A Linear Time Algorithm for the k-Cutset Constraint

Nicolas Isoart
Université Côte d'Azur, Nice, France

Jean-Charles Régin
Université Côte d'Azur, Nice, France

Abstract

In CP, the most efficient model solving the TSP is the Weighted Circuit Constraint (WCC) combined with the k-cutset constraint. The WCC is mainly based on the edges cost of a given graph whereas the k-cutset constraint is a structural constraint. Specifically, for each cutset in a graph, the k-cutset constraint imposes that the size of the cutset is greater than or equal to two. In addition, any solution contains an even number of elements from this cutset. Isoart and Régin introduced an algorithm for this constraint. Unfortunately, their approach leads to a time complexity growing with the size of the considered cutsets, i.e. with k. Thus, they introduced an algorithm with a quadratic complexity dealing with k lower or equal to three. In this paper, we introduce a linear time algorithm for any k based on a DFS checking the consistency of this constraint and performing its filtering. Experimental results show that the size of most of the k-cutsets is lower or equal than 3. In addition, since the time complexity is improved, our algorithm also improves the solving times.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases k-Cutset, TSP, Linear algorithm, Constraint

Digital Object Identifier 10.4230/LIPIcs.CP.2021.29

Funding This work has been supported by the 3IA Côte d'Azur with the reference number ANR-19-P3IA-0002.

1 Introduction

The Traveling Salesman Problem (TSP) is a fundamental graph theory problem. It consists in finding a minimum cost cycle in a graph visiting all nodes. The TSP appeared in numerous domains such as biology with genome sequencing, industry with scan chains and electronic component drilling problems, positioning of very large telescopes, data clustering, scheduling problems and many others. The applications of TSP make it as fundamental as interesting: it is often an underlying problem of real-world problems.

Like many real-world problems, the search for the existence of a TSP is NP-Complete and finding the optimal one is NP-Hard. Thus, all classical methods designed for solving NP-Hard problems have been tried (MIP, CP, SAT, ...).

In order to solve the optimization version of the TSP without side constraints, the most efficient method is based on MIP: the so-called specialized solver Concorde [1]. It is mainly based on an LP relaxation of the TSP obtained by relaxing the integrity and the subtour constraints in combination with structural cutting planes. Indeed, a large number of cutting plane algorithms correct the structural defects of the LP relaxation lower bound. Simple ones such as imposing the 2-connectivity of a graph, and more complex ones such as the so-called Comb inequalities. Nevertheless, no polynomial time algorithm is known at this time to detect whether a graph does not violate a Comb inequality. Thus, a large number of polynomial algorithms have been developed in order to consider only particular cases of Comb inequalities [4, 6, 3, 13].

However, it appears that a TSP is often combined with other constraints. For instance, precedence constraints, TSPTW where there is a time window to visit a node. Thus, Concorde can no longer be used for these problems and CP becomes a good candidate because it is more robust to side constraints. The most efficient method at this time solving the TSP in CP is the Weighted Circuit Constraint (WCC) [2] in combination with the structural k-cutset constraint [10]. The optimization part of the WCC is based on the Lagrangian Relaxation (LR) of Held and Karp [8, 9]. The lower bound of the LR is computed by selecting a node x with its two lowest cost neighbors and a minimum spanning tree in the graph without x, i.e. a 1-tree. If we find a minimum 1-tree such that all its nodes have exactly two neighbors, then an optimal solution is obtained. Thus, the 1-tree is derived through the LR process until an optimal solution is obtained. Unfortunately, solving optimally the TSP with a LR can be extremely slow. The WCC integrates filtering algorithms based on the edge costs, the 1-tree cost and a degree constraint on the nodes. With this model, Benchimol et al. were able to obtain a competitive method with Concorde for problems with less than 100 nodes. In addition, the integration of the k-cutset constraint improved the competitiveness of this method. The idea of this constraint is that each cutset of a graph must contain at least two edges and any solution contains an even number of elements from this cutset. Actually, the CP model is based on a single graph variable with mandatory and optional edges. Thus, the k-cutset constraint tries to detect inconsistency in mandatory edges and to filter optional edges.

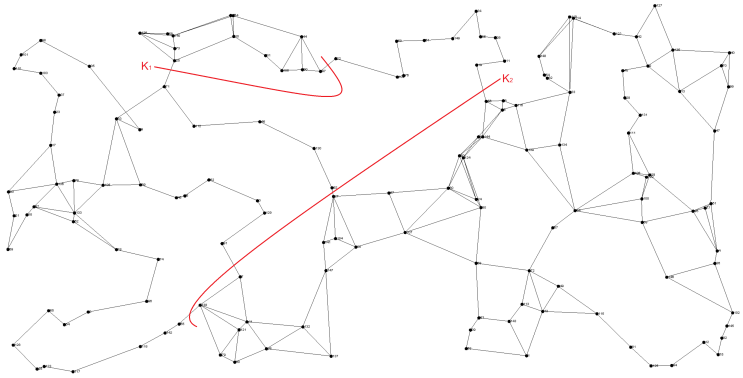

Figure 1 Graph kroA150 from TSPLib [14] while solving in a CP solver. K_1 is a 2-cutset and K_2 is a 4-cutset.

For instance, we show two k-cutsets in the graph of Figure 1. The 2-cutset K_1 contains 2 edges and the 4-cutset K_2 contains 4 edges.

Isoart and Régin [10] introduced a quadratic algorithm for the k-cutset constraint checking the consistency and performing some filtering operations. It is based on a 2-edge-connected subgraph and Tsin's algorithm [18]. Note that their algorithm only handle $k \leq 3$. This limitation is set because the number of k-cutsets in a graph is exponential: this corresponds to all possible partitions of the graph nodes, i.e. 2^n. However, we are not interested in all of them. The ones we are interested in are the k-cutsets containing k or $k-1$ mandatory edges in the graph. In addition, there are at most n mandatory edges in any TSP solution.

In this paper, we show that it is sufficient to study a set of k-cutsets lower than or equal to n in order to obtain a complete filtering. Moreover, we introduce a linear time algorithm for the k-cutset constraint for any k.

This article is organized as follows: first, we recall some concepts of graph theory. Then, we introduce the TSP in CP with the k-cutset constraint. Next, we define a new linear time algorithm for the k-cutset constraint. Finally, we discuss some experiments and we conclude.

2 Preliminaries

2.1 Definitions

The definitions about graph theory are taken from Tarjan's book [17].

A **directed graph** or **digraph** $G = (X, U)$ consists of a **node set** X and an **arc set** U, where every arc (x_i, x_j) is an ordered pair of distinct nodes. We note $X(G)$ the set of nodes of G such that $n = |X(G)|$ and $U(G)$ the set of arcs of G such that $m = |U(G)|$. In addition, $U(i)$ is the set of adjacent edges of i. The **cost** of an arc is a value associated with the arc. An **undirected graph** is a digraph such that for each arc $(x_i, x_j) \in U$, $(x_i, x_j) = (x_j, x_i)$. A **multigraph** is a digraph such that it can exist arcs that are not unique. If $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ are graphs, both undirected or both directed, G_1 is a **subgraph** of G_2 if $X_1 \subseteq X_2$ and $U_1 \subseteq U_2$. A **path** from node x_1 to node x_t in G is a list of nodes $[x_1, \ldots, x_t]$ such that (x_i, x_{i+1}) is an arc for $i \in [1..k-1]$. The path **contains** node x_i for $i \in [1..k]$ and arc (x_i, x_{i+1}) for $i \in [1..k-1]$. The path is **simple** if all its nodes are distinct. The path is a **cycle** if $k > 1$ and $x_1 = x_t$. A cycle is **Hamiltonian** if $[x_1, \ldots, x_{k-1}]$ is a simple path and contains every node of X. The **cost** of a path p, denoted by $w(p)$, is the sum of the costs of the arcs contained in p. For a graph G, a solution to the **traveling salesman problem (TSP)** in G is a Hamiltonian cycle $HC \in G$ minimizing $w(HC)$. An undirected graph G is **connected** if there is a path between each pair of nodes, otherwise it is **disconnected**. The maximum connected subgraphs of G are its **connected components**. A ***k*-edge-connected graph** is a graph in which there is no edges set of cardinality strictly less than k disconnecting the graph. A **tree** is a connected graph without a cycle. A tree $T = (X', U')$ is a **spanning tree** of G if $X' = X$ and $U' \subseteq U$. The U' edges are the **tree edges** T and the $U - U'$ edges are the **non-tree edges** T. A **minimum spanning tree** $T = (X', U')$ is a spanning tree minimizing the cost of the tree edges. A **bridge** is an edge such that its removal increases the number of connected components. A partition (S, T) of the nodes of G such that $S \subseteq X$ and $T = X - S$ is a **cut**. The set of edges $(x_i, x_j) \in U$ having $x_i \in S$ and $x_j \in T$ is the **cutset** of the (S, T) cut. A ***k*-cutset** is a cutset of cardinality k.

2.2 TSP in CP

The current best CP method solving the TSP is a combination of the Weighted Circuit Constraint (WCC) [2] and the structural constraint k-cutset [10]. The WCC is mainly based on the 1-tree Lagrangian Relaxation (LR) of Held and Karp [8, 9]. Intuitively, the LR derives a lower bound of the TSP (here, the 1-tree) until a solution of the TSP is found. A 1-tree is a minimum spanning tree in $G = (X - \{x\}, U)$ such that $x \in X$ is connected by its two nearest neighbors to the minimum spanning tree. Thus, a 1-tree covers the whole graph with n edges and a single cycle. In addition, if the 1-tree satisfies the degree constraint (each node of the 1-tree has exactly two neighbors), then the 1-tree is an optimal solution of the TSP. Therefore, the goal is to minimize the number of nodes that violate the degree constraint in the 1-tree. To do so, this constraint is integrated into the objective function and a Lagrangian multiplier π_i is associated to each node i. Let d_i be the degree of the node i in the 1-tree. For each node i of the graph, if $d_i < 2$, then π_i is decreased. Otherwise, if $d_i > 2$, then π_i is increased. Next, the edge cost $w((i, j))$ is modified such that $w'((i, j))$ is the modified cost and $w'((i, j)) = w((i, j)) + \pi_i + \pi_j$. Finally, we obtain an optimal solution of the TSP by computing a succession of 1-trees and modifying the edge costs.

However, experiments shown a very slow convergence toward the optimal solution. Thus, the WCC integrates the following filtering algorithms based on the costs:

- If an edge e does not belong to any 1-tree with cost smaller than a given upper bound, then e can be safely deleted.
- If an edge e belongs to all 1-trees with cost smaller than a given upper bound, then e is mandatory.

In addition, the WCC integrates a structural constraint imposing that each node has exactly two neighbors, i.e. the degree constraint.

Next, for each cutset of size k, the k-cutset constraint imposes that an even number of edges is mandatory. In practice, the study is limited to $k \leq 3$ because the given algorithm has a complexity growing with k. In addition, the interaction of the filtering algorithms and the convergence of the Lagrangian relaxation is not straightforward. Thus, Isoart and Régin [11] introduced an adaptive method in order to improve overall solving times.

About the search strategy, it consists in making a binary search where a left branch is an edge assignment and a right branch is an edge removal. More precisely, we use the search strategy LCFirst of Fages et al. [5] which is an interpretation of Last Conflict heuristics [7, 12] for graph variables. It selects one edge in the graph according to a heuristic and keeps branching on one extremity of this edge until the extremity is exhausted. Note that it keeps branching even if a backtrack occurs. Thus, it is a highly dynamic search strategy that learns from previous choices. Moreover, most of the search strategies are much more efficient (up to an order of magnitude) when LCFirst is used. In practice, we observe that using LCFirst strongly interferes with the Lagrangian relaxation and filtering algorithms.

In addition, the WCC uses a single undirected graph variable $G = (X, M, O)$ where all nodes are mandatory. Without loss of generality, we note O the set of optional edges, M the set of mandatory edges such that $O \cup M = U$ and $O \cap M = \emptyset$. In addition, M is a growing set and O is a shrinking set. Since we search for a solution of the TSP, when a solution is found, $|M| = n$ and $O = \emptyset$.

2.3 The k-cutset constraint

We previously said that the purpose of the k-cutset constraint is to ensure for each cutset in $G = (X, M, O)$ that a strictly positive and even number of edges are mandatory in any solution. To do so, Isoart and Régin [10] have shown the following proposition:

▶ **Proposition 1.** *Given K a k-cutset, then any Hamiltonian cycle C contains an even and strictly positive number of edges from K.*

Since G contains mandatory and optional edges, a k-cutset of G can be partitioned into two disjoint subsets of O and M. Therefore, we note a k-cutset $K = (M', O')$ of G such that $M' \subset M$ and $O' \subset O$.

▶ **Definition 2.** *For each k-cutset $K = (M', O')$, the k-cutset constraint ensure that $|M'| + |O'| \geq 2$ and $|M'|$ is even if $O' = \emptyset$.*

From Definition 2, we can therefore define the following consistency checks:

▶ **Corollary 3.** *If there is a k-cutset in G such that $k < 2$, then there is no solution for TSP(G).*

▶ **Corollary 4.** *If there is a k-cutset in G containing k mandatory edges such that k is odd, then there is no solution for TSP(G).*

In addition, we can define from Definition 2 the following filtering rules:

▶ **Corollary 5.** *Given a 2-cutset $K = (M', O')$ in G. Then, the edges of O' must become mandatory ($M \leftarrow M + O'$).*

▶ **Corollary 6.** *If there is a k-cutset K in G containing $k-1$ mandatory edges such that k is odd, then the non-mandatory edge e of K can be safely deleted ($O \leftarrow O - \{e\}$).*

▶ **Corollary 7.** *If there is a k-cutset K in G containing $k-1$ mandatory edges such that k is even, then the non-mandatory edge e of K must become mandatory ($M \leftarrow M + \{e\}$).*

Therefore, Corollary 3 and 4 allow checking the consistency, Corollary 5 and 7 allow finding mandatory edges, Corollary 6 allows removing edges.

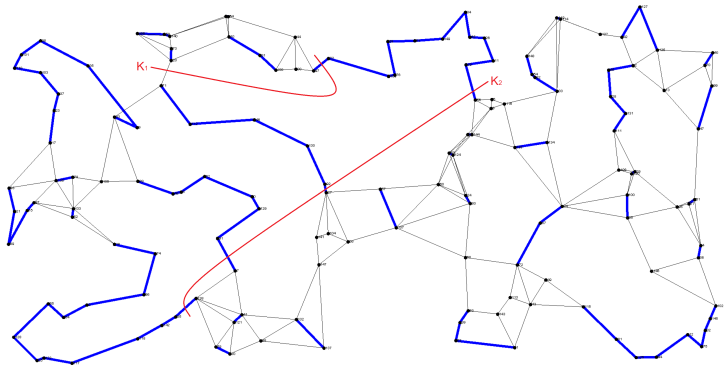

Figure 2 Representation of the graph from Figure 1 with mandatory edges (blue) and optional edges (dark). The two k-cutsets K_1 and K_2 are displayed in red.

For instance, we can deduce in Figure 2 that for the 2-cutset K_1, all the K_1 edges become mandatory by Corollary 5 or 7. Moreover, the 4-cutset K_2 is valid because the k-cutset constraint is consistent (K_2 contains only mandatory edges and $|K_2| = 4$ is even).

▶ **Definition 8.** *A k-cutset $K = (M', O')$ is failing if $|M'| = k$ and k is odd.*

▶ **Definition 9.** *A k-cutset $K = (M', O')$ is prunable if $k > 1$ and $|M'| = k-1$ and $|O'| = 1$.*

In order to find failing and prunable k-cutsets, Isoart and Régin [10] have developed an algorithm in $O(n(n+m))$. It finds all the k-cutsets such that $k \leq 2$ and all the 3-cutsets $K = (M', O')$ such that $|M'| > 0$. For $k = 1$ and $k = 2$, they use the non-trivial Tsin's algorithm [18] finding all cutsets of size smaller than or equal to 2 in a graph using a DFS in $O(n+m)$. For 3-cutsets $K = (M', O')$ such that $|M'| > 0$ in $G = (X, U)$, the main idea is the following: for each mandatory edge e_m, we look for the 2-cutsets in $G = (X, M - \{e_m\}, O)$. Since there are at most n mandatory edges in a TSP solution, this leads to a time complexity in $O(n(n+m))$. In addition, they give some practical improvements greatly reducing the number of considered mandatory edges e_m. For instance, they suggest using a 2-edge-connected subgraph of G minimizing the number of mandatory edges since all the 3-cutsets have at least 2 edges in this subgraph. Thus, it leads to the following algorithm: for each mandatory edge e_m in the 2-edge-connected subgraph, we look for the 2-cutsets in $G = (X, M - \{e_m\}, O)$.

Therefore, Corollary 3 and 5 are checked with Tsin's algorithm in $O(n+m)$. Corollary 4, 6 and 7 are checked for $k \leq 3$ in $O(n(n+m))$ with the algorithm described above. Finally, they have shown that the use of the k-cutset constraint allows reducing the number of backtracks by an order of magnitude with static strategies. Moreover, a gain of about a factor of 2 in solving times is obtained.

In the next section, we will introduce an algorithm enforcing the k-cutset constraint (i.e. checking Corollary 3, 4, 5, 6 and 7) for all k in $O(n+m)$.

3 A linear time algorithm

▶ **Definition 10.** *A **mandatory path** of G is a path $p = [x_1, ..., x_k]$ in G such that for each $i \in [1, k-1]$, the edge (x_i, x_{i+1}) is mandatory.*

▶ **Definition 11.** *A **path-merged graph** G_{p-m} of G is the graph G such that for each mandatory path $p = [x_1, ..., x_k]$ of G and $k > 2$, the nodes from x_2 to x_{k-1} are removed and a mandatory edge (x_1, x_k) is added.*

▶ **Definition 12.** *Given X' a set of nodes. The **merge** of X' is a mapping from all nodes of X' to a single node.*

▶ **Definition 13.** *Given $G_{p-m} = (X_{p-m}, M_{p-m}, O)$ a path-merged graph. A **merged graph** G_m of G_{p-m} is the multigraph G_{p-m} such that each connected component of the subgraph $G_{opt} = (X_{p-m}, \emptyset, O)$ of G_{p-m} are merged.*

▶ **Definition 14.** *Given $G_{p-m} = (X_{p-m}, M_{p-m}, O)$ a path-merged graph. A **2-merged graph** $G_2 - m$ of G_{p-m} is the multigraph G_{p-m} such that each 2-edge-connected component of the subgraph $G_{opt} = (X_{p-m}, \emptyset, O)$ of G_{p-m} are merged.*

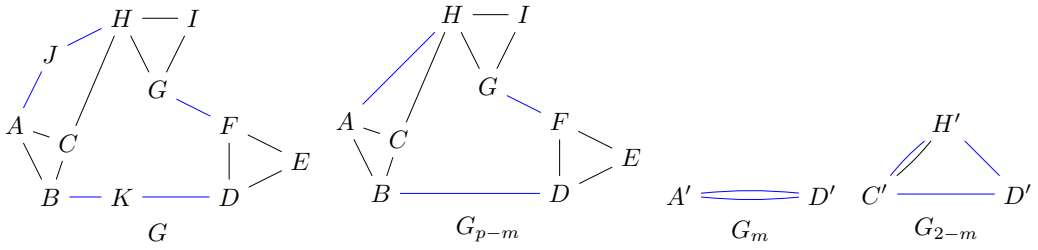

Figure 3 Given G a 2-edge-connected graph. G_{p-m} is the path-merged graph of G such that the mandatory paths are $p_1 = [A, J, H]$ and $p_2 = [B, K, D]$. G_m is the merged graph of G_{p-m}. G_{2-m} is the 2-merged graph of G_{p-m}.

For instance, Figure 3 shows an example of Definition 11, 13 and 14.

Without loss of generality, we will consider that G is connected. In addition, we will use $G_{p-m} = (X_{p-m}, M_{p-m}, O)$ the path merged graph of G, $G_m = (X_m, M_m, \emptyset)$ the merged graph of G_m and $G_{opt} = (X_{p-m}, \emptyset, O)$ the subgraph G_{p-m} containing only optional edges. Each removed nodes of G in G_{p-m} are connected with exactly two mandatory edges (thanks to the degree constraint of the WCC) and each path is replaced by a single mandatory edge. Therefore, we will often consider G_{p-m} instead of G.

3.1 Consistency Check

In order to determine whether G is consistent with the k-cutset constraint, we must check Corollary 3 and Corollary 4. We can check Corollary 3 with Tarjan's algorithm [16]. Using a DFS, it finds all the bridges of a graph (i.e. the 1-cutsets) in $O(n+m)$. Checking Corollary 4 requires to check for the existence of a failing k-cutset. We will show that it can be done in linear time with Proposition 16.

▶ **Definition 15.** *Given X' a set of nodes. We note $M^+(X') = \{(i,j)|(i,j) \in M, i \in X', j \notin X'\}$ the set of outoing mandatory edges of X' in M.*

▶ **Proposition 16.** *There is no failing k-cutset in G_{p-m} if and only if there is no connected component X' in G_{opt} such that $|M^+(X')|$ is odd.*

Proof. We note (i) there is no failing k-cutset in G_{p-m} and (ii) there is no connected component X' in G_{opt} such that $|M^+(X')|$ is odd.

$(i) \Rightarrow (ii)$ By definition, if there is no failing k-cutset in G, then there is no connected component X' in G_{opt} such that $|M^+(X')|$ is odd.

$(i) \Leftarrow (ii)$ If a non-empty k-cutset K cuts a connected component of G_{opt}, then K contains optional edges since G_{opt} is the graph of optional edges. Therefore, a k-cutset containing only mandatory edges cannot cut a connected component of G_{opt}. Then, the failing k-cutsets are obtained by partitioning the connected components of G_{opt} in G_{p-m}, i.e. all the k-cutsets of the merged graph.

For each $S \subset X_m$, the cutset size of the cut $(S, X_m - S)$ can be computed as follows: (1) make the sum of the number of adjacent edges of each node of S, then (2) subtract twice the number of edges (i,j) such that i and j belong to S (an edge connecting two nodes of S is counted twice in (1)). Since we consider that (ii) is true, the sum obtained by (1) is even. (2) substract an even number to the sum obtained by (1). Therefore, the cutset size is even and there is no failing k-cutset in G_{p-m}. ◀

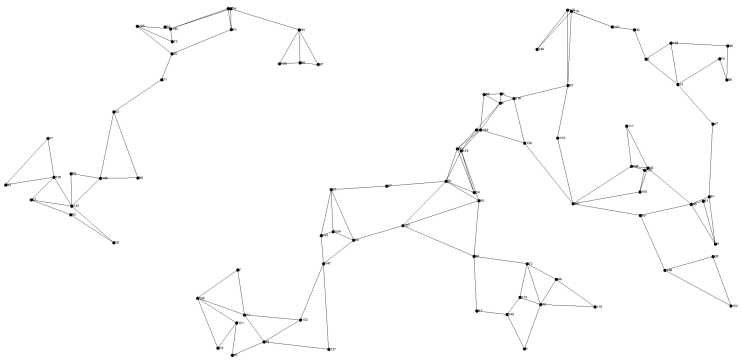

Figure 4 Representation of G_{opt} such that G is the graph of Figure 2.

For instance, in Figure 4, we notice that there are two connected components connected by 4 mandatory paths in Figure 2, so the k-cutset constraint is consistent with the graph of Figure 2. In addition, if we consider G_m of Figure 3, there is no failing k-cutset and each node has an even number of adjacent edges.

Then, we can describe an algorithm. First, compute the connected components of G_{opt}. Then, for each mandatory edge of M having its two endpoints in two different connected components of G_{opt}, we increase the number of mandatory outgoing edges for these connected components. Finally, we iterate on the connected components. If there is a connected component with an odd number of mandatory outgoing edges, then there is a failing k-cutset in G. The computation of the connected components of G_{opt} can be done in $O(n+m)$ with a DFS. The iteration over the mandatory edges of M can be done in $O(n)$. The check of the number mandatory outgoing edges for the connected components can be done in $O(n)$. Thus, we can test the consistency of the k-cutset constraint in $O(n+m)$.

3.2 Pruning

Corollary 5, 6 and 7 define filtering rules for the k-cutset constraint. First, Corollary 5 can be enforced with Tsin's algorithm [18]. It performs a single DFS in order to find all the 2-cutsets in a given graph, so it has a time complexity in $O(n+m)$.

In order to enforce Corollary 6 and 7, we need a method finding all k-cutsets having exactly $k-1$ mandatory edges, i.e. the prunable k-cutsets. Given a prunable k-cutset $K=(M',O')$ of G. If M' is removed, then the edge of O' is a bridge of G since $|O'|=1$. Formally, we define it in Proposition 17. We will exploit those bridges in order to enforce Corollary 6 and 7.

▶ **Proposition 17.** *If $K=(M',O')$ is a prunable k-cutset of G, then the edge of O' is a bridge in $G'=(X,M-M',O)$.*

Proof. A prunable k-cutset $K=(M',O')$ contains exactly $k-1$ mandatory edges and 1 optional edge. Thus, removing the $k-1$ mandatory edges in G transform K in a 1-cutset, i.e. in a bridge. ◀

▶ **Corollary 18.** *If $K=(M',O')$ is a prunable k-cutset of G, then the edge of O' is a bridge in G_{opt}.*

Proof. For each prunable k-cutset $K=(M',O')$, $M'\subseteq M$. Thus, from Proposition 17, the edge of O' is a bridge in $G'=(X,\emptyset,O)$. Therefore, it is a bridge in $G_{opt}=(X_{p-m},\emptyset,O)$. ◀

From Corollary 18, we find the edges belonging to some prunable k-cutsets in G by searching for bridges in G_{opt}. It can be done with Tarjan's DFS algorithm [16] in $O(n+m)$. Next, we must determine for each bridge whether it should be deleted or become mandatory. We therefore need to retrieve the set of prunable k-cutsets that contain each bridge.

Without loss of generality, we will consider that G is 2-edge-connected, i.e. G is connected and bridgeless. Note that it can be checked in $O(n+m)$ with Tarjan's algorithm [16]. In addition, we note $X_i(G')$ the connected component of G' containing the node i.

▶ **Proposition 19.** *Given $e\in O$ a bridge in the connected component X' of G_{opt} connecting $(X_1,X'-X_1)$. Then, the k-cutset $K=(M^+(X_1),\{e\})$ is a prunable k-cutset of G.*

Proof. In order to disconnect X_1 in G, the edges having exactly one end in X_1 must be removed, i.e. the k-cutset $K=(M',O')$ of $(X_1,X-X_1)$. It means $M'=M^+(X_1)$ and $O'=O^+(X_1)$. Since $e\in O$ is the bridge in X' of G_{opt} connecting $(X_1,X'-X_1)$, $O'=\{e\}$. Otherwise, e is not a bridge in G_{opt}. Finally, G is 2-edge connected, then $|M'|>0$. Thus, K is a prunable k-cutset of G. ◀

▶ **Proposition 20.** *Given $e\in O$ a bridge in G_{opt} and a prunable k-cutset $K'=(M',\{e\})$. If there are no failing k-cutsets in G, then there are no prunable k-cutsets K'' such that the pruning of e with K' and K'' is different.*

Proof. For any k-cutset $K''=(M'',\{e\})$ such that $M'\neq M''$, we can build a k-cutset $K'''=(M'\cup M'',\emptyset)$ containing only mandatory edges. In addition, if K''' is not a cutset, then e is not a bridge for M' or M''. If K''' is not a failing k-cutset, then $M'\cup M''$ is even. Therefore, M' and M'' are either even or odd. Thus, if there are no failing k-cutsets in G, then there are no prunable k-cutsets K'' such that the pruning of e with K' and K'' is different. ◀

From Proposition 19 and 20, we can describe a first algorithm: for each bridge of the connected component X' of G_{opt} connecting $(X_1, X' - X_1)$, count the number of mandatory edges having one end in X_1 and the other in $X - X_1$. If there is an even number of mandatory edges, then delete e. Otherwise, add e to the mandatory edges. Thus, for each bridge, we parse at most all the mandatory edges. There is at most $n - 1$ bridge in a graph and at most n mandatory edges. Therefore, this algorithm finds all the prunable k-cutsets in $O(n^2)$. Next, we will show how to improve this algorithm in order to obtain a linear time complexity. However, note that this algorithm is already much better than the one of the state of the art [10] because we find the k-cutsets for all k with a better time complexity.

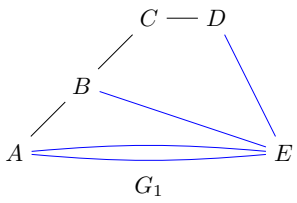

Figure 5 G_1 represents the 2-merged graph of the path-merged graph of Figure 2. Blue edges are mandatory paths and dark edges are bridges.

In Figure 5, the 2-merged graph of Figure 2, we notice that some prunable k-cutsets are much simpler than others to find. Indeed, for (A, B) there are two prunable k-cutsets: $K_1 = (M_1, \{(A, B)\})$ where $M_1 = \{(A, E), (A, E)\}$ and $K_2 = (M_2, \{(A, B)\})$ where $M_2 = \{(B, E), (D, E)\}$. In order to find M_1, we can simply search for the mandatory edges having one end in A and the other in $\{B, C, D, E\}$. In order to find M_2, we have to search for the mandatory edges having one end in $\{B, C, D\}$ and the other in $\{A, E\}$. The difference between M_1 and M_2 is that we can simply find M_1 by considering the mandatory edges with exactly one end in a single component. Thus, if for each bridge there is such a prunable k-cutset, then we simply have to count the number of outgoing mandatory edges of each 2-edge-connected components of G_{opt} in G. It leads to an algorithm with a linear time complexity. Unfortunately, this algorithm may not handle all the prunable k-cutsets. For instance, there are two prunable k-cutsets containing (B, C): $K_3 = (M_3, \{(B, C)\})$ such that $M_3 = \{(D, E)\}$ and $K_4 = (M_4, \{(B, C)\})$ such that $M_4 = \{(B, E), (A, E), (A, E)\}$. In that case, we cannot simply look at the neighbors of B or C to find the prunable k-cutsets containing (B, C). Indeed, B and C have more than one optional neighbor. It means that the bridge (B, C) disconnect G_{opt} in $(\{A, B\}, \{C, D\})$. Therefore, both connected components are not 2-edge-connected. Thus, we show in Corollary 21 that for a bridge e connecting (X_1, X_2) in G_{opt}, if X_1 (resp. X_2) is 2-edge-connected, then it exists a prunable k-cutset containing e and all the mandatory edges having exactly one end in X_1 (resp. X_2).

▶ **Corollary 21.** *Given X' a connected component of G_{opt}. If $e \in O$ is a bridge in X' of G_{opt} connecting $(X_1, X' - X_1)$ such that X_1 is 2-edge-connected, then there is a prunable k-cutset $K = (M^+(X_1), \{e\})$.*

Proof. Immediate from Proposition 19. ◀

The advantage of Corollary 21 over Proposition 19 is that the 2-edge-connected component X_1 of Corollary 21 are disjoint nodes sets. Thus, parsing the neighbors of the 2-edge-connected components consider at most twice the total number of mandatory edges whereas Proposition 19 can reconsider for each component all the mandatory edges of the 2-merged graph. We will use this idea in order to obtain a linear time algorithm.

In Figure 5, all the k-cutsets formed by the neighborhood of a component are: $K_A = (\{(A,E),(A,E)\},\{(A,B)\})$, $K_B = (\{(B,E)\},\{(B,A),(B,C)\})$, $K_C = (\emptyset,\{(C,B),(C,D)\})$ and $K_D = (\{(D,E)\},\{(D,C)\})$. Among them, K_A and K_D are prunable k-cutsets, then we can immediately deduce for K_1 that (A,B) is deleted (by Corollary 6) and for K_D that (D,C) becomes mandatory (by Corollary 5 or 7). If we update the k-cutsets we have: $K_A = (\{(A,E),(A,E)\},\emptyset\})$, $K_B = (\{(B,E)\},\{(B,C)\})$, $K_C = (\{(C,D)\},\{(C,B)\})$ and $K_D = (\{(D,E),(D,C)\},\emptyset)$. We notice that both K_B and K_C become prunable k-cutsets. Thus, (C,B) becomes mandatory for both K_B and K_C (by Corollary 5 or 7). Finally, all bridges of G_1 have been solved.

Note that the subgraph of optional edges of the 2-merged graph can be more sophisticated than a simple path of bridges edges: it can be a tree. However, it cannot exist a cycle in this subgraph since the 2-edges-connected components are merged in the 2-merged graph. For example, (1) of Figure 6 shows a 2-merged graph such that the subgraph of optional edges is not a single path. We note that (1) is rooted in A and each node of the set of nodes $S = \{B,F,G,H,I,J\}$ has exactly one optional neighbor. Thus, we can start by applying Corollary 21 on S, i.e. the leaves of the tree. Leaves are always valid candidates for Corollary 21 because they have no optional child and a single optional parent. Then, either there are no more leaves and therefore there are no more prunable k-cutsets or there are leaves and we can apply Corollary 21 to these leaves. Finally, we suggest to recursively apply this process until there are no more leaves in the tree. A sketch of the algorithm is:

- Find bridges of G with the DFS-based Tarjan's algorithm.
- Mark the 2-edge-connected components in postorder, i.e. the order of a node is set when it is backtracked in the DFS.
- For each mandatory edge (i,j), increase the number of outgoing mandatory edges of the 2-edge-connected component of i and j.
- Iterate over the 2-edge-connected components C_i of the 2-merged graph with the defined postorder and prune the bridge connected to C_i.

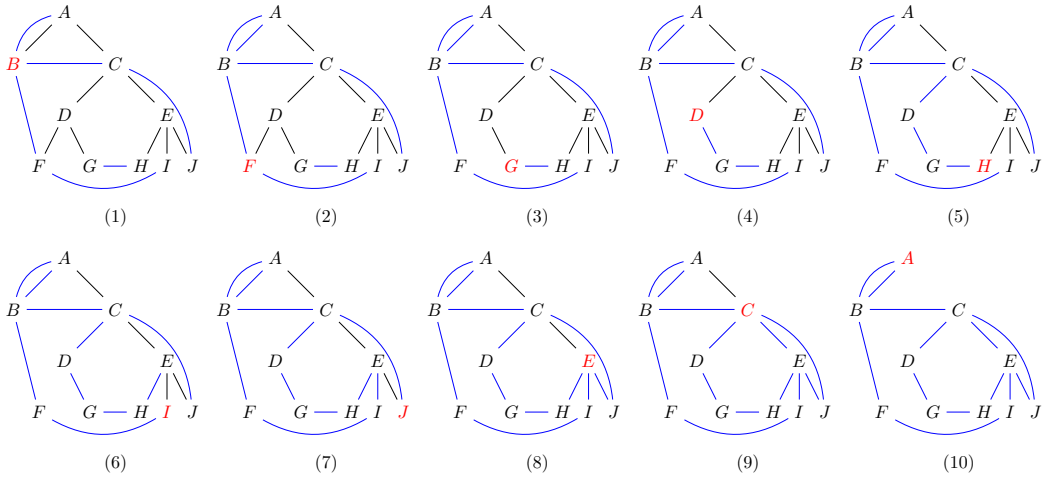

Figure 6 Example of an execution of the k-cutset pruning algorithm on the graph (1). Blue edges are mandatory paths, dark edges are bridges. The red node is the current 2-edge-connected component of the algorithm step.

For instance, Figure 6 shows an execution of the algorithm in a 2-merged graph. Tarjan's algorithm allows us to create the 2-merged graph where black edges are bridges and blue edges are mandatory edges. In this tree, the postorder traversal is $\{B,F,G,D,H,I,J,E,C,A\}$.

Note that the postorder is used to guarantee that for each node considered in the execution of the algorithm, all its children have already been pruned. From (1) to (10), we show the iteration over the 2-edge-connected components C_i marked as red nodes. Finally, finding the bridges is performed in $O(n+m)$, parsing the mandatory edges is performed in $O(n)$ and parsing the 2-edge-connected components is performed in $O(n)$. Thus, our algorithm finds the prunable k-cutsets for all k in $O(n+m)$. Algorithm 1 is a possible implementation.

It takes as input $G = (X, M, O)$. We note CC the set of connected components in G_{opt} and $2CC$ the set of 2-edge-connected components in G_{opt} such that $2CC_i$ is the 2-edge-connected component containing the node i. First, we start by searching the bridges with Tarjan's algorithm based on a DFS in order to build CC and $2CC$. Within the DFS, $2CC$ is constructed with respect to the postorder tree traversal. Thus, iterating on $2CC$, we obtain the postorder tree traversal of the 2-merged graph of G. In Tarjan's algorithm, a 2-edge connected component C is found when a bridge e is found. Thus, we associate e with C by setting C.bridge=e. For instance, in (1) of Figure 6, each node is a 2-edge-connected component knowing its parent, i.e. a bridge. Thus, the only node having no parent is the root node and all the other nodes have a single parent that is an optional edge.

Secondly, we count the number of outgoing mandatory edges for each connected component of CC and for each 2-edge-connected component of $2CC$. We then consider the connected components $C \in CC$ such that $|C| > 1$. If $|C| = 1$, then C has two adjacent mandatory edges and C contains a single node. Thus, considering C such that $|C| > 1$ is equivalent of considering the path-merged graph of G. In addition, we know that each node i of C has at most one adjacent mandatory edge (i, j). Therefore, $|M(i)| \leq 1$. Otherwise, the node i would not belong to C. We note $M(i).firstEdge()$ the first edge in the list of the adjacent mandatory edges of i. Since we are looking for the outgoing mandatory edges, we only consider the nodes with $M(i) = 1$. If j is an outgoing edge of C, then we increase the number of outgoing mandatory edges of C noted $M^+(C)$. If i and j do not belong to the same 2-edge-connected component, then we increase the number of outgoing mandatory edges of $2CC_i$ noted $M^+(2CC_i)$ ($M^+(2CC_j)$ is increased when the node j is considered by the foreach). Note that we can check if the node i belongs to the nodes set C' with the funtion $C'.isIn(i)$. In (1) of Figure 6, there is a single connected component C so $M^+(C) = 0$. However, there are 10 2-edge-connected components ($\{A, B, C, D, E, F, G, H, I, J\}$). For example, $M^+(2CC_A) = 1$ and $M^+(2CC_B) = 3$. Afterwards, we check the consistency. If there is $C \in CC$ such that $M^+(C)$ is odd, then there is a failing k-cutset and we return False. In (1) of Figure 6, there is no $C \in CC$ such that $M^+(C)$ is odd, so (1) is consistent with the k-cutset constraint.

Thirdly, we perform the pruning step. We iterate on all the 2-edge connected components C of $2CC$. We note (i, j) the bridge associated to C. If $M^+(C)$ is odd, then (i, j) becomes mandatory (i.e. it is added to M) and $M^+(2CC_i)$ and $M^+(2CC_j)$ are increased by 1. Whether $M^+(C)$ is even or odd, (i, j) is removed from O. Indeed, if the edge becomes mandatory, then it must not be in the set of the optional edges. For instance, in (1) of Figure 6, we consider the node B and the bridge (B, A). Note that in (1), $M^+(2CC_A) = 1$ and $M^+(2CC_B) = 3$. Since $M^+(2CC_B)$ is odd, (B, A) becomes mandatory and $M^+(2CC_A) = 2$ and $M^+(2CC_B) = 4$. Next, we consider the node F and its bridge (F, D). $M^+(2CC_F) = 2$ so (F, D) is removed and $M^+(2CC_F)$ remains unchanged. Then, we repeat this process until (10). We note that the nodes are chosen in the postorder.

Algorithm 1 Perform the consistency check and the pruning of k-cutset constraint.

```
1  k-cutset (G = (X, M, O))
       Input: A graph G=(X,M,O).
       Output: A boolean specifying whether G contains a failing k-cutset
       // CC : connected components of G − M (G_opt)
       // 2CC : postorder 2-edge-connected components of G − M (G_opt)
2      computeBridgesDFS(G − M, CC, 2CC) ;
3      foreach connected components C ∈ CC do
4          if |C| > 1 then
5              foreach node i ∈ C do
6                  if |M(i)| = 1 then
7                      (i, j) ← M(i).firstEdge();
8                      if not C.isIn(j) then
9                          M+(C) ← M+(C) + 1;
10                     if not 2CC_i.isIn(j) then
11                         M+(2CC_i) ← M+(2CC_i) + 1;
       // Consistency check
12     foreach connected components C ∈ CC do
13         if M+(C) is odd then return False ;
       // Pruning
14     foreach 2-edge-connected components C ∈ 2CC do
15         if C.bridge ≠ nil then
16             (i, j) ← C.bridge;
17             if M+(C) is odd then
18                 M+(2CC_i) ← M+(2CC_i) + 1;
19                 M+(2CC_j) ← M+(2CC_j) + 1;
20                 M ← M + (i, j);
21             O ← O − (i, j);
22     return True;
```

4 Experiments

The algorithms have been implemented in Java 11 in a locally developed constraint programming solver. The experiments were performed on Clear Linux with an Intel Xeon E5-2696v2 and 64 GB of RAM. The instances are from the TSPLib [14], a library of reference graphs for the TSP. We rerun experiments ran in Isoart and Régin [10] and exclude instances solved in less than two seconds. We also include some harder instances. The name of each instance is suffixed by its number of nodes. In our implementation, the TSP is modeled by the WCC using the CP-based LR configuration introduced in Isoart and Régin [11]. We note "state of the art" the TSP model with the state of the art k-cutset algorithm and "linear full k-cutset" the TSP model with our algorithm. The search strategy used is LCFirst with the heuristic minDeltaDeg [5] which is also the state of the art. Given $e = (i, j)$ an edge, minDeltaDeg selects the edge with the minimum difference between the sum of the number of optional neighbors of i and j and the sum of the number of mandatory neighbors of i and j. Thus, we

compare linear full k-cutset and the state of the art. We give the number of backtracks (#bk) and the solving time in seconds in arrays for the k-cutset constraint with different search strategies. In addition, we set a timeout *t.o.* of 100,000 seconds. All considered instances are symmetric graphs.

Table 1 General results comparing the state of the art and linear full k-cutset.

Instance	(1) State of the art		(2) Linear full k-cutset		Ratio (1)/(2)	
	time(s)	#bk	time(s)	#bk	time(s)	#bk
kroB100	5.4	4,816	3.6	3,308	1.5	1.5
kroE100	2.3	1,804	2.4	2,212	0.9	0.8
pr124	2.8	1,856	3.5	2,482	0.8	0.7
pr136	20.4	18,684	20.0	21,446	1.0	0.9
kroA150	6.1	4,164	4.1	2,798	1.5	1.5
kroB150	262.6	247,574	153.6	154,002	1.7	1.6
si175	288.5	301,102	280.8	358,676	1.0	0.8
rat195	38.5	24,274	37.8	27,512	1.0	0.9
d198	14.0	7,192	8.6	4,782	1.6	1.5
kroA200	401.0	237,806	323.7	200,392	1.2	1.2
kroB200	127.8	87,296	135.9	109,322	0.9	0.8
tsp225	121.5	65,002	139.3	89,946	0.9	0.7
gr229	227.0	166,378	139.8	114,434	1.6	1.5
gil262	5,230.2	2,254,728	2,970.7	1,711,410	1.8	1.3
pr264	4.7	690	4.9	642	0.9	1.1
a280	7.0	2,372	6.6	2,484	1.1	1.0
lin318	32.9	7,834	11.0	3,456	3.0	2.3
gr431	1,724.8	265,698	1,358.6	247,090	1.3	1.1
pcb442	15,081.5	4,130,580	16,490.1	5,555,756	0.9	0.7
d493	95,916.6	13,478,616	69,247.1	11,346,180	1.4	1.2
mean	5,975.8	1,065,423.3	4,567.1	997,916.5	1.3	1.1

Table 1 shows the solving times and backtrack numbers for the state of the art and the linear full k-cutset. A ratio column display both solving times and backtrack numbers gain for the linear full k-cutset. For most instances, we observe a gain in backtrack numbers and solving times. Otherwise, the results are quite close to the state of the art results. Indeed, we observe an average gain in solving time of 30% and 10% in backtracks. On average, the state of the art runs in 178 backtracks per seconds while linear full k-cutset runs in 219 backtracks per seconds. The low backtrack number gain suggests that most of the cutsets are in fact k-cutsets with $k \leq 3$, and therefore the state of the art algorithm already finds most of the cutsets.

Nevertheless, the TSP model includes a Lagrangian relaxation and the relation between the filtering algorithms and the Lagrangian relaxation is not clear [15, 11]. Moreover, the LCFirst minDeltaDeg search strategy is extremely dynamic. Thus, in order to have a better understanding of the impact of the linear full k-cutset, we compare it with the static search strategy maxCost, i.e. edges are selected by decreasing costs.

In Table 2, we observe a gain on all instances that are both solved by (1) and (2), the solving is therefore much more stable. In addition, we observe that the instance gil262 timeout in the state of the art. Excluding it, we obtain an average gain of 80% in solving time

Table 2 General results of the static search strategy maxCost comparing the state of the art (1) with our algorithm (2).

Instance	(1) State of the art		(2) Linear full k-cutset		Ratio (1)/(2)	
	time(s)	#bk	time(s)	#bk	time(s)	#bk
kroB100	7.7	7,640	6.1	7,056	1.3	1.1
kroE100	10.6	11,328	6.3	6,136	1.7	1.8
pr124	1.3	316	1.1	294	1.1	1.1
pr136	101.5	86,860	73.4	86,094	1.4	1.0
kroA150	57.7	55,940	46.7	49,016	1.2	1.1
kroB150	408.1	344,440	314.2	299,728	1.3	1.1
si175	4,168.6	4,661,506	3,190.8	4,305,208	1.3	1.1
rat195	486.1	372,438	364.0	358,936	1.3	1.0
d198	71.9	52,618	50.2	41,432	1.4	1.3
kroA200	3,111.1	1,944,312	1,813.5	1,209,136	1.7	1.6
kroB200	491.3	333,440	376.7	303,318	1.3	1.1
tsp225	15,252.7	11,579,074	10,224.3	9,209,292	1.5	1.3
gr229	2,349.7	1,955,538	1,591.3	1,607,300	1.5	1.2
gil262	*t.o.*	*t.o.*	70,450.6	36,507,156	$\geq$1.2	
pr264	7.5	902	6.7	748	1.1	1.2
a280	46.9	20,380	9.6	4,024	4.9	5.1
lin318	65.0	21,292	14.4	5,934	4.5	3.6

and 60% in backtracks. Thus, this shows that the impact of considering k-cutsets for any k is useful for the performance. In addition, these results suggest that LCFirst minDeltaDeg fills a part of the lack of structural constraints in the TSP model in CP.

In Table 3, we study the size of the founded k-cutsets in linear full k-cutset with LCFirst minDeltaDeg. We can observe that the mean size the founded k-cutsets is 3. It confirms the fact that the state of the art algorithm already finds a large part of the k-cutsets. However, larger k-cutsets exist: the average number of maximum size of the k-cutsets is 14.1. This is why we obtain a more interesting backtrack gain than the state of the art whereas the average k-cutset size is 3.

Finally, our linear full k-cutset algorithm is simple to implement and allows us to obtain an improvement of the solving times and the number of backtracks.

5 Conclusion

In this paper, we have introduced a new linear time algorithm checking the k-cutset constraint for any k. Experiments have shown that our algorithm leads to an improvement of solving times. Moreover, we have shown that on average most of the cutsets are of size 3 even if we found some much larger cutsets. We hope that other structural constraints will be integrated into the WCC: they make the CP competitive in the same way that Comb inequalities make the MIP efficient.

Table 3 Comparison of mean and max k-cutsets size.

Instance	k-cutsets size	
	mean	max
kroB100	2.63	7
kroE100	2.72	7
pr124	2.50	10
pr136	2.57	14
kroA150	2.86	8
kroB150	2.21	11
si175	2.63	13
rat195	4.77	26
d198	2.40	13
kroA200	2.79	10
kroB200	2.96	13
tsp225	2.93	13
gr229	3.08	14
gil262	2.55	17
pr264	2.48	25
a280	3.31	9
lin318	5.47	14
gr431	2.45	16
pcb442	3.53	24
d493	2.43	26
mean	3.0	14.5

References

1 David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

2 Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3):205–233, 2012. URL: `https://hal.archives-ouvertes.fr/hal-01344070`.

3 Václav Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Math. Program.*, 5(1):29–40, 1973. `doi:10.1007/BF01580109`.

4 Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. `doi:10.4153/CJM-1965-045-4`.

5 Jean-Guillaume Fages, Xavier Lorca, and Louis-Martin Rousseau. The salesman and the tree: the importance of search in CP. *Constraints*, 21(2):145–162, 2016.

6 Martin Grötschel and Manfred Padberg. On the symmetric travelling salesman problem i: Inequalities. *Mathematical Programming*, 16:265–280, December 1979. `doi:10.1007/BF01582116`.

7 Robert Haralick and Gordon Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, January 1979.

8 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

9 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

10 Nicolas Isoart and Jean-Charles Régin. Integration of structural constraints into tsp models. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming*, pages 284–299, Cham, 2019. Springer International Publishing.

11 Nicolas Isoart and Jean-Charles Régin. Adaptive CP-Based Lagrangian Relaxation for TSP Solving. In Emmanuel Hebrard and Nysret Musliu, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 300–316, Cham, 2020. Springer International Publishing.

12 Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary, and Vincent Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18):1592–1614, 2009.

13 Adam N. Letchford and Andrea Lodi. Polynomial-Time Separation of Simple Comb Inequalities. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization*, pages 93–108, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

14 Gerhard Reinelt. TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

15 Meinolf Sellmann. Theoretical Foundations of CP-Based Lagrangian Relaxation. In *Principles and Practice of Constraint Programming – CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 – October 1, 2004, Proceedings*, pages 634–647, 2004.

16 Robert E. Tarjan. A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2:160–161, 1974.

17 Robert E. Tarjan. *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

18 Yung H. Tsin. Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130–146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP).