Efficient Implementation of the Global Cardinality Constraint with Costs

Margaux Schmied
Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France

Jean-Charles Régin
Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France

Abstract

The success of Constraint Programming relies partly on the global constraints and implementation of the associated filtering algorithms. Recently, new ideas emerged to improve these implementations in practice, especially regarding the all different constraint.

In this paper, we consider the cardinality constraint with costs. The cardinality constraint is a generalization of the all different constraint that specifies the number of times each value must be taken by a given set of variables in a solution. The version with costs introduces an assignment cost and bounds the total sum of assignment costs. The arc consistency filtering algorithm of this constraint is difficult to use in practice, as it systematically searches for many shortest paths. We propose a new approach that works with upper bounds on shortest paths based on landmarks. This approach can be seen as a preprocessing. It is fast and avoids, in practice, a large number of explicit computations of shortest paths.

2012 ACM Subject Classification Computing methodologies → Planning for deterministic actions; Theory of computation → Constraint and logic programming

Keywords and phrases global constraint, filtering algorithm, cardinality constraints with costs, arc consistency

Digital Object Identifier 10.4230/LIPIcs.CP.2024.27

Funding This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

1 Introduction

In Constraint Programming (CP), a problem is defined on variables and constraints. Each variable is provided with a domain defining the set of its possible values. A constraint expresses a property that must be satisfied by a set of variables. CP uses a specific resolution method for each constraint.

The success of CP relies on the use of high-performance filtering algorithms (also known as propagators). These algorithms remove values from variable domains that are not consistent with the constraint, i.e. that do not belong to a solution of the constraint's underlying sub-problem. The most well-known propagator is that of the all different (alldiff) constraint, which specifies that a set of variables must all take different values. The efficiency in practice of that propagator strongly depends on its implementation. Thus, algorithms proposing practical improvements on Régin's algorithm [15] are still appearing [22, 21].

In this article, we consider another constraint introduced by Régin that is also popular [3, 12, 18, 7, 4]: the cardinality constraint with costs [14]. We propose to try to speed up its filtering algorithm when there is nothing to deduce. This is often the case at the start of the search, particularly as the optimal value is far from known. In addition, at this stage, the gains can be significant since few values have been removed from the domains, and so the complexity of the algorithms is greater. This approach can be particularly interesting with

aggressive restarting methods and could simplify the use of CP: there is less need to worry about the inference strength of constraints versus their cost. We can worry less about the type of filtering to be used and consider the arc consistency right away.

The global cardinality constraint (gcc) [16] is a generalization of the alldiff constraint. A gcc is specified in terms of a set of variables $X = \{x_1, ..., x_p\}$ which take their values in a subset of $V = \{v_1, ..., v_d\}$. It constrains the number of times a value $v_i \in V$ is assigned to variables in X to belong to an interval $[l_i, u_i]$.

A gcc with costs (costgcc) is a generalization of a gcc in which a cost is associated with each value of each variable. Then, each solution of the underlying gcc is associated with a global cost equal to the sum of the costs associated with the assigned values of the solution. In a costgcc constraint the global cost must be less than a given value, H.

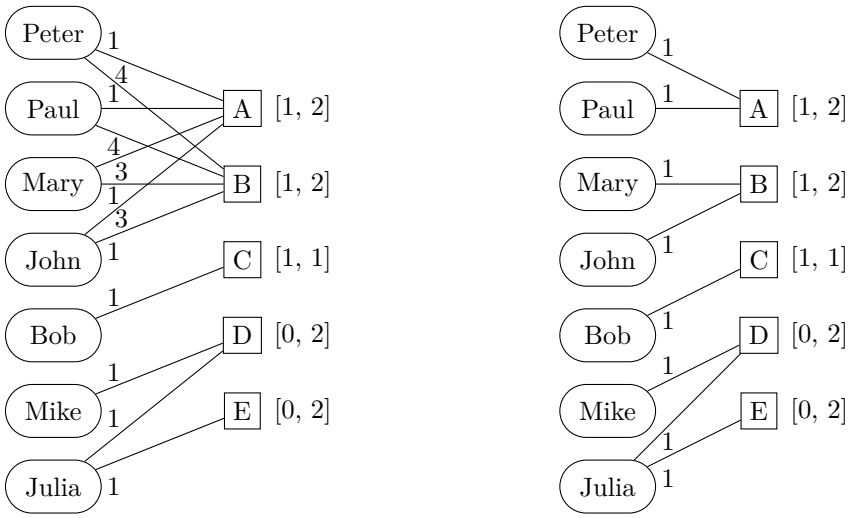


Figure 1 Example of a global cardinality constraint with costs. Source [14]. The sum of assignment costs must be less than or equal to 11. On the left, the original problem and on the right, the same problem after deleting all arcs that cannot belong to a solution.

Cardinality constraints with costs has proved useful in many real-life applications, such as routing, scheduling, rostering, or resource allocation problems. The total costs are often used for expressing preferences, time or cost.

Figure 1 gives an example of a costgcc constraint and the associated filtering algorithm. There are 7 workers represented by the variables $Peter, Paul, Mary, John, Bob, Mike, Julia$ and 5 tasks represented by the values A, B, C, D, E. Each worker has the ability to perform certain tasks and must perform exactly one of them. There is an arc from a worker to a task if the worker can perform the task, its cost corresponding to the time it takes the worker to perform the task. A task has a capacity defining the number of times it must be performed. For example, A must be performed between 1 and 2 times. The objective is to find an assignment whose sum of costs is less than 11. The best possible assignment has a cost of 7, so it is a solution. On the right-hand side of Figure 1, all arcs that cannot belong to a solution have been removed. For example, the arc $(Peter, B)$ can be deleted. If B is assigned to $Peter$, then the maximum capacity of B will be exceeded, so the arc $(Mary, B)$ or $(John, B)$ cannot be part of the solution. If $(John, B)$ is kept, then a value must be assigned to $Mary$, the only possibility is $(Mary, A)$ with a cost of 3. The cost of all assignments is now 12, which is more than 11, so this is not a solution. Similarly, if $(Mary, B)$ is kept, then the only possibility for $John$ is $(John, A)$ with a cost of 3 and the total cost is 12, which is too high.

The filtering algorithm associated with a costgcc constraint [14] can be described as repetitive. First, it computes a maximum flow at minimum cost to determine whether the constraint is consistent (i.e., admits a solution). Then, to find out whether a variable x can be instantiated with a value a, it tries to pass a unit of flow through the arc representing the assignment of a to x so that the total cost of the flow is less than H. This operation involves computing min-cost flow through an arc from a given min-cost flow. This can be done by searching for a shortest path between x and a in the residual graph of the min-cost flow. Furthermore, it has been shown that it is possible to avoid computing a shortest path for each value of each variable and that computing one shortest path per assigned value (which is less than the number of variables) is sufficient [14]. Unfortunately, the algorithm is repeated for each assigned value, which often proves prohibitively expensive.

In this paper, we introduce a new approach avoiding this repetitive aspect as much as possible. Our approach is based on several observations:

- Finding a min-cost flow for each assignment is not necessary. Finding that there exists a flow whose cost is less than H is enough.
- It is not necessary to compute any path exactly because we are only interested in their costs, not the path. Further, the exact value of the cost is not necessary either. An upper bound below a maximum cost is sufficient.
- The use of landmarks (i.e., particular nodes) have proved their worth in speeding up computations of the shortest paths between large elements (millions of nodes) [6]. Let x and y be two nodes of a graph and p be another node called landmark, then we have: $d(x,p) + d(p,y) \geq d(x,y)$ where $d(i,j)$ is the shortest path distance from i to j. Thus, by selecting one or several good landmarks p we can find a good upper bound of $d(x,y)$ for each pair of nodes x, y.
- Calls to the filtering algorithm often do not remove any value. This means that the margin (i.e., slack between H and the min-cost flow value) is often large relative to the data, so using the upper bound should give good results.

On the basis of the above, we propose to introduce preprocessing in order to reduce the effective shortest path computations as proposed by Régin's algorithm. Our approach is to search for landmarks and use them to compute upper bounds on paths to avoid unnecessary explicit shortest path computations. We consider several types of landmarks to integrate the structure of the graph, such as landmarks at the periphery (outline) of the graph or at the center. The advantage of this approach is its low cost because only two shortest paths are required per landmark. We also introduce a way to quickly detect whether a costgcc constraint is arc consistent.

The paper is organized as follows. Section 2 recalls some preliminaries on constraint programming, graph and flow theory. Section 3 describes Régin's algorithm because our method is based on it. Section 4 introduces upper bounds on shortest paths based on landmarks and, in Section 5, the arc consistency algorithm is accordingly adapted. Section 6 details some landmark selection methods. Section 7 gives some experiments on classical problems showing that our approach dramatically reduces the number of computed shortest paths.

2 Preliminaries

The following definitions, theorems and algorithms are based on the following papers and books: [14, 2, 10, 17, 1].

Constraint Programming

A finite constraint network $\mathcal{N}$ is defined as a set of $n \in N$ variables $X = \{x_1, \ldots, x_n\}$, a set of current domains $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible values for variable x_i, and a set $\mathcal{C}$ of constraints between variables. We introduce the particular notation $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$ on which constraint definitions were stated. A constraint C on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ that specifies the allowed combinations of values for the variables $x_{i_1}, \ldots, x_{i_r}$. An element of $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ is called a tuple on $X(C)$ and is denoted by τ. In a tuple τ, the assignment of the i^{th} variable is denoted by τ_i. $var(C, i)$ represents the i^{th} variable of $X(C)$. A value a for a variable x is often denoted by (x, a). Let C be a constraint. A tuple τ on $X(C)$ is valid if $\forall (x, a) \in \tau, a \in D(x)$. C is consistent iff there exists a tuple τ of $T(C)$ which is valid. A value $a \in D(x)$ is consistent with C iff $x \notin X(C)$ or there exists a valid tuple τ of $T(C)$ with $(x, a) \in \tau$.

The costgcc constraint is formally defined as follows.

▶ **Definition 1** ([14]). *A **global cardinality constraint with costs** is a constraint C associated with a cost function on $X(C)$ `cost`, an integer H and in which each value $a_i \in D(X(C))$ is associated with two positive integers l_i and u_i*

$$T(C) = \{\ \tau \text{ such that } \tau \text{ is a tuple on } X(C)$$
$$\text{and } \forall a_i \in D(X(C)) : l_i \leq \#(a_i, \tau) \leq u_i$$
$$\text{and } \Sigma_{i=1}^{|X(C)|} cost(\text{var}(C, i), \tau[i]) \leq H\ \}$$

It is denoted by $costgcc(X, l, u, cost, H)$.

To understand how arc consistency on a costgcc is established, some concepts from graph theory and flow theory are required.

Graph theory

A directed graph or digraph $G = (X, U)$ consists of a node set X and an arc set U, where every arc (x, y) is an ordered pair of distinct nodes. We will denote by $X(G)$ the node set of G and by $U(G)$ the arc set of G. The cost of an arc is a value associated with the arc. When costs are associated with arcs, one should talk about weighted directed graphs.

A path from node x_1 to node x_k in G is a list of nodes $[x_1, ..., x_k]$ such that (x_i, x_{i+1}) is an arc for $i \in [1..k-1]$. The path is called simple if all its nodes are distinct. The cost of a path P, denoted by $cost(P)$, is the sum of the costs of the arcs contained in P. A shortest path from a node s to a node t is a path from s to t whose cost is minimum.

Flow theory

Let G be a digraph where each arc (x, y) is associated with three information: l_{xy} the lower bound capacity, u_{xy} the upper bound capacity and c_{xy} the cost of the arc.

A flow in G is a function f satisfying the following two conditions:

- For any arc (x, y), f_{xy} represents the amount of some commodity that can "flow" through the arc. Such a flow is permitted only in the indicated direction of the arc, i.e., from x to y. For convenience, we assume $f_{xy} = 0$ if $(x, y) \notin U(G)$.
- A conservation law is fulfilled at each node: $\forall y \in X(G) : \sum_x f_{xy} = \sum_z f_{yz}$.

The cost of a flow f is $cost(f) = \sum_{(x,y) \in U(G)} f_{xy} c_{xy}$.

The feasible flow problem consists in computing a flow in G that satisfies the capacity constraint. That is finding f such that $\forall(x,y) \in U(G)\ l_{xy} \leq f_{xy} \leq u_{xy}$. The minimum cost flow problem consists in finding a feasible flow f such that $cost(f)$ is minimum.

A min cost flow can be computed thanks to the augmenting shortest path algorithm. The main idea of the basic algorithms of flow theory is to proceed by successive improvement of flows that are computed in a graph in which all the lower bounds are zero and the current flow is the zero flow (i.e., the flow value is zero on all arcs).

First, assume that there is no lower capacity. So, consider that all the lower bounds are equal to zero and suppose that you want to increase the flow value for an arc (x,y). In this case, the zero flow is a feasible flow. Let P be a shortest path from y to x different from (y,x), and $val = min(\{u_{xy}\} \cup \{u_{ab} \text{ s.t. } (a,b) \in P\})$. Then we can define the function f on the arcs of G such that $f_{ab} = val$ if $(a,b) \in P$ or $(a,b) = (x,y)$, and $f_{ab} = 0$ otherwise. This function is a flow in G and $f_{xy} > 0$. Now, from this flow we can define a particular graph without any flow value and all lower bounds equal to zero, the residual graph.

▶ **Definition 2.** *The **residual graph** for a given flow f, denoted by $R(f)$, is the digraph with the same node set as in G and with the arc set defined as follows:*
$\forall(x,y) \in U(G)$:

- *$f_{xy} < u_{xy} \Leftrightarrow (x,y) \in U(R(f))$ and has cost $rc_{xy} = c_{xy}$ and upper bound capacity $r_{xy} = u_{xy} - f_{xy}$.*
- *$f_{xy} > l_{xy} \Leftrightarrow (y,x) \in U(R(f))$ and has cost $rc_{yx} = -c_{xy}$ and upper bound capacity $r_{yx} = f_{xy} - l_{xy}$.*

All the lower bound capacities are equal to 0.

Then, we can select an arc and apply the previous algorithm on this arc in order to increase its flow value. By dealing only with shortest path we can guarantee that the computed flow will have a minimum cost.

Now consider the lower capacities. In this case, we can use the algorithm mentioned by Régin:

Start with the zero flow f^o. This flow satisfies the upper bounds. Set $f = f^o$, and apply the following process while the flow is not feasible:

1) pick an arc (x,y) such that f_{xy} violates the lower bound capacity in G (i.e., $f_{xy} < l_{xy}$).
2) Find P a shortest path from y to x in $R(f) - \{(y,x)\}$.
3) Obtain f' from f by sending flow along P; set $f = f'$ and goto 1)

If, at some point, there is no path for the current flow, then a feasible flow does not exist. Otherwise, the obtained flow is feasible and is a minimum cost flow.

3 Filtering Algorithm

Our work builds on top of the original costgcc filtering (i.e., [14]). Before presenting how we speed up the algorithm for costgcc, let us briefly review the original algorithm.

There is a relation between a costgcc and the search for min-cost flow in a particular graph.

▶ **Definition 3** ([14])**.** *Given $C = costgcc(X,l,u,cost,H)$. The value graph of C is the bipartite graph $GV(C) = (X(C), D(X(C)), U)$ where $(x,a) \in U$ if $a \in D_x$. The **value network** of C is the directed graph $N(C)$ with l_{xy} the lower bound capacity, u_{xy} the upper bound capacity and c_{xy} the cost on arc from the node x to the node y. $N(C)$ is obtained from the value graph $GV(C)$ by:*

- *Orienting each edge of* $GV(C)$ *from values to variables.* $\forall x \in X(C) : \forall a \in D(x) : l_{ax} = 0$, $u_{ax} = 1$ *and* $c_{ax} = cost(x, a)$.
- *Adding a node* s *and an arc from* s *to each value.* $\forall a \in D(X(C))$: $l_{sa} = l_a$, $u_{sa} = u_a$ *and* $c_{sa} = 0$.
- *Adding a node* t *and an arc from each variable to* t. $\forall x \in X(C) : l_{xt} = 1$, $u_{xt} = 1$ *and* $c_{xt} = 0$.
- *Adding an arc* (t, s) *with* l_{ts}= u_{ts}=$|X(C)|$ *and* c_{ts}=0.

▶ **Property 4** ([14]). *A costgcc* C *is consistent iff there is a minimum cost flow in the value network of* C *whose cost is less than or equal to* H.

Figure 2 represents the residual graph of the value network of the costgcc constraint defined in Figure 1. This is the graph computed from a flow resulting of the min cost flow algorithm applied on the value network. In the residual graph, the optimal solution corresponds to the arcs oriented from the variables to the values. The optimal cost value is 7.

For clarity, in the remainder, we consider that $C = costgcc(X, l, u, cost, H)$ is a costgcc constraint and that f is min cost flow in $N(C)$. We also assume that the arc consistency of the underlined gcc of C has been established.

The consistency of a value relates to the existence of a particular path in the residual graph of the min cost flow.

▶ **Property 5** ([14]). *A value* a *of a variable* x *is not consistent with* C *iff the two following properties hold:*

- $f_{ax} = 0$
- $d_{R(f)}(x, a) > H - cost(f) - rc_{ax}$

where $d_{R(f)}(x, a)$ *is the shortest path between* x *and* a *in the residual graph of* f, *and* rc_{ax} *is the residual cost of the arc* (a, x).

To establish arc consistency, the previous property could be checked for each value of each variable. However it is possible to reduce the number of computed shortest paths.

▶ **Corollary 6** ([14]). *Given any variable* x *and* b *the value of* x *such that* $f_{bx} = 1$. *Then, the value* a *of* x *is not consistent with* C *iff the two following properties hold:*

- $f_{ax} = 0$
- $d_{R(f)}(b, a) > H - cost(f) - rc_{ax} - rc_{xb}$

An example of the application of Property 5 is given in Figure 2. The length of the shortest path from *Julia* to E has a cost of -1 (see blue arcs) and the cost of the arc (E, *Julia*) is $rc_{EJulia} = 1$. Thus we have $d_{R(f)}(Julia, E) = -1$ and $H - cost(f) - rc_{EJulia} = 11 - 7 - 1 = 3$, so we have $-1 \leq 3$. From Property 5 it means that $(E, Julia)$ is consistent. The shortest path from *Peter* to B is $d_{R(f)}(Peter, B) = 1$ and the cost of the arc (B, *Peter*) is $rc_{BPeter} = 4$ (see red arcs). Hence, we have $H - cost(f) - rc_{BPeter} = 11 - 7 - 4 = 0$, so $1 > 0$. (B, *Peter*) is inconsistent, the arc is then removed.

4 Upper Bounds of Shortest Paths

Although Corollary 6 reduces the number of computations required to establish the arc consistency of the constraint, it systematically computes a large number of shortest paths. Precisely, the algorithm involves computing the shortest path between each assigned value and all other values which makes it difficult to use in practice. In addition, the constraint is often arc consistent, rendering any computation useless. The aim of our approach is therefore to reduce the number of operations computed unnecessarily.

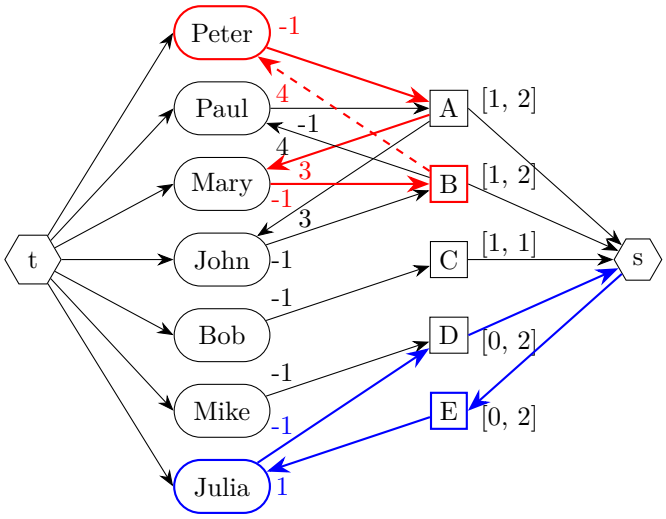


Figure 2 Example of computation of the consistency for the arcs $(E, Julia)$ and $(B, Peter)$. The value B is not consistent with $Peter$. Thus, the dotted arc can be removed from the graph.

We present a much more applied approach, based on the fact that Corollary 6 relies on the existence of a path of length less than a given value. It is not necessary to know the path precisely, or even to know its value. An upper bound is sufficient.

We can therefore immediately establish the following proposition:

▶ **Proposition 7.** *Let $B^+(x, a) \geq d_{R(f)}(x, a)$ be any upper bound on the length of the shortest path from x to a. If*

$$B^+(x, a) \leq H - cost(f) - rc_{ax}$$

then the value a of a variable x is consistent with C.

A good way of obtaining an upper bound on a distance between two points is to use the triangle inequality. Here we are talking about the triangle inequality with respect to the shortest path distances in the graph, not an embedding in Euclidean space or some other metric, which need not be present.

▶ **Property 8.** *Let x, y, and p be three nodes of a graph. According to the triangle inequality computed on shortest paths, we have:*

$$d(x, p) + d(p, y) \geq d(x, y)$$

Here, p is a particular node called landmark.

Upper bounds obtained by the triangular inequality have been shown to be useful for guiding the computation of shortest paths. The ALT algorithm, yielding excellent results in practice for computing shortest paths in a very large graph, is based on this technique [6].

Property 5 and Corollary 6 can be rewritten for landmarks:

▶ **Proposition 9.** *Given any variable x such that $f_{bx} = 1$, a any value of x and p any landmark. If one of the two condition is satisfied*

$$d_{R(f)}(x, p) + d_{R(f)}(p, a) \leq H - cost(f) - rc_{ax}$$
$$d_{R(f)}(b, p) + d_{R(f)}(p, a) \leq H - cost(f) - rc_{ax} - rc_{xb}$$

then the value a of x is consistent with C.

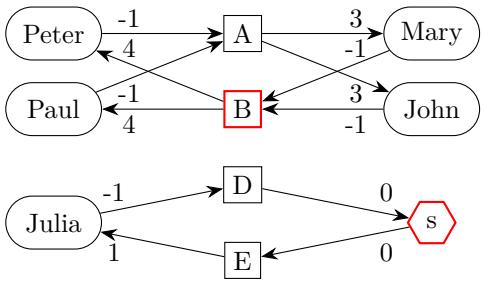


Figure 3 Example of landmark use. Nodes B and s, shown in red, are selected as landmarks.

The residual graph may have several strongly connected components. Each component must be treated separately. Thus, at least one landmark per component must be selected.

Thanks to the use of upper bounds we can go even further. It is possible to compute the consistency of all values of variables of a strongly connected component by checking a single condition.

▶ **Definition 10.** *Consider S a strongly connected component of $R(f)$, p a landmark in S, $x \in S$ a variable, and a a value of x. We define:*

- $d^{max}_{R(f)}(\cdot, p) = \max_{x \in S}(d_{R(f)}(x, p))$
- $d^{max}_{R(f)}(p, \cdot) = \max_{x \in S}(d_{R(f)}(p, x))$
- $rc^{max} = \max_{x \in S, a \in D(x)}(rc_{ax})$

This leads to the following proposition:

▶ **Proposition 11.** *Given S a strongly connected component of $R(f)$ and p a landmark in S. If*

$$d^{max}_{R(f)}(\cdot, p) + d^{max}_{R(f)}(p, \cdot) \leq H - cost(f) - rc^{max}$$

then all the values of all the variables involved in S are consistent with C.

The advantage of this method is that if the condition is satisfied, we can guarantee that all the values of a strongly connected component are consistent by computing only two shortest paths per landmark.

Figure 3 gives an example of a residual graph on which Proposition 9 or 11 can be applied. There are 2 strongly connected components $\{Peter, A, Mary, Paul, B, John\}$ and $\{Julia, D, s, E\}$. At least 2 landmarks are required (one for each component). We select B and s arbitrarily.

Thanks to Proposition 11 we see that the maximum shortest path through s is the path from D to $Julia$ with $d^{max}_{R(f)}(\cdot, S) = d(D, s) = 0$ and $d^{max}_{R(f)}(s, \cdot) = d(s, Julia) = 1$. Furthermore, the longest arc of this strongly connected component is $rc^{max} = rc_{EJulia} = 1$. Thus we have $d(D, s) + d(s, Julia) = 1$ and $H - cost(f) - rc_{EJulia} = 3$, so we have $1 \leq 3$. This confirms that all the values of variables in the strongly connected component of s are consistent with the constraint. If the Proposition 11 can guarantee that all values of variables are consistent in this strongly connected component then we can easily deduce that the Proposition 9 can also do it.

For the first strongly connected component, Proposition 9 and 11 do not guarantee the consistency of the values of the variables. It is therefore necessary to compute exact shortest paths between values and variables.

Algorithm 1 Arc Consistency Algorithm for a Strongly Connected Component.

```
arcConsistencyWithLandmarks(f, R_f, S, P);
for p ∈ P do
    d(p, ·) ← shortestPath_{R(f)}(p, ·) // shortest path in R(f) ;
    d(·, p) ← shortestPath_{R̄(f)}(p, ·) // shortest path in R̄(f), the reverse graph of R(f) ;
// Check of Proposition 11 ;
rc^max ← max_{x∈S,a∈D(x)}(rc_ax) ;
for p ∈ P do
    d^max_{R(f)}(·, p) ← max_{x∈S}(d_{R(f)}(x, p)) ;
    d^max_{R(f)}(p, ·) ← max_{x∈S}(d_{R(f)}(p, x)) ;
    if d^max_{R(f)}(·, p) + d^max_{R(f)}(p, ·) ≤ H − cost(f) − rc^max then
        // all values of all variables of S are consistent ;
        return ;
Δ ← {a such that f_sa > 0} ;
for value b ∈ Δ do
    for x such that f_bx = 1 do
        δ(b) ← {a such that a ∈ D(x) and a ≠ b} ;
        computePath ← false ;
        // Check of Proposition 9 ;
        for a ∈ D(x) while not computePath do
            dpmin ← min_{p∈P}(d_{R(f)}(x, p) + d_{R(f)}(p, a));
            if dpmin > H − cost(f) − rc_ax then
                computePath ← true ;
        // Check for an explicit shortest path computation ;
        if computePath then
            d_{R(f)}(b, ·) ← shortestPath(b, ·) ;
            for a ∈ D(x) do
                if d_{R(f)}(b, a) > H − cost(f) − rc_ax − rc_xb then remove a from D(x);
```

5 Improved Filtering Algorithm

We can now describe Algorithm 1, which eliminates values that are inconsistent with the constraint. The algorithm takes as parameters a min cost flow f, its residual graph $R(f)$, a strongly connected component represented by its set of nodes S and P a set of landmarks of S. This algorithm must therefore be called for each strongly connected component. The algorithm begins by checking whether Proposition 11 holds. If true, then the algorithm stops, since this means that all the values of the variables in the connected component S are consistent. Otherwise, it is necessary to check each value potentially inconsistent individually. So, for each of those values Proposition 9 is checked. If it is satisfied, then the value is consistent, otherwise an explicit shortest path is computed to determine whether the value is consistent or not.

When testing Corollary 6, we could refine the algorithm by identifying the nodes for which we need to search for a shortest path from b to them, but this is not interesting in practice as the shortest path algorithm will quickly find that they are at an acceptable distance from b.

Practical improvements

One can compute landmarks only when they are needed. This consideration is effective in practice and a simple modified version of the basic algorithm is possible. This modification proceeds by iteration over the landmarks. Consider V the set of values for which a shortest path must be computed.

The following process is defined: The landmark p is considered. Proposition 11 is checked according to p. If it is satisfied then V is emptied (all values are consistent) otherwise the values V that satisfies Proposition 9 according to p are removed from V, because they are consistent.

This process is repeated while V is not empty and some landmarks remain. In other words, the landmarks are successively considered while the status of some values is not determined.

If there are no more landmarks to compute, then, and only then, shortest paths are explicitly computed for the value in V. In practice, it is frequent to find that all values are consistent without using all the landmarks. This practical improvement means that not all landmarks need to be systematically computed.

Note that the landmark approach subsumes all the practical improvements proposed by Régin.

As far as the shortest path algorithm is concerned, it is interesting to remove the negative costs from the residual graph in order to use Dijkstra's algorithm, as mentioned by Régin. It only requires one shortest path computation [14].

Complexity

Let SP be the complexity of computing a shortest path from one node to all others. Régin's algorithm has a complexity of $\Omega(\delta \times SP)$ in the best case and $O(\delta \times SP)$ in the worst case, where δ is the number of assigned values. With landmarks, the complexity in the best case is in $\Omega(FindP + |P| \times SP)$ where $|P|$ is the number of landmarks and $FindP$ is the complexity of finding the landmarks. This complexity is obtained when Proposition 11 detects that every value is consistent. Note that, this detection can happen on the first landmark and so we can have $|P| = 1$. In the worst case, the complexity is the same as that of Régin's algorithm, provided that $|P|$ is in $O(\delta)$ and $FindP$ is in $O(\delta \times SP)$. As with the ALT method, we consider several landmarks in order to have a better chance of finding landmarks that avoid explicit shortest path computations.

6 Landmark Selection

There are different methods for selecting landmarks.

Random

A landmark is randomly selected. This method is fast to find landmarks, so we used it to compare to other methods.

Outline

The method is based on an approximation of the outlines of a graph.

▶ **Definition 12.** *The **outlines of a graph** G are defined by one or more pairs of nodes (x, y) with $x, y \in X$ that maximize the minimum distance between x and y among all pairs of nodes in the graph.*

To find the pair of nodes representing the outline, we use a well-known 2-approximation. First, we perform a shortest-path search starting from an arbitrary node x, then select the node y, which is the furthest node from x, as a landmark. Next, the shortest paths from

y are computed, and z the node furthest from y is selected as the second landmark. The outline is therefore (y, z) and the landmarks y and z. The complexity of finding a landmark depends on the complexity of computing the two shortest paths, and is therefore in $O(SP)$.

Center

The method is based on an approximation of the center of a graph.

▶ **Definition 13.** *The **center of a graph** G is defined by one or more nodes $x \in X$ that minimize the maximum distance from them to any other node in the graph.*

As the definition of the outlines and the center are similar, the selection of landmarks is also similar. We search for the outlines (x, y) with $x, y \in X$ and select as the center the node z that lies halfway between x and y. The landmark is z. The complexity is the same as for the previous method, $O(SP)$.

Outline & center

The method is based on both outlines and center of a graph, that is a pair of outlines and a center are selected as described earlier.

Maximum degree

The method is based on the node's degree. We select as a landmark the node $x \in X$ that maximizes $(deg^+(x) + deg^-(x)) \times min(deg^+(x), deg^-(x))$, where $deg^+(x)$ (resp. $deg^-(x)$) is the number of outgoing arcs of x (resp. incoming arcs to x). We used this formula to choose nodes with a large number of predecessors and successors. We also expect to choose nodes with a good balance between predecessors and successors. To find landmarks we traverse every node once, giving a complexity of $O(|X|)$.

All these methods must be applied for each strongly connected component.

7 Experimentation

The experiments were carried out on a computer with an Intel Core i7-3930K CPU 3,20 GHz processor, 64 GB of memory and running under Windows 10 Pro. All algorithms were implemented in Java (openjdk-17) in an internal CP solver.

The results relate to the solving of four problems, the traveling salesman problem (TSP) [9], the StockingCost problem [8], the flexible job shop scheduling problem (FJSSP) [13, 20] and a problem of assigning child to activities (CHILD) [19]. The TSP data are the instances (77) of the TSPLIB [5] having less than 1,500 cities. Some of them involve more than a million of edges. The StockingCost data are those used in a Houndji's paper [8], this is random data distributed define as 100 instances with 500 periods. Precisely, the StockingCost instances have 500 variables and 500 values. The FJSSP data come from two different sources, given by Pelleau [13] and Weise [20]. There are 370 instances with between 5 and 20 variables linked to a few values (between 5 and 10), and most instances have between 50 and 300 arcs. The CHILD instance contains only real-life data from [19]. There are 623 children and 317 activities. Each child must be assigned to one activity. One activity can be associated with multiple children.

For each instance of each problem, we measure the information relating to the establishment of the arc consistency of the costgcc constraint at the root of the search tree. The mean of the results for each data set are reported in the tables.

The H value of the TSP instances comes from the heuristic of Lin-Kernighan [11]. Most of the time, this value is the optimal value. For the instances StockingCost, FJSSP and CHILD the regular H is the smallest value such that there exists at least one solution and the big H is twice as large as the regular H.

It is important to pay close attention to the relationship between the value of H and the value of the minimum-cost flow. Indeed, the costgcc constraint sometimes represents a lower bound of the optimal solution, and this lower bound can be more or less distant from the optimal solution. So if H is the value of the optimal solution, then the min-cost flow may well have a much lower value. This is particularly true for the TSP problem.

Shortest paths are computed by using Dijkstra's algorithm and strongly connected components are computed by using Tarjan's algorithm.

The following abbreviations are used for the landmark selection methods: C for the center selection, O for the outline selection, C & O for the combination between center and outline, Deg for the selection based on the maximum degree and R for the random selection. In addition, line 5+ contains the minimum values for a number of landmarks ranging from 5 to 10.

7.1 Results Tables

We consider a shortest path calculation to be the calculation of the shortest paths from one node to all the others.

The number of shortest paths calculated is an important parameter for distinguishing between algorithms. Some shortest path computations cannot be avoided, particularly those required to detect inconsistent values. However, some shortest path computations are useless, as they do not allow us to establish the inconsistency of any value. Precisely, if the shortest path computation from b In Corollary 6 does not lead to any deletion of values then this path computation is useless.

Table 1 compares the number of shortest path computations performed by Régin's algorithm and by our approach as a function of the number of landmarks allowed in. The number of shortest paths required to compute landmarks are included.

Table 2 shows the average number of useless shortest path computations for each dataset. We consider that shortest path computations for landmarks are always useless, so they are always included. That is why there are never 0 computations with landmarks.

Table 3 gives the time required by each method.

7.2 Results Analysis

Table 1 shows that our approach generally computes significantly fewer shortest paths than Régin's algorithm for all landmarks selection methods. For the TSP instances, we compute on average between 2 and 47 times fewer shortest paths than Régin's algorithm. The difference is significant for all instances except for the StockingCost instances with Regular H. It should be noted that our approach is always better or equivalent and allows us to detect quickly whether the constraint is arc consistent in certain cases.

In the best case, our approach does not compute any shortest paths other than those required to determine landmarks. Our approach can compute more shortest paths only when there is no inconsistent arc and the extra computation is due to the landmarks. The number of useless path computations is also reduced by our method (See Table 2).

For computation times, we find the same kind of results as before (See Table 3). The gain average factors evolve between 1 and 57.

Table 1 Establishment of the arc consistency of a costgcc constraint: average number of computed shortest paths depending on the number of landmarks and the landmark selection method.

	Régin	Landmark Number	C	O	C & O	Deg	R
TSP ($\leq$ 100 cities)	57.6	1	31.7	36.3	36.2	**27.7**	**27.7**
		2	35.3	39.9	39.8	32.5	29.5
		3	38	42.7	42.5	32.5	28.5
		4	41.6	46.3	46.1	32	30.1
		5+	44.8	50	50.2	32	32.2
TSP ($>$ 100 & $<$ 250 cities)	163.3	1	42.2	45	47.9	**40.5**	**40.5**
		2	44.4	47.4	46.3	41.6	41.6
		3	46	49.3	48.2	41.2	41.2
		4	48.6	51.9	50.8	42.3	42.3
		5+	50.2	54.1	52.2	43.1	43.3
TSP ($\geq$ 250 cities)	662.7	1	18.1	19.8	19.8	17.8	17.8
		2	18.5	21.4	21.4	18.1	18.1
		3	18.5	21	19.3	**16.3**	**16.2**
		4	18.8	21.6	19.9	16.4	**16.3**
		5+	19	21.8	20.1	16.7	16.4
StockingCost (Regular H)	**493.3**	1	496.9	497.3	496.9	495.3	495.3
		2	500.8	501.2	500.8	497.3	497.2
		3	504.7	505.1	504.7	499.2	499.1
		4	508.6	509	508.6	501.2	501
		5+	512.5	512.9	512.6	503.2	503
StockingCost (Big H)	493.3	1	4	4	4	**2**	**2**
		2	4	4	4	**2**	**2**
		3	4	4	4	**2**	**2**
		4	4	4	4	**2**	**2**
		5+	4	4	4	**2**	**2**
FJSSP (Regular H)	10.4	1	8.3	5.1	4.8	**2**	6.3
		2	8.3	5.1	4.8	**2**	5.3
		3	8.3	5.1	4.8	**2**	4.6
		4	8.3	5.1	4.8	**2**	4
		5+	8.3	5.1	4.8	**2**	4
FJSSP (Big H)	10.4	1	4.5	4.3	4.3	**2**	3.2
		2	4.5	4.3	4.3	**2**	2.8
		3	4.5	4.3	4.3	**2**	2.6
		4	2.9	4.3	4.3	**2**	2.4
		5+	2.9	4.3	4.3	**2**	2.4
CHILD (Regular H)	**108**	1	112	112	112	109	110
		2	116	116	116	111	112
		3	120	120	120	113	114
		4	124	124	124	115	116
		5+	128	128	128	117	118
CHILD (Big H)	108	1	4	4	4	**2**	**2**
		2	4	4	4	**2**	**2**
		3	4	4	4	**2**	**2**
		4	4	4	4	**2**	**2**
		5+	4	4	4	**2**	**2**

7.2.1 Landmark Number and Selection Method

We can see that the results do not change much as a function of the number of landmarks. The major part of problems have best or equivalent results with 4 landmarks, but the difference is minimal. When it is not mentioned 4 landmarks are used.

Table 2 Establishment of the arc consistency of a costgcc constraint: number of average shortest paths computed uselessly with 4 landmarks.

	Régin	C	O	C & O	Deg	R
TSP ($\leq$ 100 cities)	35.8	19.7	24.4	24.3	**8.3**	**8.2**
TSP ($>$ 100 & $<$ 250 cities)	131.1	16	19.7	18.6	**10.1**	**10.1**
TSP ($\geq$ 250 cities)	649.8	5.9	8.6	6.9	**3.4**	**3.3**
StockingCost (Regular H)	**0**	3.2	7.6	11.2	7.8	7.6
StockingCost (Big H)	493.3	4	4	4	**2**	**2**
FJSSP (Regular H)	**0**	8.3	5.1	4.8	4	4
FJSSP (Big H)	10.1	2.9	4.3	4.3	**2**	2.4
CHILD (Regular H)	**0**	16	16	16	8	8
CHILD (Big H)	108	4	4	4	**2**	**2**

Table 3 Establishment of the arc consistency of a costgcc constraint: computation times (in ms) and ratio. Experimentation with 4 landmarks.

		Régin	C	O	C & O	Deg	R
TSP ($\leq$ 100 cities)	Mean	7.3	5.9	6	6.6	5.7	**4.5**
	Median	3.4	3.6	4.4	4.1	3.6	**3.3**
	Ratio		1.2	1.2	1.1	1.3	**1.6**
TSP ($>$ 100 & $<$ 250 cities)	Mean	76.6	**29.8**	30.6	30.2	**28.6**	31.1
	Median	51.2	**14.3**	16	17	15.4	**14.3**
	Ratio		**2.6**	2.5	2.5	**2.7**	2.5
TSP ($\geq$ 250 cities)	Mean	12124.9	278.9	275.2	275.4	**213**	265
	Median	2310.2	126.8	117.7	90.6	89.1	**85.9**
	Ratio		43.5	44.1	44	**56.9**	45.8
StockingCost (Regular H)	Mean	603.83	**511.8**	617.9	626.2	580.3	639.4
	Median	585.7	553.3	186.9	186.4	248	**166.4**
	Ratio		**1.2**	1	1	1	0.9
StockingCost (Big H)	Mean	534.76	34.1	32.4	**31.6**	33.2	32.6
	Median	519.1	33.8	32.4	31.9	32.8	**30.1**
	Ratio		15.7	16.5	**16.9**	16	16.4
FJSSP (Regular H)	Mean	0.4	0.5	**0.3**	0.4	0.4	0.5
	Median	**0.1**	0.3	0.2	0.3	0.2	0.3
	Ratio		0.8	**1.7**	0.75	1	0.8
FJSSP (Big H)	Mean	0.4	0.4	**0.3**	**0.3**	**0.3**	**0.3**
	Median	**0.1**	0.2	0.2	0.2	0.2	0.2
	Ratio		1	**1.3**	**1.3**	**1.3**	**1.3**
CHILD (Regular H)	Time	65.1	69.2	**54.4**	67.6	75.9	65.4
	Ratio		0.9	**1.2**	1	0.8	1
CHILD (Big H)	Time	58.2	7	6.5	7.3	**6**	**6**
	Ratio		8.3	9	8	**9.7**	**9.7**

Two methods of landmark selection appear to be more effective in practice: the method based on maximum node degrees and the random node selection method. As there is little difference between these two methods, and the former is more robust than the latter, we recommend defining landmarks based on maximum degree nodes.

7.2.2 Impact of the practical improvement of Section 5

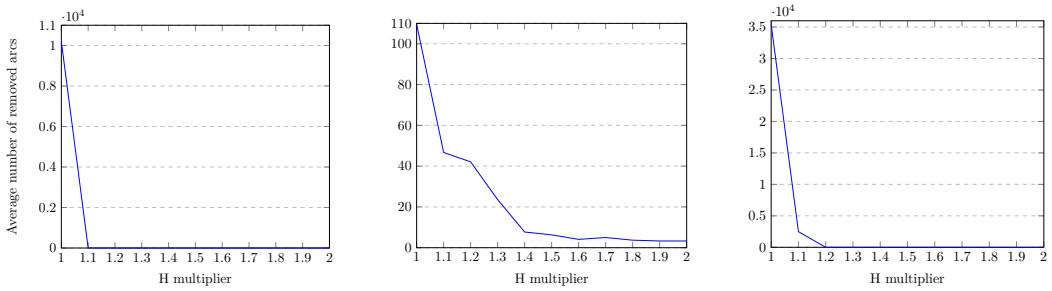
Thanks to this practical improvement all the authorized landmarks are not systematically used. This is clearly seen for StockingCost and CHILD instances with big H. The computation of a single landmark is sufficient to guarantee that all the values are consistent.

7.2.3 StockingCost, FJSSP and CHILD problems

For the StockingCost, FJSSP and CHILD problems, the results strongly depends on the value of H.

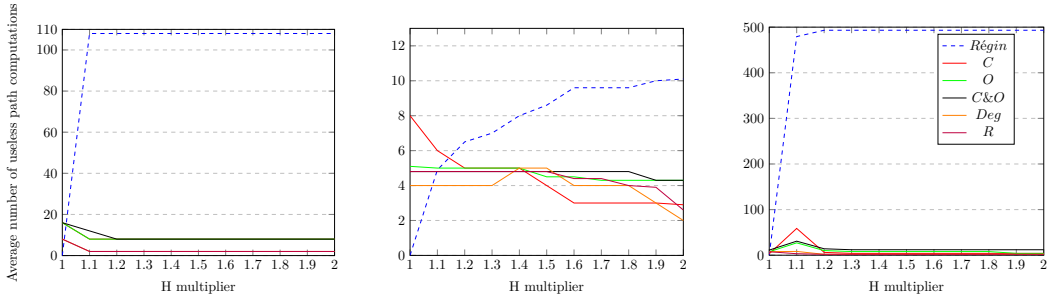
For the Regular H value the results are similar to those of Régin's algorithm. In these problems, Regular H is close to the optimal value of the min cost flow of the underlined costgcc. Thus, there is less margin and therefore more inconsistent values. FJSSP instances are also small and do not allow us to highlight the usefulness of landmarks. Indeed, in a small instance, computing a landmark gives us access to less information than in a large instance. In addition, for practical use, it is more interesting to save time on large instances since they take longer to resolve than on small instances which are already quick to resolve.

For a Big H value the landmark method clearly outperforms Régin's algorithm.



(a) CHILD. **(b)** FJSSP. **(c)** StockingCost.

Figure 4 Evolution of the average number of removed arcs for the CHILD, FJSSP and StockingCost instances in function of the multiplier of H.



(a) CHILD. **(b)** FJSSP. **(c)** StockingCost.

Figure 5 Evolution of the average number of useless path computations for CHILD, FJSSP and StockingCost instances in function of the multiplier of H. The experimentation involves 4 landmarks.

Figures 4 and 5 provide information on the relationship between H values and the number of useless path computations. The landmark approach performs very well as soon as the H value deviates a little from the optimal value, in other words, as soon as there is a little margin and therefore fewer inconsistent values.

7.2.4 TSP results

The improvement brought about by our approach for instances from the TSP problem are strong. This is mainly due to the relationship between the H value given by the TSP value and the underlying costgcc constraint. In the case of the TSP, the optimal value of the min

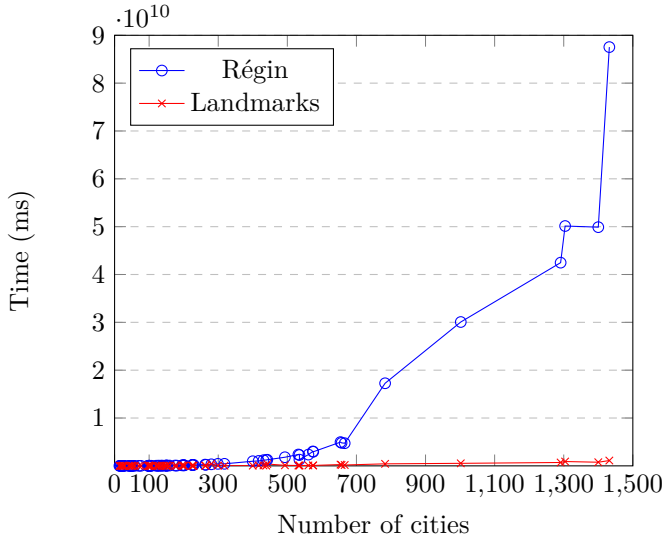


Figure 6 Evolution of the time in relation to the size of TSP instances. The landmarks are selected with the degree method and 4 landmarks. The blue plot with circles is the time of the Régin algorithm and the red with crosses is the time of the landmarks algorithm.

cost flow is lower than H, because the costgcc constraint only models a part of the problem and in fact represents a real relaxation. So even with an optimal H value for TSP, there is a margin for the costgcc constraint.

To better appreciate the performance of the landmarks, Figure 6 shows the evolution of time in milliseconds as a function of the size of the TSP dataset instances. The blue plot with circles shows the time taken by the Régin's algorithm, while the red plot with crosses shows the time taken by the algorithm using the maximum degrees and 4 landmarks. Clearly, the use of landmarks is drastically faster than the Régin's algorithm. The larger the instance, the more useful landmarks become.

Figure 7 shows the evolution of the speed-up ratio (Régin time/Landmarks time) on the instances of the TSP dataset. The landmark selection algorithm is based on maximum degrees with 4 landmarks. We can also see in this graph that the more data there is, the higher the gain factor. As mentioned above, this can be explained by the fact that in a large structure, the landmarks contain a lot of information compared with a smaller structure. In these experiments, note that if we omit the assigned variables there is only one strongly connected component in the value network. Overall, we find that our algorithm significantly speeds up the previous approach, up to about 80 times faster for large problems.

8 Conclusion

This paper proposes an efficient implementation of the arc consistency algorithm of the cardinality constraint with costs. This constraint is present in many industrial problems and the establishment of the arc consistency is often too slow to be used in practice, as it is based on finding the shortest paths from the assigned values. We introduce a new method that uses upper bounds on shortest paths based on triangular inequalities and landmarks. This approach avoids the computation of many shortest paths and improves the computation time of the arc consistency filtering algorithm. The larger the graph and the larger the margins, the greater the improvement will be. In addition, we have introduced a sufficient condition, which is quick to compute, for a costgcc constraint to be arc consistent.

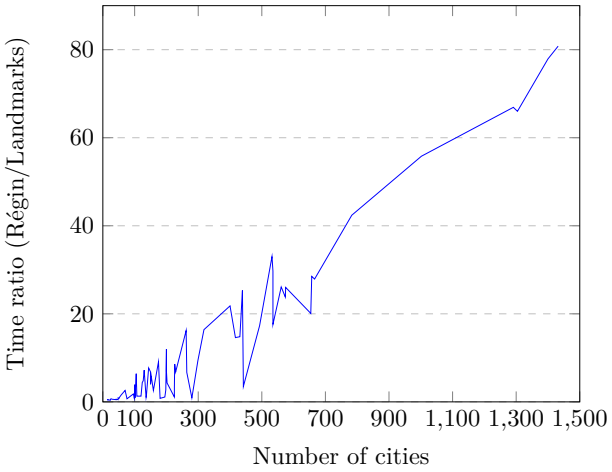


Figure 7 Evolution of the speedup ratio in relation to the size of TSP instances. The landmarks are selected with the degree method and 4 landmarks.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.
2. C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd, 1985.
3. G. Demassey, S.and Pesant and L.-M. Rousseau. A Cost-Regular Based Hybrid Column Generation Approach. *Constraints*, 11(4):315–333, December 2006. doi:10.1007/s10601-006-9003-7.
4. S. Ducomman, H. Cambazard, and B. Penz. Alternative Filtering for the Weighted Circuit Constraint: Comparing Lower Bounds for the TSP and Solving TSPTW. In *AAAI 2016*, Phoenix, United States, February 2016. URL: https://hal.science/hal-01420964.
5. R. Gerhard. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.
6. A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, USA, 2005. Society for Industrial and Applied Mathematics.
7. S. Gualandi and F. Malucelli. Constraint Programming-based Column Generation. *Annals of Operations Research*, 204(1):11–32, April 2013. doi:10.1007/s10479-012-1299-7.
8. V. R. Houndji, P. Schaus, and L. Wolsey. The item dependent stockingcost constraint. *Constraints*, 24(2):183–209, April 2019. doi:10.1007/s10601-018-9300-y.
9. N. Isoart. *The traveling salesman problem in constraint programming*. Theses, Université Côte d'Azur, November 2021. URL: https://theses.hal.science/tel-03554009.
10. E. Lawler. *Combinatorial optimization - networks and matroids*. Holt, Rinehart and Winston, New York, 1976.
11. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21:498–516, 1973. URL: https://api.semanticscholar.org/CorpusID:33245458.
12. P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Spracklen. Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251:35–61, 2017. doi:10.1016/j.artint.2017.07.001.
13. M. Pelleau, A. Miné, C. Truchet, and F. Benhamou. A constraint solver based on abstract domains. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 434–454, 2013. doi:10.1007/978-3-642-35873-9_26.

14 J.-C. Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3/4):387–405, July 2002. doi:10.1023/A:1020506526052.

15 J.-C. Régin. Filtering algorithm for constraints of difference in csps. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1, July 1994.

16 J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI'96*, pages 209–215, January 1996.

17 R. E. Tarjan. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, USA, 1983.

18 W.-J. Van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4):347–373, September 2006. doi:10.1007/s10732-006-6550-4.

19 S. Varone and C. Beffa. Dataset on a problem of assigning activities to children, with various optimization constraints. *Data in Brief*, 2019.

20 T. Weise. jsspinstancesandresults: Results, data, and instances of the job shop scheduling problem, 2019–2020. A GitHub repository with the common benchmark instances for the Job Shop Scheduling Problem as well as results from the literature, both in form of CSV files as well as R program code to access them. URL: https://github.com/thomasWeise/jsspInstancesAndResults.

21 X. Zhang, J. Gao, Y. Lv, and W. Zhang. Early and efficient identification of useless constraint propagation for alldifferent constraints. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1126–1133. International Joint Conferences on Artificial Intelligence Organization, July 2020. Main track. doi:10.24963/ijcai.2020/157.

22 X. Zhang, Q. Li, and W. Zhang. A fast algorithm for generalized arc consistency of the alldifferent constraint. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1398–1403. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi:10.24963/ijcai.2018/194.