is a set of arithmetic constraints of the following forms:

$$(\sum_{\substack{U_i \in \mathcal{U} \\ \mathcal{U} \subseteq \mathcal{V}}} U_i)\ o_i\ W_i, \quad (\sum_{\substack{U_i \in \mathcal{U} \\ \mathcal{U} \subseteq \mathcal{V}}} U_i^2)\ o_i\ W_i, \quad (\prod_{\substack{U_i \in \mathcal{U} \\ \mathcal{U} \subseteq \mathcal{V}}} U_i)\ o_i\ W_i,$$

where o_i is one of the operators $\leq$, $\geq$ or $=$, and W_i is a constant or a variable. Even if we cannot enforce bound-consistency for $\mathit{alldifferent\_arith}$ in polynomial time when o_i is the equality, coming up with this constraint is motivated by the following reasons: Some benchmarks (e.g., magic squares, hexagons, cubes, bimagic squares) involve an $\mathit{alldifferent}$ constraint together with arithmetic constraints. Each arithmetic constraint is an equality between a left hand side, corresponding to a sum, and a variable, such the left hand side only mentions a subset of variables of the $\mathit{alldifferent}$. This constraint also occurs in sum coloring of graphs [9].

We provide for $\mathit{alldifferent\_arith}$ the three filtering options:

A first option, called G_{BC}, where only bound-consistency is used.

A second option, called G_{BCF}, where we also ignore: (1) values that initially do not belong to any domain of the variables of $\mathit{alldifferent}$, and (2) all already fixed variables of $\mathit{alldifferent}$.

A third option, named G_{AC}, where we also perform arc-consistency on $\mathit{alldifferent}$.

Alg. 1, and 2 can be easily extended in order to take into account variables that were fixed, as well as all values that could not be initially assigned by any variables. Note that by Property 1, after the bound-consistency algorithm of *alldifferent* is completed, the minimum and maximum value of each not yet fixed variable does not equal any fixed variable. Nor does it equal any value that was initially removed from all domains.

n	R_{BC}	R_{AC}	C_{BC}	C_{AC}	G_{BC}	G_{BCF}	G_{AC}
g-2	36.6/1239005	0.04/1207	0.05/*0*	0.04/*0*	*0.01*/*0*	*0.01*/*0*	*0.01*/*0*
g-4	–/–	0.16/3615	0.05/*0*	0.05/*0*	0.01/*0*	*0.0*/*0*	0.02/*0*
g-5	5.53/174385	0.03/428	0.04/*0*	0.06/*0*	*0.01*/*0*	*0.01*/*0*	*0.01*/*0*
g-6	–/–	1.27/40700	0.04/*0*	0.03/*0*	*0.0*/1	0.01/1	0.01/1
g-7	29.7/1235358	0.16/4413	0.04/*0*	0.04/*0*	*0.01*/20	*0.01*/20	*0.01*/4
g-8	–/–	–/–	0.04/*0*	0.05/*0*	*0.01*/18	*0.01*/18	*0.01*/8
g-9	121.0 /3824084	61.0 /1542016	0.05/*0*	0.06/*0*	*0.01*/15	*0.01*/15	0.02/11
g-10	–/–	4.12 /109090	0.06/*0* 0.06/*0*	0.04/*0*	0.01/4	*0.0*/4	0.01/3
g-11	–/–	–/–	0.05/*0*	0.06/*0*	9.22 /176985	8.62 /178916	*0.02*/175
g-12	–/–	0.75/19168	0.06/*0*	0.05/*0*	*0.01*/9	*0.01*/9	*0.01*/6
g-13	–/–	–/–	0.06/*0*	0.07/*0*	*0.01*/12	*0.01*/12	*0.01*/*0*
g-14	–/–	–/–	0.05/*0*	0.07/*0*	*0.01*/33	*0.01*/33	*0.01*/42
mix2-23	3.92/95479	0.02/230	0.04/*0*	0.04/*0*	*0.01*/*0*	*0.01*/*0*	*0.01*/*0*
mix-23	2.56/69671	0.02/288	0.05/*0*	0.04/*0*	0.01/*0*	*0.0*/*0*	0.01/*0*
g-1	–/–	17.6 /373852	0.11/*0*	0.09/*0*	0.02/*0*	*0.01*/*0*	0.03/*0*
g-2	–/–	258.0 /6181267	0.08/*0*	0.11/*0*	*0.02*/32	*0.02*/32	*0.02*/5
g-3	–/–	–/–	0.09/*2*	0.1/*2*	*0.02*/26	*0.02*/26	*0.02*/18

n	R_{BC}	R_{AC}	C_{BC}	C_{AC}	G_{BC}	G_{BCF}	G_{AC}
6	*0.15*/66	*0.15*/66	3.6/*21*	3.57/*21*	0.18/25	0.17/*21*	0.18/*21*
7	–/–	–/–	–/–	–/–	267.0 /49827	*224.0* /34659	227.0 /*34657*
8	–/–	–/–	197.0/1775	202.0/*1773*	23.9/2406	22.2/1806	*21.8*/1806
9	–/–	–/–	166.0/*195*	173.0/*195*	–/–	5.14/*195*	*5.0*/*195*

n	R_{BC}	R_{AC}	C_{BC}	C_{AC}	G_{BC}	G_{BCF}	G_{AC}
8	*0.0*/26	0.01/26	0.04/*2*	0.02/*2*	0.01/6	*0.0*/5	*0.0*/5
9	*0.01*/264	0.02/264	0.16/*63*	0.16/*63*	0.02/175	*0.01*/90	*0.01*/90
10	0.1/1280	0.11/1280	1.03/*329*	1.04/*329*	0.14/889	0.07/403	*0.06*/403
11	0.75/6593	0.7/6593	9.69/*2097*	9.88/*2097*	1.05/3970	0.62/2349	*0.4*/2349
12	33.0 /217318	34.1 /217318	–/–	–/–	64.5 /170336	45.7 /*124053*	*29.1* /*124053*

n	R_{BC}	R_{AC}	C_{BC}	C_{AC}	G_{BC}	G_{BCF}	G_{AC}
	0.03/56	0.02/56	0.21/*15*	0.23/*15*	*0.01*/23	0.02/*15*	0.02/*15*

n	R_{BC}	R_{AC}	C_{BC}	C_{AC}	G_{BC}	G_{BCF}	G_{AC}
	0.0/2	*0.0*/2	0.01/2	*0.0*/2	*0.0*/*0*	0.01/*0*	*0.0*/*0*

Table 1. Benchmark results, from top to bottom: Kakuro, Magic Square, Golomb, Magic Hexagon, and Magic Product. Column 1 gives the instance. Each cell shows CPU time in seconds/backtracks, or –/–, if it timed out.

4 Evaluation

In order to evaluate our filtering algorithm, we select a number of benchmarks which mix *alldifferent* and sum, or product, constraints. Unless otherwise stated, variables were assigned using the order in which they were passed to *alldifferent_arith* and dichotomic search for assigning each variable. For these benchmarks we performed the following evaluations:

- First, with a standard model where the *alldifferent* and the arithmetic constraints are stated separately. We test two variants, one where bound-consistency is used for *alldifferent* [8], and one where arc-consistency is used for *alldifferent* [10]. These two variants are respectively called R_{BC} and R_{AC}.
- A second model that adds to the standard model a *global_cardinality_with_costs* constraint [2] for each linear equality. The cost matrix of *global_cardinality_with_costs* is defined in such a way that each assigned value v has cost v. These two variants are respectively called C_{BC} and C_{AC}.
- A third model using one single *alldifferent_arith* constraint and its three options G_{BC}, G_{BCF}, G_{AC}.

We experimented our constraint with the Kakuro benchmark and some problems of the CSPlib: Magic square and variants, Magic Hexagon, and Golomb.[7] We used SICStus Prolog 4.2 on a quad core 2.8 GHz Intel Core i7-860 machine with 8MB cache per core, running Ubuntu Linux (using only one processor core). A time-out limit of 5 CPU minutes was given. Table 1 summarizes the results.

Using *global_cardinality_with_costs* (options C_{BC} and C_{AC}) leads to a smaller number of backtracks in all the benchmarks, at a price of a slowdown up to 35 times compared to the fastest method. This was expected since, on the one hand, the filtering algorithm of *global_cardinality_with_costs* is very heavy, and on the other hand it performs arc-consistency[8] as opposed to bound-consistency.

The new methods presented in this paper, i.e., G_{BC} and G_{BCF}, usually lead to the fastest answer. Taking into account the fixed variables (G_{BCF}) sometimes allows to reduce the time by a factor of two compared to G_{BC}. However, performing in addition full arc-consistency (i.e., G_{AC}) usually does not pay off, except for one instance of Kakuro as well as for Golomb.

5 Conclusion

We have provided a generic $O(n \log n)$ bound-consistency filtering algorithm for handling the conjunction $V_0, V_1, \ldots, V_{n-1} \in \mathbb{N}^+ \wedge$ *alldifferent*$(V_0, V_1, \ldots, V_{n-1}) \wedge f(V_0) \oplus f(V_1) \oplus \cdots \oplus f(V_{n-1}) \leq cst$, where $\oplus$ and f have given properties. We have evaluated these new methods on a number of benchmarks. A challenging question is whether a combination of this filtering algorithm and dedicated heuristics could solve open instances of the bimagic square problem.

REFERENCES

[1] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127, 1978.

[2] J.-C. Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3–4):387–405, 2002.

[3] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalog, 2nd ed. (rev. a). Technical Report T2012-03, SICS, 2012.

[4] C. Bessière. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of CP*, chapter 3. Elsevier, 2006.

[5] N. Lipski and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15:324–346, 1981.

[6] J.-F. Puget. A fast algorithm for the bound consistency of *alldiff* constraints. In *Proc. AAAI*, pages 359–366. AAAI Press, 1998.

[7] A. Lopez-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the *alldifferent* constraint. In *Proc. IJCAI*, pages 245–250, 2003.

[8] K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the *sortedness* and the *alldifferent* constraint. In *Proc. CP*, volume 1894 of *LNCS*, pages 306–319. Springer-Verlag, 2000.

[9] Mohammad R. Salavatipour. On sum coloring of graphs. *Discrete Appl. Math.*, 127:477–488, May 2003.

[10] J.-C. Régin. A filtering algorithm for constraints of difference in CSP. In *Proc. AAAI*, pages 362–367, 1994.

[11] P. Galinier, B. Jaumard, R. Morales, and G. Pesant. A constraint-based approach to the Golomb ruler problem. In *Proc. CPAIOR*, 2001.

[7] For Golomb we combine the essential *alldifferent* constraint on all the deltas with the redundant linear constraints on the deltas [11].

[8] More precisely, it performs arc-consistency independently for the $\leq$ side and the $\geq$ side of the equality, but not for the conjunction of the two sides.