The WeightedCircuitsLmax Constraint

Kim Rioux-Paradis[1] and Claude-Guy Quimper[1]

Université Laval
Québec City, Canada
kim.rioux-paradis.1@ulaval.ca
claude-guy.quimper@ift.ulaval.ca

Abstract. The travelling salesman problem is a well-known problem that can be generalized to the m-travelling salesmen problem with min-max objective. In this problem, each city must be visited by exactly one salesman, among m travelling salesmen. We want to minimize the longest circuit travelled by a salesman. This paper generalizes the Circuit and WeightedCircuit constraints and presents a new constraint that encodes m cycles all starting from the same city and whose lengths are bounded by a variable L_{max}. We propose two filtering algorithms, each based on a relaxation of the problem that uses the structure of the graph and the distances between each city. We show that this new constraint improves the solving time for the m travelling salesmen problem.

1 Introduction

Constraint programming offers a large number of constraints to model and to solve combinatorial problems [1]. These constraints serve two purposes: they facilitate the modelling of a problem and they offer strong filtering algorithms that reduce the search space. When solving an optimization problem with a Branch & Bound approach, computing a tight bound on the objective function is crucial to limit the size of the search tree. Global constraints can help compute the bound by filtering the objective variable with respect to a large number of variables in the problem.

We propose a new global constraint that helps to model and solve the m-travelling salesman problem with a min-max objective. This problem consists in planning the routes of m salesmen that start at the depot D, visit $n-1$ cities exactly once, and return to the depot. We want to minimize the longest route. While there exist several global constraints that can be used to encode this problem, few include an optimization criterion, and none minimize the length of the longest cycle. With the new global constraint WeightedCircuitsLmax, one can easily model the m-travelling salesman problem and compute tight bounds on the objective function.

The filtering algorithm we propose uses two relaxations both inspired by the 1-tree relaxation [2] used for the WeightedCircuit constraint. We name these relaxations the 1-forest relaxation and the cluster relaxation. For each relaxation, we develop a filtering algorithm that takes into account the minimization of the

longest cycle. We compare a model that uses this new constraint against a model that does not use it and we show its efficiency.

Section 2 describes the problem and the data structures used to design new filtering algorithms. Section 3 presents the new constraint WeightedCircuitsLmax. Section 4 and 5 each present a relaxation and a filtering algorithm. Section 6 explains how to detect situations where the constraint is consistent and filtering algorithms do not need to be executed. Finally, Section 7 presents the experimental results.

.

2 Background

We present the m-travelling salesman problem. We follow with the presentation of data structures, graph theory, and algorithms that we later use as tools to design our filtering algorithms. We also present global constraints useful to model the travelling salesman problem and its variants.

2.1 M-TSP

The m-travelling salesmen problem, denoted m-TPS, is a well-known generalization of the travelling salesman problem (TSP) [3]. The problem's input is a graph $G = (V, E, d)$ where V is a set of vertices (or cities), E a set of edges, and $d(i, j) = d(j, i)$ is a symmetric function that returns the distance between the vertices i and j. The city $n - 1$ is called the depot $D = n - 1$. The goal is to find m disjoint routes for the m salesmen leaving and returning to the depot. All cities have to be visited exactly once by only one salesman. The objective is to minimize the total distance. The min-max m-TSP [4,5,6,7] is the same problem but the objective is to minimize the longest route. L_z is the sum of the distance travelled by the z^{th} salesman and L_{max} the maximum of the L_z. Minimizing the longest circuit is often sufficient to balance the workload between each salesman.

There are different ways to solve a m-TSP exactly. Exact methods based on branch and bound [8,9] and integer programming are often used [10]. One can also reduce the m-TSP to a TSP problem with one salesman and use exact methods for this problem [11,12]. There are many heuristics that have been developed to efficiently obtain good solutions for the m-TSP problem, without providing guaranties about the optimality of the solution. We recommend a survey by Bektas [13] for an overview of these techniques.

2.2 The constraints Circuit and Cycles

Laurière [14] introduces the Circuit($[X_1, \ldots, X_n]$) constraint. This constraint is satisfied if the directed graph composed of the vertices $V = \{1, \ldots, n\}$ and the arcs $E = \{(i, X_i) \mid i \in V\}$ contains exactly one cycle. The variable X_i is the node that follows i on the cycle. This constraint models the Hamiltonian cycle problem. Let $G = \langle V, E \rangle$ be an undirected graph. For every node i, one defines

a variable X_i with a domain containing the nodes adjacent to i. The constraint Circuit($[X_1, \ldots, X_n]$) is satisfiable if and only if G contains a cycle visiting each node exactly once, i.e. a Hamiltonian cycle. Therefore, it is NP-Hard to perform a complete filtering on this constraint. However, many algorithms [15,16,17] perform a partial filtering in polynomial time.

Beldiceanu and Contejean [18] generalize the Circuit constraint into the Cycles($[N_1, \ldots, N_n], m$) constraint to impose exactly m cycles rather than one. The cycles must be disjoint, the salesmen do not leave from the same depot. They generalize Cycles further by proposing Cycles($[X_1, \ldots, X_n], m, [W_1, \ldots, W_n], \min, \max$) where W_i is a weight assigned to node i, not to an edge. The variables min and max are weights of the smallest and largest cycles. Further generalization can force nodes to belong to different cycles and control the length of individual cycles. These constraints are primilarly introduced as powerful modelling tools.

Benchimol et al. [19] present several filtering algorithms for the WeightedCircuit($[X_1, \ldots, X_n], d, L$) constraint. Focacci et al. [20,21] also study this constraint that is satisfied when the constraint Circuit($[X_1, \ldots, X_n]$) is satisfied and when the total weight of the selected edges is no more than L, i.e. $\sum_{i=1}^{n} w(X_i, i) \leq L$. They use different relaxations and reduced-cost based filtering algorithms to filter the constraint, including the 1-tree relaxation that we describe in Section 2.6.

Without introducing a new constraint, Pesant et al. [22] show how to combine constraint programming and heuristics to solve the travelling salesman problem with and without time windows.

2.3 Disjoints Sets

The algorithms presented in the next sections use the disjoint sets data structure. This data structure maintains a partition of the set $\{0, \ldots, n-1\}$ such that each subset is labelled with one of its elements called the representative. The function Find(i) returns the representative of the subset that contains i. The function Union(i,j) unites the subsets whose representatives are i and j and returns the representative of the united subset. Find(i) runs in $\Theta(\alpha(n))$ amortized time, where $\alpha(n)$ is Ackermann's inverse function. Union(i, j) executes in constant time [23].

2.4 Minimum Spanning Tree

A weighted tree $T = (V, E, w)$ is an undirected connected acyclic graph. Each pair of vertices is connected by exactly one path. The weight of a tree $w(T) = \sum_{e \in E} w(e)$ is the sum of the weight of its edges. A *spanning tree* of a graph $G = \langle V, G \rangle$ is a subset of edges from E that forms a tree covering all vertices in V. A *minimum spanning tree* $T(G)$ is a spanning tree of G whose weight $w(T(G))$ is minimal [24].

Kruskal's algorithm [25] finds such a tree in $\Theta(|E| \log |V|)$ time. It starts with an empty set of edges S. The algorithm goes through all edges in non-decreasing

order. An edge is added to S if it does not create a cycle among the selected edges in S. The disjoint sets are used to verify this condition. The algorithm maintains an invariant where the nodes of each tree form disjoint sets. The nodes that belong to the same sets are connected by a path of edges selected by Kruskal's algorithm. To test whether the addition of an edge (i, j) creates a cycle, the algorithm checks whether the nodes i and j belong to the same disjoint set. If not, the edge is selected and the disjoint sets that contain i and j are united.

Let $T(G)$ be the minimum spanning tree of the graph $G = \langle V, E \rangle$. Let $T_e(G)$ be the minimum spanning tree of G for which the presence of the edge e is imposed. The reduced cost of the edge e, denoted $\widetilde{w}(e)$, is the cost of using e in the spanning tree : $\widetilde{w}(e) = w(T_e(G)) - w(T(G))$. The reduced cost $\widetilde{w}(e)$ of an edge $e = (i, j) \in E$ can be computed by finding the edge s with the largest weight lying on the unique path between nodes i and j in $T(G)$. We obtain $\widetilde{w}(e) = w(e) - w(s)$ [26].

A *spanning forest* of a graph $G = \langle V, E \rangle$ is a collection of trees whose edges belong to E and whose nodes span V. The weight of a forest is the sum of the weights of its trees. A *minimum spanning forest* of m trees is a spanning forest of m trees whose weight is minimum. Kruskal's algorithm can be adapted to find a spanning forest. One simply needs to prematurely stop the algorithm after finding m trees, i.e. after $|V| - m$ unions.

2.5 Cartesian Tree

The Path Maximum Query problem is defined as follows: Given a weighted tree $T = (V, E, w)$ and two nodes $u, v \in V$, find the edge with the largest weight that lies on the unique path connecting u to v in T. This problem can be solved with a simple traversal of the tree in $\mathcal{O}(|E|)$ time. The offline version of the problem is defined as follows. Given a weighted tree $T = \langle V, E, w \rangle$ and a set of queries $Q = \{(u_1, v_1), \ldots, (u_{|Q|}, v_{|Q|})\}$, find for each query$(u_i, v_i) \in Q$ the edge with the largest weight that lies on the unique path connecting u_i to v_i in T. This problem can be solved in $\mathcal{O}(|E| + |Q|)$ time using a Cartesian tree as we explain below.

A Cartesian tree T_C of a tree $T = \langle V, E, w \rangle$ is a rooted binary, and possibly unbalanced, tree with $|V|$ leaves and $|E|$ inner nodes. It can be recursively constructed following Demaine et al. [27]. The Cartesian tree of a tree containing a single node and no edge ($|V| = 1$ and $|E| = 0$) is a node corresponding to the unique node in V. If the tree T has more than one node, the root of its Cartesian tree corresponds to the edge with the largest weight denoted $e_{\max} = \arg\max_{e \in E} w(e)$. The left and right children correspond to the Cartesian trees of each of the two trees in $E \setminus \{e_{\max}\}$. Finding the largest edge on a path between u_i and v_i in T is equivalent to finding the lowest common ancestor of u and v in the Cartesian tree T_C. The Cartesian tree can be created in $\mathcal{O}(|V|)$ time after sorting the edges in preprocessing [27]. Tarjan's off-line lowest common ancestor algorithm [23] takes as input the Cartesian tree T_C and the queries Q and returns the lowest common ancestor of each query in Q, i.e. the

edge with largest weight lying on the path from u_i to v_i in T for each i. When implemented with the disjoint set data structure by Gabow and Tarjan [28], the algorithm has a complexity of $\mathcal{O}(|V| + |Q|)$.

2.6 1-Tree relaxation

Held and Karp [2] introduce the 1-tree relaxation to solve the travelling salesman problem that Benchimol et al. [19] use to filter the WeightedCircuit constraint. The relaxation partitions the edges E into two disjoint subsets: the subset of edges connected to the depot D, denoted $E_D = \{(i,j) \in E \mid i = D \vee j = D\}$, and the other edges denoted $E_O = E \setminus E_D$. A solution to the travelling salesman problem is a cycle and therefore has two edges in E_D and the remaining edges in E_O form a simple path. In order to obtain a lower bound on the weight of this cycle, Held and Karp [2] select the two edges in E_D that have the smallest weights and compute the minimum spanning tree of the graph $G' = \langle V \setminus \{D\}, E_O \rangle$. The weight of the two selected edges in E_D plus the weight of the minimum spanning tree gives a lower bound on the distance travelled by the salesman. This relaxation is valid since a simple path is a tree and therefore, the minimum spanning tree's weight is no more than the simple path's weight.

3 Introducing WeightedCircuitsLmax

We introduce the constraint WeightedCircuitsLmax that encodes the m circuits of the salesman that start from the depot D, visit all cities once, and return to the depot. All circuits have a length bounded by L_{max}.

$$\begin{aligned} &\text{WeightedCircuitsLmax}([S_0, ..., S_{m-1}], \\ &[N_0, ..., N_{n-2}], d[0, ..., n-1][0, ..., n-1], L_{max}) \end{aligned} \tag{1}$$

The variable S_k is the first city visited by the salesman k. The variable N_i is the next city visited after city i. The symmetric matrix parameter d provides the distances between all pairs of cities. The variable L_{max} is an upper bound on the lengths of all circuits. The constraint WeightedCircuitsLmax is designed to be compatible with the Circuit constraint in order to take advantage of its filtering algorithms. For that reason, the value associated to the depot is duplicated m times. Each salesman returns to a different copy of the depot. The integers from 0 to $n-2$ represent the $n-1$ cities and the integers from $n-1$ to $n+m-2$ represent the depot. If $N_i \leq n-2$, the salesman visit city N_i after city i. If $n-1 \leq N_i$, the salesman returns to the depot after visiting city i. Consequently, we have $\text{dom}(S_k) \subseteq \{0, \ldots, n-2\}$ and $\text{dom}(N_i) \subseteq \{0, \ldots, n+m-2\}$.

While using the constraint WeightedCircuitsLmax one can post the constraint Circuit$([N_0, ..., N_{n-2}, S_0, ..., S_{m-1}])$ to complement the filtering of the WeightedCircuitsLmax constraint. The filtering algorithms we present in Sections 4 and 5 for the WeightedCircuitsLmax constraint are based on the cost L_{max}. On the other hand, the filtering algorithms of Circuit constraints are based on the structure of the graph. The filtering algorithms are complementary.

Circuit is a special case of WeightedCircuitsLmax where $m = 1$ and $L_{max} = \infty$. Enforcing domain consistency on circuit is NP-Hard [14]. Therefore, enforcing domain consistency on WeightedCircuitsLmax is NP-Hard.

4 1-forest relaxation

We now describe a relaxation of the WeightedCircuitsLmax constraint. We introduce two relaxations to the WeightedCircuitsLmax constraint. These relaxations are used to compute a bound on the length of the longest cycle and to filter the starting variables S_i and the next variables N_i. As seen in the previous section, it is possible to use the WeightedCircuitsLmax constraint in conjunction with the Circuit constraint. The filtering algorithm that we present is added to the filtering that is already done by the Circuit constraint.

4.1 Relaxation

The domains of the variables S_i and N_i encode the following graph that we denote G. Let the vertices be $V = \{0, \ldots, n-1\}$ where the depot is the node $D = n-1$. Let the edges $E = E_D \cup E_O$ be partitioned into two sets. The set of edges adjacent to the depot $E_D = \{(D, i) \mid \max(\text{dom}(N_i)) \geq n-1\} \cup \bigcup_{k=0}^{m-1} \text{dom}(S_k)$ and the edges that are not adjacent to the depot $E_O = \{(i, j) \mid i < j \wedge (i \in \text{dom}(N_j) \vee j \in \text{dom}(N_i))\}$. The weight of an edge is given by the distance matrix $w(i, j) = d[i][j]$.

We generalize the 1-tree relaxation to handle multiple cycles passing by a unique depot D. We call this generalization the 1-forest. Rather than choosing 2 edges in E_D, we choose the m shortest edges $a_0, \ldots, a_{m-1} \in E_D$. We do not choose the $2m$ shortest edges because there could be a salesman that goes back and forth to a node v using twice the edge $e_{v,D}$. Choosing $2m$ different edges is therefore not a valid lower bound. Rather than computing the minimum spanning tree with the edges in E_O, we compute the minimum spanning forest of m trees $T_0, \ldots, T_{m-1}$ using the edges in E_O. Then, L_{max} has to be greater than or equal to the average cost of the trees. The lower bound on L_{max} is given by $c([a_0, \ldots, a_{m-1}], [T_0, ..., T_{m-1}])$.

$$c([a_0, \ldots, a_{m-1}], [T_0, ..., T_{m-1}]) = \frac{1}{m}\left(2\sum_{i=0}^{m-1} w(a_i) + \sum_{i=0}^{m-1} w(T_i)\right) \tag{2}$$

We show the validity of this relaxation.

Theorem 1. *$c([a_0, \ldots, a_{m-1}], [T_0, ..., T_{m-1}])$ is a lower bound on L_{max} for the constraint* WeightedCircuitsLmax$([S_0, ..., S_{m-1}], [N_1, \ldots, N_{n-1}], d[0, \ldots, n-1][0, \ldots, n-1], L_{max})$.

Proof. Consider a solution to the constraint where the length of the longest circuit is minimized. Let $E_D^S \subseteq E_D$ be the edges of the circuits connected to the depot and let $E_O^S \subseteq E_O$ be the other edges.

There are 2 edges by circuit going or returning to the depot (with the possibility of duplicates for salesmen visiting a single city). Therefore, there are $2m$ edges in E_D^S, counting duplicates. A lower bound for the cost of the edges in E_D^S is twice the cost of the m shortest edges in E_D.

The edges in E_O^S form a forest of m trees. Hence, a lower bound of the cost of the edges in E_O^S is the cost of a minimum forest of m trees T_i.

$$\begin{aligned} c([a_0,\ldots,a_{m-1}],[T_0,...,T_{m-1}]) &= \frac{1}{m}\left(2\sum_{i=0}^{m-1} w(a_i) + \sum_{i=0}^{m-1} w(T_i)\right) \\ &\leq \frac{1}{m}\left(\sum_{e_{D,i}\in E_D^S} w(e_{D,i}) + \sum_{e_i,j\in E_O^S} w(e_{i,j})\right) \end{aligned}$$

Since the average cost per salesman is smaller than or equal to the maximum cost, we obtain: $c([a_0,\ldots,a_{m-1}],[T_0,...,T_{m-1}]) \leq L_{max}$. □

The algorithm to compute $c([a_0,\ldots,a_{m-1}],[T_0,...,T_{m-1}])$ works as follows. First, we pick the m smallest edges in E_D. Then, we compute a forest of m trees with Kruskal's algorithm in $\mathcal{O}(|E|\log|V|)$ time.

4.2 Filtering the edges in E_D

We want to filter the edges $e = (D, i)$ in E_D that are not selected by the relaxation. We want to know whether $c([a_0,\ldots,a_{m-2},e],[T_0,...,T_{m-1}])$ is greater than L_{max}. If it is the case, then e cannot be in the solution because the cost of using this edge is too large. The filtering rule (3) removes $i+m$ from the domain of all starting time variable S_k and removes all values associated to the depot from the domain of N_i.

$$\begin{aligned} c([a_0,\ldots,a_{m-1},],[T_0,...,T_{m-1}]) + \frac{1}{m}(w(e) - w(a_{m-1})) > \max(\mathrm{dom}(L_{max})) \\ \implies S_k \neq i \wedge N_i < n-1 \quad \forall\, k \in \{0,\ldots,m-1\} \end{aligned} \tag{3}$$

4.3 Filtering the edges in E_O

We want to decide whether there exists a support for the edge $e = (i, j)$ in E_O. In other words, we want to find a forest of m trees $T'_0,\ldots,T'_{m-1}$ that contains e such that $c([a_0,\ldots,a_{m-1},],[T'_0,...,T'_{m-1}])$ is no greater than $\max(\mathrm{dom}(L_{max}))$. If e belongs to the trees $T_0,\ldots,T_{m-1}$ computed in Section 4.1, e has as support and should not be filtered. Otherwise, there are two possible scenarios: the nodes i and j belong to the same tree in $T_0,\ldots,T_{m-1}$ or they do not.

Same tree T_β Given the edge $e = (i, j)$, if nodes i and j belong to a tree T_β, the cost of adding the edge e is equal to its reduced cost $\widetilde{w}(e)$ (see Section 2.4), i.e. the weight $w(e)$ minus the largest weight of an edge lying on the unique path in T_β between i and j. If $c([a_0, \ldots, a_{m-1},], [T_0, ..., T_{m-1}]) + \widetilde{w}(e)$ is greater than $\max(\text{dom}(L_{max}))$, then edge e must be filtered out from the graph, i.e. i is removed from the domain of N_j and j is removed from the domain of N_i.

To efficiently compute the reduced costs, we construct a Cartesian tree (see Section 2.5) for each tree in $T_0, \ldots, T_{m-1}$ in $\mathcal{O}(|V|)$ time. For each edge $e = (i, j)$ such that i and j belong to the same tree, we create a query to find the edge with the largest weight between i and j in the tree. Tarjan's off-line lowest common ancestor algorithm [23] answers, in batches, all queries in time $\mathcal{O}(|V| + Q)$ where Q is the number of queries. For each edge, we compute the reduced cost and check whether filtering is needed.

Different trees T_ϵ, T_δ If the nodes i and j of the edge $e = (i, j)$ do not belong to the same tree, adding the edge e to the trees $T_0, \ldots, T_{m-1}$ connects two trees together. In order, to maintain the number of trees to m, one needs to remove an edge from any tree. To minimize the weight of the trees, the edge e' we remove must be the one with the largest weight, i.e. the last edge selected by Kruskal's algorithm. We obtain the reduced cost $\widetilde{w}(e) = w(e) - w(e')$. Using this reduced cost, we filter the variables N_i as we did when the nodes i and j belong to the same tree.

5 Clusters relaxation

The 1-forest relaxation is fast to compute, but its bound is not always tight. Counting twice the m shortest edges is less effective than counting the $2m$ edges that could be in the circuits. Moreover, for $m = 2$ salesmen, if the solution has a long and a short circuit, the bound lies between the length of both circuits. In that case, the lower bound for the longest circuit is not tight. We refine the 1-forest relaxation to better capture the structure of the graph and to obtain a tight bound on the longest cycle when all variables are assigned.

5.1 Relaxation

We consider the m trees $\{T_0, \ldots, T_{m-1}\}$ as computed in the 1-forest relaxation. Let $C_1, \ldots, C_r$ be a partition of the trees into clusters such that the trees that belong to the same cluster are connected with each other with edges in E_O and are not connected to the trees from the other clusters. Figure 1 shows an example of three trees partitioned into two clusters. Note that the number of clusters r is between 1 and m. In a solution, cities visited by a salesman necessarily belong to the same cluster. We compute a lower bound on L_{max} for each cluster and keep the tightest bound.

We choose two edges $e_1^{C_\alpha}$ and $e_2^{C_\alpha}$ in E_D for each cluster C_α. If the cluster contains a single node v, we choose twice the edge $e_{v,D}$, i.e. $e_1^{C_\alpha} = e_2^{C_\alpha} = e_{v,D}$. If

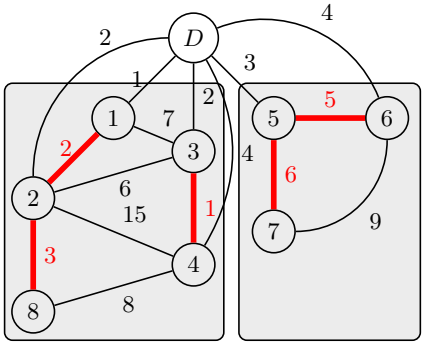

Fig. 1: Three trees grouped into two clusters. Edges that belong to a tree are in bold.

the cluster contains more than one node, we choose the two shortest edges that connect the depot to a node in the cluster. The weight of a cluster C_α, denoted $w(C_\alpha)$, is the weight of the two chosen edges $e_1^{C_\alpha}$ and $e_2^{C_\alpha}$ and the weight of the trees in the cluster.

$$w(C_\alpha) = \sum_{T_\beta \in C_\alpha} w(T_\beta) + w(e_1^{C_\alpha}) + w(e_2^{C_\alpha})$$

The nodes that belong to a cluster can be visited by a maximum of $\lambda = m - r + 1$ salesmen. In such a case, the average length of a circuit is given by the weight of the cluster divided by λ and it constitutes a valid lower bound of L_{max}.

$$\mu(\{C_1, \ldots, C_r\}) = \frac{1}{\lambda} \max_{C_\alpha} w(C_\alpha) \tag{4}$$

Theorem 2. *$\mu(\{C_1, \ldots, C_r\})$ is a lower bound for L_{max}.*

Proof. Consider a solution to the constraint where the length of the longest circuit is minimized. Let $E_D^S \subseteq E_D$ be the edges of the circuits connected to the depot and let $E_O^S \subseteq E_O$ be the other edges.

$$\mu(\{C_1, \ldots, C_r\}) = \frac{1}{\lambda} \max_{C_\alpha} \left(\sum_{T_\beta \in C_\alpha} w(T_\beta) + w(e_1^{C_\alpha}) + w(e_2^{C_\alpha}) \right) \tag{5}$$

$$\leq \max_{C_\alpha} \frac{1}{\lambda} \left(\sum_{\substack{e_{i,j} \in E_O^S \\ i \in C_\alpha}} w(e_{i,j}) + \sum_{\substack{e_{i,D} \in E_D^S \\ i \in C_\alpha}} w(e_{i,D}) \right) \tag{6}$$

Algorithm 1: ComputeClusters(G,m)

Data: $G = (V, E)$, m, the number of salesmen
Result: num_nodes_Eo, the number of nodes by clusters, max_edge_Eo, the longest edge of each cluster, weight_Eothe weight of each cluster
begin
  trees $\leftarrow$ $DisjointsSets(|V|)$
  clusters $\leftarrow$ $DisjointsSets(|V|)$
  num_clusters $\leftarrow |V|$
  for $i = 0..|V| - 1$ **do**
    weight_Eo[i] $\leftarrow 0$
    max_edge_Eo[i] $\leftarrow 0$
    num_nodes_Eo[i] $\leftarrow 1$
  for $e = (i, j) \in E$ *in non-decreasing order of weight* **do**
    $v_i \leftarrow$ clusters.FIND(i)
    $v_j \leftarrow$ clusters.FIND(j)
    if $v_i \neq v_j$ **then**
      if *num_clusters* $< |V| - m$ **then**
        $v_f \leftarrow$ trees.UNION(v_i, v_j)
      $v_f \leftarrow$ clusters.UNION(v_1, v_2)
      num_clusters $\leftarrow$ num_clusters $- 1$
1   weight_Eo[v_f] $\leftarrow$ weight_Eo[v_1] + weight_Eo[v_2]
2   max_edge_Eo[v_f] $\leftarrow w(e)$
      num_nodes_Eo[v_f] $\leftarrow$ num_nodes_Eo[v_1] + num_nodes_Eo[v_2]
  return *weight_Eo,max_edge_Eo, num_nodes_Eo*

Let c denote the longest circuit. Since the longest circuit is longer than or equal to the average length circuit, we obtain:

$$\leq \sum_{\substack{e_{i,j} \in E_O^S \\ i \in c}} w(e_{i,j}) + \sum_{\substack{e_{i,D} \in E_D^S \\ i \in c}} w(e_{i,D}) \tag{7}$$

$$\leq L_{max} \tag{8}$$

□

To compute the lower bound $\mu(\{C_1, \ldots, C_r\})$, we need to compute the clusters and select edges from E_O. To do so, we adapt Kruskal's algorithm as shown in Algorithm 1. Kruskal already uses a disjoint set data structure where the nodes of a tree are grouped into a set. We add another disjoint set data structure where the nodes of a cluster are grouped in a set. We process the edges in non-decreasing order of weight. When processing the edge (i, j), if i and j belong to two distinct clusters, we unite these clusters to form only one. The algorithm also merges the tree that contains i with the tree that contains j, but only if there are more than $|V| - m$ trees in the current forest. While we create each set, we keep three vectors that keep track of: the longest edge of each cluster, the weight of each cluster, and the number of nodes in each cluster. As for Kruskal's

Algorithm 2: ComputeEdgesFromE_D(E_D,num_nodes_Eo)

```
Data: E_D, num_nodes_Eo
Result: max_weight_Ed, weight_Ed, num_edges_Ed
for i = 1..n − 1 do
    max_weight_Ed[i] ← 0
    weight_Ed[i] ← 0
    num_edges_Ed[i] ← 0
for (i, D) ∈ E_D in non-decreasing order of weigth do
    v_i ← clusters.FIND(i)
    if num_edges_Ed[v_i] < 2 then
        weight_Ed[v_i] ← weight_Ed[v_i] + w(e)
        num_edges_Ed[v_i] ← num_edges_Ed[v_i] + 1
        max_weight_Ed[v_i] ← w(e)
for i = 0..n − 1 do
    rep ←clusters.FIND(i)
    if rep = i then
        if num_edges_Ed[i] = 1 ∧ num_nodes_Eo[i] > 1 then
            weight_Ed[i] ← 2 × weight_Ed[i]
        else if num_edges_Ed[i] < 2 then Fail
    return max_weight_Ed, weight_Ed
```

algorithm, the running time complexity is dominated by sorting the edges by weight which is done in $\mathcal{O}(|E| \log |V|)$.

Using the clusters computed by Algorithm 1, Algorithm 2 selects the edges from E_D by processing the edges in non-decreasing order of weight. When processing the edge (i, D), the algorithm finds the cluster that contains the node i. It selects the edge (i, D) only if fewer than two edges were selected for the cluster that contains i. Lines 1 and 2 update the sum of the weights of the selected edges for that cluster and the largest edge selected for the cluster. This information will be used later in the filtering algorithm. The second for loop checks whether there are clusters linked to the depot with a single edge. If it is the case and the cluster contains only one node, we make that edge count for double. If the cluster has more than one node, the constraint is unsatisfiable.

To compute a lower bound on L_{max}, we go through each cluster C. In (4), the summation is given by the entry of the vector weight_Eo corresponding to the cluster C_α as computed by Algorithm 1. The weight of the edges $w(e_1^{C_\alpha})+w(e_2^{C_\alpha})$ is given by the entry in the vector weight_Ed corresponding to the cluster C_α.

Overall, the cluster relaxation complements the 1-forest relaxation as follows. At the top of the search tree, when variable domains contain many values, this cluster relaxation is not as tight as the 1-forest relaxation. There are very few clusters and the cluster relaxation only selects two edges per cluster in the set E_D while the 1-forest relaxation selects m edges no matter how many clusters there are. However, as the search progresses down the search tree, there are fewer values in the domains and more clusters. Computing the maximum cycle per cluster becomes more advantageous than computing the average tree of the

1-forest relaxation. The cluster relaxation provides an exact bound when all variables are instantiated, which is not the case for the 1-forest relaxation.

5.2 Filtering the edges in E_D

We filter the edges in E_D based on the cluster relaxation as follows. Let $C_{\alpha(i)}$ be the cluster that contains the node i. For each edge $e = (D, i) \in E_D$, we check whether $\frac{1}{\lambda}\left(w(C_\alpha) - e_2^{C_{\alpha(i)}} + w(e)\right) \leq L_{max}$. In other words, we check whether substituting the edge $e_2^{C_{\alpha(i)}}$ by e induces a cost that is still below the desired threshold. If not, we remove i from the domain of S_k for all $k \in \{0, \ldots, m-1\}$ and we remove D from the domain of N_i.

5.3 Filtering the edges in E_O

As for the 1-forest relaxation, we filter edges in E_O differently depending they connect two nodes of the same tree or from different trees.

Same tree: Consider an edge $e = (i, j)$ such that i and j belong to the same tree and the same cluster that we denote $C_{\alpha(i)}$. We compute the reduced cost $\widetilde{w}(e)$, using a Cartesian tree, exactly as we do for the 1-forest relaxation in Section 4.3. If the inequality $w(C_{\alpha(i)}) + \widetilde{w}(e) \leq \max(\text{dom}(L_{max}))$ does not hold, we remove i from the domain of N_j and remove j from the domain of N_i.

Different trees: Consider an edge $e = (i, j)$ such that i and j do not belong to the same tree. However, by definition of a cluster, i and j belong to the same cluster that we denote $C_{\alpha(i)}$. Let e' be the edge with the largest weight in a tree of the cluster $C_{\alpha(i)}$, i.e. the last edge selected by Algorithm 1. We can substitute e' by e without changing the number of trees. The reduced cost of edge e is $\widetilde{w}(e) = w(e) - w(e')$. If $w(C_{\alpha(i)}) + \widetilde{w}(e) > \max(\text{dom}(L_{max}))$, we remove i from the domain of N_j and remove j from the domain of N_i.

6 Special Filtering Cases

There exist conditions when the filtering algorithm, whether it is based on the 1-forest or the cluster relaxation, does not do any pruning. Some of these conditions are easy to detect and can prevent useless executions of the filtering algorithms.

We consider two consecutive executions of the filtering algorithm. The second execution is either triggered by the instantiation of a variable or by constraint propagation. We check which values are removed from the domains between both executions. The removal of these values can trigger further filtering in two situations:

1. The upper bound of $\text{dom}(L_{max})$ is filtered;
2. An edge selected by the 1-forest or the cluster relaxation is filtered;

$$
\begin{aligned}
&\min L_{max} \\
&\text{s.t.} \\
&S_v \in \{0,\ldots,n-2\} && \forall\ v\ \in \{0,\ldots,m-1\} \\
&N_i \in \{0,\ldots,n+m-1\} \setminus \{i\} && \forall i \in \{1,\ldots,n\} \\
&\textsc{circuit}(N_0,\ldots,N_{n-1},S_0,\ldots,S_{m-1}) \\
&D[i] = 0 && \forall\ i \in \{n-1,\ldots,n+m-2\} \\
&D[i] = d'[n-1][S_{i-m-n+1}] + D[S_{i-m-n+1}] && \forall\ i \in \{n+m-1,\ldots,n+2m-2\} \\
&D[i] = d'[i][N_i] + D[N_i] && \forall\ i \in \{0,\ldots,n-1\} \\
&L_{max} \geq D[i] && \forall\ i \in\ \{n+m-1,\ldots,n+2m-2\}
\end{aligned}
$$

Fig. 2: Model 1 for the m-TSP.

In situation 1, the edges selected by the 1-forest and cluster relaxations remain unchanged. However, it is possible that the reduced cost of some edges are too large for the new bound on L_{max} and that these edges need to be filtered. In such a case, we do not need to recompute the relaxation nor the Cartesian trees, but we need to check whether the reduced cost of each edge is too high.

In situation 2, the trees and the clusters must be recomputed and the filtering algorithm needs to be executed entirely. If neither situation 1 nor 2 occurs, for instance if only an edge in E_O that does not belong to a tree is filtered, no filtering needs to be done and the algorithm does not need to be executed.

7 Experiments

We solve the m-TSP problem as defined in Section 2.1. We compare two different models in our experimentation. The model 1 is presented in Figure 2. The variable S_v indicates the starting address for the salesman v. The variable N_v indicates the address visited after address v. The integers between 0 and $n-2$ represent the addresses while the integers from $n-1$ to $n+m-2$ represent the arrival at the depot. The variable $D[v]$ encodes the remaining distance that a salesman needs to travel to reach the depot from address v. The constraint ELEMENT connects the distance variables $D[v]$ with the next variables N_v. The model 2 is based on model 1 that we augment with the constraint $\textsc{WeightedCircuitsLmax}(S_0,\ldots,S_{m-1},N_1,\ldots,N_n,d',L_{max})$. In the model, we make sure that the distance matrix can report the distances with the duplicates of the depot. We define the matrix $d'[i][j] = d[\min(i,n-1)][\min(j,n-1)]$.

Both models use the CIRCUIT constraint to exploit the graph's structure and to achieve a strong *strucural filtering*. In addition, Model 2 uses the WEIGHTEDCIRCUITSLMAX constraint to perform an *optimality filtering* that is based on the bound of the objective function. Our experiments aim at showing the advantage the new algorithms offer by performing optimality filtering.

We use instances from the m-TSP benchmark developed by Necula et al. [29]. We also generate instances of $\{10, 20, 50\}$ addresses in Quebec City with uni-

formly distributed longitude in [-71.20,-71.51] and latitude in [46.74, 46.95] with the shortest driving time as the distance between the addresses. Experiments are run on a MacBook Pro with a 2.7 GHz Intel Core i5 processor and 8 Gb of memory with the solver Choco 4.0.6 compiled with Java 8. We select the variable N_i and assign it to value j such that the distance $d(i, j)$ is minimal. However, other heuristics could have been used [30]. We report the best solution found after a 10-minute timeout.

7.1 Result and discussion

The first experiment aims to determine which relaxation should be used: the 1-forest relaxation, the cluster relaxation, or both. We solved random instances with 10 addresses with 1, 2, and 3 salesmen using model 2. Figure 3a shows that the cluster relaxation is better than the 1-forest relaxation. However, all instances except one were solved faster when combining both relaxations. For this reason, for model 2, we combine both relaxations for the rest of the experiments.

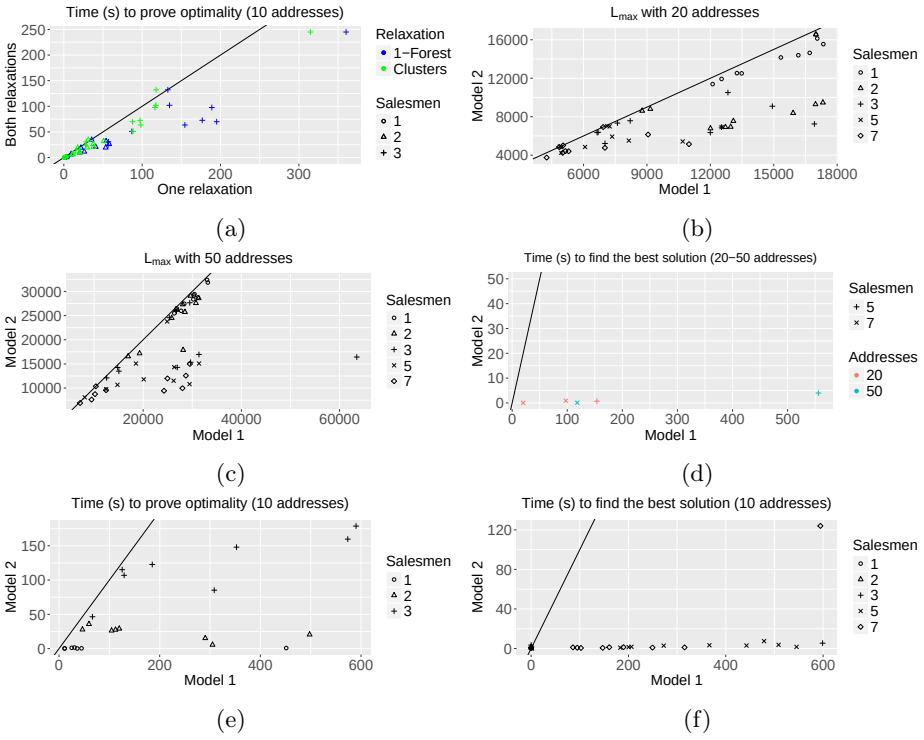

Fig. 3: Comparison of model 1 (without the WeightedCircuitsLmax) and model 2 (with WeightedCircuitsLmax) for instances with random addresses in Quebec City.

We compare model 1 against model 2. Figure 3b and Figure 3c present the objective function obtained by both models after 10 minutes with instances of 20

and 50 random addresses. Since we minimize, a point under the identity function indicates that model 2 finds a better solution. No solutions were proved optimal. However, we clearly see that model 2 finds better solutions and the gap increases as the number of salemen increase. As shown on Figure 3d, for instances where solutions of equivalent quality are returned by both models, model 2 finds the solution faster in most of the cases.

Some instances with 10 addresses are solved to optimality. Figure 3e reports the solving times for these instances and model 2 is clearly faster. For unsolved instances, both models return the same solutions (without proving their optimality). Figure 3f shows that model 2 returns the solution instantaneously.

Table 1 compares models 1 and 2 using a standard benchmark from [29]. Model 2 either finds a better solution or finds an equivalent solution faster for all instances.

			Model 1		Model 2	
Instances	n	m	L_{max}	Time last solution	L_{max}	Time last solution
Eil51	51	1	2307	6.0	2307	**1.5**
	51	2	1183	13.9	1183	**1.4**
	51	3	1906	0.5	**1183**	0.1
	51	5	638	0.1	**631**	211.2
	51	7	486	591.2	**457**	130
Berlin52	52	1	41534	3.1	**41276**	175.3
	52	2	32316	599.6	**20979**	164.4
	52	3	14629	599.3	**14457**	40.6
	52	5	14543	599.4	**13986**	42.5
	52	7	9668	347.4	**9668**	0.5
Eil76	76	1	3459	497.1	**3458**	307.0
	76	2	3284	1.8	**3278**	466.7
	76	3	2989	561.4	**2988**	375.2
	76	5	847	585.7	**730**	578.9
	76	7	651	34.9	651	**0.3**
Rat99	99	1	12093	68.8	**12084**	213.9
	99	2	11946	69.1	**11937**	343.4
	99	3	11881	78.2	**11872**	298.6
	99	5	11863	64.9	**11854**	372.9
	99	7	11813	108.4	**11804**	325.9

Table 1: Result for TSPLIB instances

8 Conclusion

We presented a constraint that models the m-travelling salesmen problem. We proposed two complementary filtering algorithms based on two relaxations. Experiments show that these filtering algorithms improve the solving times and the quality of the solutions. In future work, inspired from [19], we would like to use additive bounding to reinforce the relaxations. We also want to consider the number of salesman m as a variable instead of a parameter.

References

1. Beldiceanu, N., Carlsson, M., Rampon, J.: Global constraint catalog, 2nd edition (revision a). Technical Report 03, SICS (2012)
2. Held, M., Karp, R.: The traveling-salesman problem and minimum spanning trees. Operations Research **18**(6) (1970) 1138–1162
3. Laporte, G., Nobert, Y.: A cutting planes algorithm for the m-salesmen problem. Journal of the Operational Research Society (1980) 1017–1023
4. M.França, P., Gendreau, M., Laportt, G., Müller, F.M.: The m-traveling salesman problem with minmax objective. Transportation Science **29**(3) (1995) 267–275
5. Necula, R., Breaban, M., Raschip, M.: Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In: Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on, IEEE (2015) 873–880
6. Narasimha, K.V., Kivelevitch, E., Sharma, B., Kumar, M.: An ant colony optimization technique for solving min–max multi-depot vehicle routing problem. Swarm and Evolutionary Computation **13** (2013) 63–73
7. Somhom, S., Modares, A., Enkawa, T.: Competition-based neural network for the multiple travelling salesmen problem with minmax objective. Computers & Operations Research **26**(4) (1999) 395–407
8. Ali, A.I., Kennington, J.L.: The asymmetric m-travelling salesmen problem: A duality based branch-and-bound algorithm. Discrete Applied Mathematics **13**(2-3) (1986) 259–276
9. Gromicho, J., Paixão, J., Bronco, I.: Exact solution of multiple traveling salesman problems. In: Combinatorial Optimization. Springer (1992) 291–292
10. Kara, I., Bektas, T.: Integer linear programming formulations of multiple salesman problems and its variations. European Journal of Operational Research **174**(3) (2006) 1449–1458
11. Rao, M.R.: A note on the multiple traveling salesmen problem. Operations Research **28**(3-part-i) (1980) 628–632
12. Jonker, R., Volgenant, T.: An improved transformation of the symmetric multiple traveling salesman problem. Operations Research **36**(1) (1988) 163–167
13. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega **34**(3) (2006) 209–219
14. Lauriere, J.L.: A language and a program for stating and solving combinatorial problems. Artificial intelligence **10**(1) (1978) 29–127
15. Caseau, Y., Laburthe, F.: Solving small tsps with constraints. In: Proceedings of the 14th International Conference on Logic Programming (ICLP 1997). 316—-330
16. Kaya, L.G., Hooker, J.N.: A filter for the circuit constraint. In: In Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006). (2006) 706–710
17. Fages, J., Lorca, X.: Improving the asymmetric tsp by considering graph structure. Technical Report 1206.3437, arxiv (2012)
18. Beldiceanu, N., Contejean, E.: Introducing global constraints in chip. Mathematical and computer Modelling **20**(12) (1994) 97–123
19. Benchimol, P., Hoeve, W.J.V., Régin, J.C., L.-M. Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**(3) (2012) 205–233
20. Focacci, F., Lodi, A., Milano, M.: Embedding relaxations in global constraints for solving tsp and tsptw. Annals of Mathematics and Artificial Intelligence **34**(4) (2002) 291–311

21. Focacci, F., Lodi, A., Milano, M.: A hybrid exact algorithm for the tsptw. INFORMS Journal on Computing **14**(4) (2002) 403–417
22. Pesant, G., Gendreaul, M., Rousseau, J.M.: Genius-cp: A generic single-vehicle routing algorithm. In: In Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP 1997). (1997) 420–434
23. Tarjan, R.E.: Applications of path compression on balanced trees. Journal of the ACM (JACM) **26**(4) (1979) 690–715
24. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing **7**(1) (1985) 43–57
25. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society **7**(1) (1956) 48–50
26. Chin, F., Houck, D.: Algorithms for updating minimal spanning trees. Journal of Computer and System Sciences **16**(3) (1978) 333–344
27. Demaine, E.D., Landau, G.M., Weimann, O.: On cartesian trees and range minimum queries. In: International Colloquium on Automata, Languages, and Programming, Springer (2009) 341–353
28. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. In: Proceedings of the 15th annual ACM symposium on Theory of computing. (1983) 246–251
29. Necula, R., Breaban, M., Raschip, M.: Performance evaluation of ant colony systems for the single-depot multiple traveling salesman problem. In: International Conference on Hybrid Artificial Intelligence Systems, Springer (2015) 257–268
30. Fages, J.G., Prud'Homme, C.: Making the first solution good! In: ICTAI 2017 29th IEEE International Conference on Tools with Artificial Intelligence. (2017)