Discrete Optimization 7 (2010) 1–12

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Generating partitions of a graph into a fixed number of minimum weight cuts

Gerhard Reinelt [*], Klaus M. Wenger

University of Heidelberg, Institute for Computer Science, Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany

ARTICLE INFO

Article history:
Received 20 May 2004
Received in revised form 2 July 2009
Accepted 7 July 2009
Available online 3 August 2009

Keywords:
Mincut
Graph partitioning
Cactus
Traveling salesman problem

ABSTRACT

In this paper, we present an algorithm for the generation of all partitions of a graph G with positive edge weights into k mincuts. The algorithm is an enumeration procedure based on the cactus representation of the mincuts of G. We report computational results demonstrating the efficiency of the algorithm in practice and describe in more detail a specific application for generating cuts in branch-and-cut algorithms for the traveling salesman problem.

1. Introduction

Throughout this paper, $G = (V, E, w)$ denotes a simple connected undirected graph with vertex set $V = \{1, \dots, n\}$, edge set E, $|E| = m$, and positive real edge weights $w_e > 0$, for all $e \in E$. An *edge* $e \in E$ is an unordered pair $\{u, v\}$ of vertices $u, v \in V$. A *cut* in G is the set of edges linking some vertex set $U \neq \emptyset$, $U \neq V$, to its complement $\overline{U} = V \setminus U$. Thus a cut is completely defined by either of its *shores* U or $\overline{U}$. For the purposes of this paper we will denote a cut by its inducing node set U (or $\overline{U}$) and consider U and $\overline{U}$ as different cuts. The *size* of a cut $U \subset V$ is its cardinality $|U|$. The weight of a cut $U \subset V$ is defined as the sum of the weights of the edges between U and $\overline{U}$ and is denoted by $w(U : \overline{U})$. A global minimum weight cut in G is also called *mincut* and its weight is denoted by λ. The set of all mincuts of G is denoted by $\mathcal{M}$.

For some integer k, $2 \leq k \leq n$, a *k-cut* is a k-partition $\{V_1, \dots, V_k\}$ of V, i.e., a partition of V into k cuts V_i. The elements of a partition are also called *cells*. The *weight* of a k-cut is defined as the total weight of all edges with end nodes in different cells. A *minimum k-cut* is a k-cut of minimum weight and its weight is denoted by λ_k. Obviously $\lambda = \lambda_2$ and $\lambda_{k'} \geq \lambda_k$, if $k' > k$. Since every cell of a k-cut has weight at least λ, it follows that $\lambda_k \geq k\frac{\lambda}{2}$.

This paper focuses on the situation where a minimum k-cut has the least possible weight $\lambda_k = k\frac{\lambda}{2}$ which is equivalent to saying that every cell is a mincut. Such a k-cut is called a *mincut k-partition* and the set of all mincut k-partitions of G is denoted by $\mathcal{M}_k$. While G always has a minimum k-cut, it does not necessarily always have a mincut k-partition. Namely, if $\lambda_k > k\frac{\lambda}{2}$, then $\mathcal{M}_k = \emptyset$.

The minimum 2-cut or *mincut problem* is a classical problem and well studied (see Chekuri et al. [1] and Jünger et al. [2] for overviews and computational comparisons of several mincut algorithms). Dinitz et al. [3] introduced the so-called *cactus representation*, denoted by $\mathcal{H}$ in the following, of the set $\mathcal{M}$ of all mincuts of G. Whereas G has $O(n^2)$ mincuts [4], a cactus $\mathcal{H}$ only needs $O(n)$ space for storing $\mathcal{M}$. Furthermore, $\mathcal{H}$ also mirrors the inclusion and intersection structures of $\mathcal{M}$, gives quick access to the size of mincuts, helps to keep the space requirement of the suggested generation algorithm small, and enables an efficient test for mincut-intersection.

[*] Corresponding author.
E-mail address: gerhard.reinelt@informatik.uni-heidelberg.de (G. Reinelt).

1572-5286/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2009.07.001

A minimum k-cut in G where k is not fixed, i.e., is part of the input, cannot be found in polynomial time [5] (unless P=NP). For fixed k, Goldschmidt and Hochbaum in [5] give the first polynomial algorithm (requiring $O(n^{k^2/2-3k/2+4})$ minimum (s,t)-cut computations). Karger and Stein [4] suggest a randomized Monte Carlo algorithm for the problem. Saran and Vazirani [6] and Kapoor [7] propose $2-\frac{2}{k}$-approximation algorithms for the minimum k-cut problem. Zhao et al. [8] based their approximation of minimum k-cuts on minimum 3-cuts.

The literature on minimum k-cuts often concentrates on $k \le 6$. For minimum 3-cuts in planar graphs, see Hochbaum and Shmoys [9] and He [10]. Burlet and Goldschmidt [11] give an $O(mn^3)$ algorithm for the minimum 3-cut problem. Nagamochi and Ibaraki [12] are concerned with $k \in \{3, 4\}$ and Nagamochi et al. [13] with $k \in \{5, 6\}$. Levine [14] suggests fast randomized algorithms for the computation of minimum $\{3, 4, 5, 6\}$-cuts. Kamidoi et al. [15] present algorithms for minimum 3- and 4-cuts requiring $O(n^4)$ and $O(n^6)$ maximum flow computations, respectively.

Given k nodes, the problem of finding a minimum k-cut such that each cell contains exactly one of these nodes is NP-hard for $k > 2$ (Dahlhaus et al. [16]). In the case of a planar graph, this problem is solvable in polynomial time for fixed k [16] (see Yeh [17] for an elegant algorithm). Dahlhaus et al. [16] give a $2-\frac{2}{k}$-approximation algorithm for the minimum k-terminal cut problem; Calinescu et al. [18] and Karger et al. [19] improve on the approximation guarantee. For $k = 3$, both Cunningham and Tang [20] and Karger et al. [19] obtain the best possible approximation guarantee $\frac{12}{11}$.

In this paper we consider the problem of generating all mincut k-partitions of a graph G, for some given k. Section 2 describes the cactus representation of mincuts which is the basis for the efficient algorithm for generating $\mathcal{M}_k$. Section 3 outlines the general framework of the algorithm taking a sequence of mincuts as input. In Section 4 this sequence is reduced by ignoring mincuts that cannot be in a mincut k-partition. The test for intersection of mincuts described in Section 5 is a key to the efficiency of our algorithm. Section 6 discusses computational results demonstrating the efficiency in practice. Section 7 is concerned with the particular application of using mincut k-partitions in branch-and-cut approaches to the TSP. For the TSP, the generation can be speeded up further by skipping some undesirable mincut k-partitions, as described in Section 8. Section 9 shows that the generation of the desirable partitions is feasible and fast even for large TSP instances.

2. The cactus representation of mincuts

A graph is called *cactus* if every edge is contained in at most one cycle. We distinguish between *cycle edges* (contained in a cycle) and *tree edges* (contained in no cycle). Cacti we encounter here will be connected and for cacti we say *node* instead of vertex. A cycle with h edges is called an *h-cycle*. For technical reasons, we replace every tree edge by a pair of parallel edges so that we obtain 2-cycles. Whenever appropriate, the edges of a cactus are assumed to have weight 1.

Given $G = (V, E, w)$, there is a cactus with node set N and a mapping $\pi : V \to N$ such that $\mathcal{M} = \{\pi^{-1}(C) \mid C \subset N \text{ is a mincut of the cactus}\}$ (see Dinitz et al. [3] or Naor and Vazirani [21]). This cactus is called a *cactus representation* of $\mathcal{M}$ or a *cactus* of G. Its nodes $n_i \in N$ correspond to subsets $\pi^{-1}(\{n_i\})$ (possibly $\emptyset$) of V. Every mincut of the cactus induces a mincut of G and every mincut of G can be obtained at least once in this way. For a cut $C \subset N$ we write $V(C)$ for $\pi^{-1}(C)$ and say that $V(C) \subset V$ is *induced* by C. A cactus node inducing $\emptyset$ is called *empty*.

The mincuts of a cactus are easily obtained by removing two different edges of the same cycle which splits a cactus into two connected components. The nodes of each component form a mincut of the cactus. By traversing the connected component of a mincut of a cactus of G and collecting the vertices of G induced by cactus nodes, we get the induced mincut $M \in \mathcal{M}$ of G in time $O(|M|)$. A cactus of G can thus be viewed as a condensed set representing all $O(n^2)$ mincuts of G in space $O(n)$ instead of $O(n^3)$ necessary for listing all mincuts explicitly.

There may be more than one cactus representation of $\mathcal{M}$, but the representation can be made unique by requiring additional properties. A cactus node belonging to exactly ν cycles is called a *ν-junction node*. A representation is called *canonical* [22–24] if there is no empty 2-junction node belonging to a 2-cycle and no empty 3-junction node. The canonical cactus is unique [25] and every cactus of G can be transformed into its canonical form [22,24] denoted by $\mathcal{H}$ in the following.

The first polynomial construction algorithm is due to Karzanov and Timofeev [26]. Nagamochi and Kameda [25] show that the canonical cactus of G has at most $2n$ nodes. We refer to De Vitis [22], Fleischer [23], Nagamochi et al. [24], and Wenger [27] for deterministic algorithms constructing $\mathcal{H}$ from G.

The deletion of two edges of a cycle of a cactus yields two complementary mincuts. In order to define without ambiguity which mincut we refer to, we orient the cactus edges. From now on, let the edges of a cactus be oriented in such a way that we walk around a cycle if we follow the directed edges of a cycle.

A directed edge c points from its tail node $\text{tail}(c)$ to its head node $\text{head}(c)$. Every mincut of $\mathcal{H}$ and thus every mincut of G can now be described by an ordered pair (c_i, c_j) of cactus edges where $c_i \ne c_j$ and both c_i and c_j belong to the same cycle. We define that (c_i, c_j) describes the mincut of $\mathcal{H}$ or G at the head of c_i. The mincut described by (c_j, c_i) is the complement of the mincut described by (c_i, c_j).

Fig. 1 shows an example of a cactus representation. In the graph G, dotted edges have weight 1 and solid edges have weight 2. The pair (c_1, c_2) describes the mincut $\{4, 18\}$ of G. The pair (c_3, c_4) describes a different mincut of $\mathcal{H}$ containing an empty node, but the same mincut $\{4, 18\}$ of G. If a mincut of G is induced by more than one mincut of $\mathcal{H}$, then it is induced by exactly two. Such a mincut of G is called a *double mincut*. Only empty 2-junction nodes of $\mathcal{H}$ give rise to double mincuts [22].

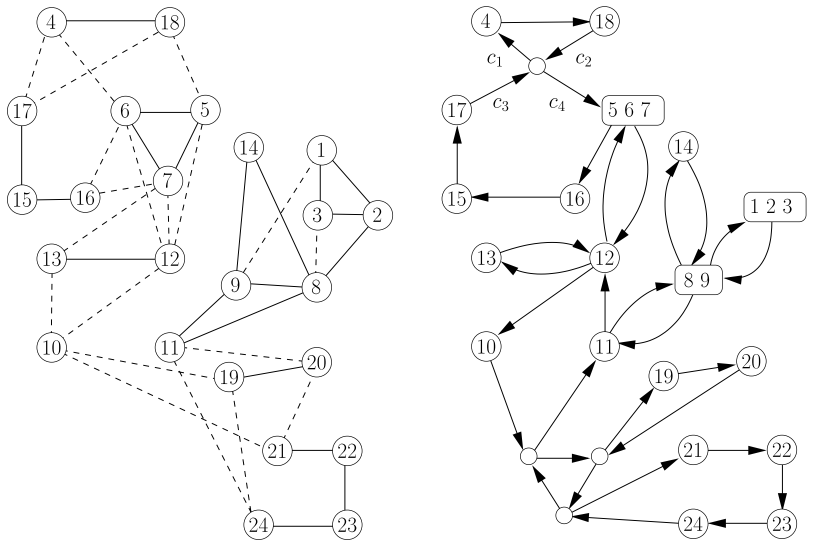

Fig. 1. A graph G with 24 vertices and its canonical cactus $\mathcal{H}$ with 23 nodes.

Remark 1. To avoid ambiguity and to lock enough nodes for the intersection test in Section 5, we always describe a double mincut D of G by the pair of cactus edges (c_i, c_j) such that $\text{head}(c_i)$ is empty and equals $\text{tail}(c_j)$. We rule out the second possibility for inducing D. For example, we choose (c_3, c_4) instead of (c_1, c_2) to describe the double mincut $\{4, 18\}$ in Fig. 1.

A mincut $M \in \mathcal{M}$ obtained by removing two different edges of a given cycle of $\mathcal{H}$ is said to be *represented* by the cycle. A mincut of G that is described by a pair (c_i, c_j) of edges of a given cycle such that $\text{head}(c_i) = \text{tail}(c_j)$ is called a *bead* of G represented by the cycle. An h-cycle of $\mathcal{H}$ represents h beads of G.

3. Generating all mincut k-partitions

We have $|\mathcal{M}| = O(n^2)$ and there are $O(n^{2(k-1)})$ minimum k-cuts in G [4]. We select them in the case of $\lambda_k = k\frac{\lambda}{2}$ in an efficient way from the larger set of all $O(n^{2k})$ many k-subsets of $\mathcal{M}$. For fixed k, generating these $O(n^{2k})$ objects and testing the mincuts in each of them in $O(n)$ for intersection and cover results in a naive polynomial time algorithm for generating $\mathcal{M}_k$. We know from experiments [28] that such a straightforward algorithm is not practicable and more sophistication is needed.

3.1. Exhaustive search

A *partial mincut partition* (PMP) of V is a set of pairwise disjoint mincuts of G such that their union does not cover V. Our algorithm systematically enumerates PMPs and tries to extend them to a mincut partition of V with k cells. It can be viewed as generating a search tree where at the root node no mincut is chosen yet, and where on level ℓ, $1 \leq \ell < k$, exactly ℓ mincuts are in the PMPs considered. The mincut k-partitions are among the leaves on level k.

During the enumeration we repeatedly choose and test mincuts for insertion into a PMP. Of course, it is crucial that only a small part of the possibilities really has to be explored. To this end we need criteria for excluding branches of the tree from further consideration and we have to be able to decide quickly whether a mincut considered for insertion into a PMP intersects with a mincut already in the PMP.

Since partitions are sets, the order of cells is irrelevant. Therefore, let us fix a sequence (M_i) of all mincuts $M_i \in \mathcal{M}$ and let i^* be the highest index of a mincut M_i already contained in a PMP. Then, we only need to consider mincuts M_i of the sequence with index $i > i^*$ for insertion into the PMP. The generation algorithm starts by growing a PMP initialized with $\{M_1\}$. After having generated all mincut k-partitions containing M_i, but no mincut with index smaller than i, the algorithm generates all mincut k-partitions containing M_{i+1}, but no mincut with index smaller than $i + 1$.

It turns out that the choice of the sequence (M_i) is crucial for efficiency. We sort the mincuts according to non-increasing size, i.e., (M_i) is a sequence such that $|M_r| \geq |M_s|$, for $r \leq s$. This sorted sequence allows for an efficient truncation test and an efficient cactus-based test for intersection of mincuts. Inserting large mincuts early into a PMP quickly covers large portions of V in a greedy fashion and eliminates many mincuts intersecting with the large ones already chosen.

Let us consider a mincut M for insertion into a PMP. If M is not ruled out by a size-based criterion below, we test if M intersects with a mincut in the PMP (detailed in Section 5). If we realize that M should not be inserted into the PMP, then we can skip M.

We give two fast size-based criteria for the truncation of the search tree. Let J be the set of indices of the mincuts in the current PMP (relative to the fixed sequence (M_i) of mincuts) and let μ denote the minimum size of a mincut in the input sequence (M_i).

First, a mincut $M \in \mathcal{M}$ satisfying

$$|M| + \sum_{j\in J} |M_j| + (k - |J| - 1) \cdot \mu > n \tag{1}$$

can be ignored. Because we have to choose $(k - |J| - 1)$ additional mincuts on top of M to grow the current PMP into a mincut k-partition, the size of M is too large. Criterion (1) could be strengthened by, for example, replacing $(k - |J| - 1) \cdot \mu$ with the sum of the sizes of the $(k - |J| - 1)$ smallest mincuts in (M_i).

Second, we can exploit the fact that the sizes of the mincuts in the input sequence (M_i) are non-increasing. Let us consider the mincut $M_{i'}$ for insertion into the current PMP. We know that all mincuts with index $i > i'$ in the sequence have size $|M_i| \leq |M_{i'}|$. If

$$|M_{i'}| + \sum_{j\in J} |M_j| + (k - |J| - 1) \cdot |M_{i'}| < n \tag{2}$$

then we can ignore $M_{i'}$ and all mincuts in (M_i) with index $i > i'$, since the size of $M_{i'}$ is too small. Criterion (2) can be strengthened by replacing $(k - |J| - 1) \cdot |M_{i'}|$ with $\sum_{i' < i \leq i'+k-|J|-1} |M_i|$ where $M_{i'}$ can be ignored if i runs over the end of the sequence in this sum.

Theorem 1. *The algorithm for generating $\mathcal{M}_k$ for an n-vertex graph G has time complexity $O(n^{2k+1})$.*

Proof. Let $N = |\mathcal{M}|$. Assume, in the worst case, that we never truncate any branch of the search tree. On level ℓ of the full N-ary search tree there are N^ℓ nodes. This sums up to $1 + N + N^2 + \cdots + N^k = O(N^k) = O(n^{2k})$ nodes in the full N-ary tree of height k. For $\ell > 1$ we perform a test for intersection of mincuts to get from a node on level $\ell - 1$ to a node on level ℓ. A brute-force implementation of this test runs in $O(n)$. This sums up to $O(n^{2k+1})$ in total and dominates the time required for the construction of $\mathcal{H}$ from G and for building the input sequence of mincuts. □

Remark 2. There are exactly $\binom{h}{k}$ mincut k-partitions containing only mincuts represented by a given h-cycle of $\mathcal{H}$ where $2 \leq k \leq h$. If h is large and k is neither close to 2 nor close to h, we have combinatorial explosion. Long paths in G consisting of edges weighted by $\frac{\lambda}{2}$ result in long cycles in $\mathcal{H}$. In the application of mincut k-partitions to the TSP described below, there is a natural way to suppress long cycles in $\mathcal{H}$ and the proposed algorithm is fast in practice.

3.2. Building the input sequence of mincuts

An explicit sequence of the mincuts of G requires $O(n^3)$ space. Using the cactus $\mathcal{H}$, the space requirement for the input sequence (M_i) can be substantially reduced to $O(n^2)$ by describing each mincut by a pair of cactus edges as opposed to an explicit list of vertices. For double mincuts it is important to take Remark 1 into account. We want to have a double mincut only once in (M_i). The total space requirement of the generation algorithm for $\mathcal{M}_k$ (excluding the output) is dominated by the space requirement of the input sequence (M_i).

To make fast decisions in the algorithm, we need access to the size of every mincut in (M_i) in constant time. Therefore, in addition to the pair of edges describing a mincut, we store its size. Since the mincut sizes are integral and bounded, the sequence (M_i) can be sorted in $O(n^2)$ time.

We could extract each $M \in \mathcal{M}$ from $\mathcal{H}$ to determine its size. This would amount to $O(n^3)$ time. One can do better by exploiting the structure of $\mathcal{M}$ mirrored in the cactus $\mathcal{H}$. Let us assume that we can access the size of each bead of G in constant time. Then, we can efficiently determine the size of each $M \in \mathcal{M}$. Building the input sequence of mincuts takes $O(n^2)$ time and the sizes can be determined along the way without increasing this time complexity.

To achieve this, we scan $\mathcal{M}$ in the following order when building the (still unsorted) sequence. The cycles of $\mathcal{H}$ are considered one after the other. For an h-cycle with edges $c_1, \ldots, c_h$ (where $\text{head}(c_i) = \text{tail}(c_{i+1})$, for $1 \leq i < h$, and $\text{head}(c_h) = \text{tail}(c_1)$) the mincuts represented by the cycle are considered in the order $(c_1, c_2), (c_1, c_3), \ldots, (c_1, c_h), (c_2, c_3), \ldots, (c_2, c_h), \ldots, (c_{h-1}, c_h)$. A complementary mincut (c_j, c_i) is considered directly after (c_i, c_j). We may skip some mincuts of $\mathcal{H}$ according to Remark 1. To obtain the size of a mincut (c_i, c_j), the sizes of the beads between $\text{head}(c_i)$ and $\text{tail}(c_j)$ have to be summed up. The sizes can be updated in constant time when switching from one mincut to the next one.

The cactus $\mathcal{H}$ of G has $O(n)$ edges and therefore G has $O(n)$ beads. The sizes of all beads can be determined in $O(n^2)$ total time by extracting the beads from $\mathcal{H}$. Consequently, we improved from $O(n^3)$ to $O(n^2)$.

Theorem 2. *If the cactus $\mathcal{H}$ of G is available, a sequence of all $O(n^2)$ mincuts of G, each one described by a pair of directed cactus edges, together with the sizes of the mincuts, requires $O(n^2)$ space and can be computed in $O(n^2)$ time—also if the sequence has to be sorted with respect to mincut sizes.*

The beads of G are in 1–1 correspondence with the directed edges of $\mathcal{H}$ and we associate with every edge c of $\mathcal{H}$ the size of the bead at $\text{head}(c)$. We now show that the total time $O(n^2)$ mentioned above to determine the sizes of all beads can be reduced.

```
Algorithm: ComputeAllBeadSizes
Input: the canonical cactus $\mathcal{H}$ of the graph $G$
Output: size of bead of $G$ corresponding to edge $c$ of $\mathcal{H}$ in $b[c]$

for each edge $c$ of $\mathcal{H}$ do $b[c] := 0$
for each edge $c$ of $\mathcal{H}$ do
  if $b[c] = 0$ then DetermineBeadSize($c$)
end
```

```
procedure DetermineBeadSize($c$)
$b[c]$ := the number of vertices of $G$ contained in node head($c$) of $\mathcal{H}$
if head($c$) induces a mincut of $G$ (which is then a bead) then return
for each edge $e$ of $\mathcal{H}$ incident to head($c$) do
  if head($e$) = head($c$) then continue
  if $e$ belongs to the same cycle of $\mathcal{H}$ as $c$ then continue
  while head($e$) $\neq$ head($c$) do
    if $b[e] = 0$ then DetermineBeadSize($e$)
    $b[c] := b[c] + b[e]$
    $e$ := the next edge of the cycle of $\mathcal{H}$ to which edge $e$ belongs
return
```

Fig. 2. An algorithm for computing the sizes of all beads of G.

The following algorithm labels every edge c of $\mathcal{H}$ with the size of the bead at head(c) (see Fig. 2).

In this algorithm, the array b storing the bead sizes is global and all other variables are local. As opposed to the brute-force $O(n^2)$ algorithm, the recursive algorithm does not extract the beads of G from $\mathcal{H}$ to compute their sizes. The nodes of $\mathcal{H}$ inducing a mincut of G are exactly the 1-junction nodes. The remaining nodes of $\mathcal{H}$ are *cutnodes*, i.e., they disconnect $\mathcal{H}$ when removed from $\mathcal{H}$. Recall that a cutnode may be empty or not.

Because $\mathcal{H}$ has $O(n)$ edges we obtain the following result.

Theorem 3. *The sizes of all beads of G can be computed in time $O(n)$ if the cactus $\mathcal{H}$ of G is available.*

4. Reducing the length of the input sequence of mincuts

There may be mincuts $M \in \mathcal{M}$ that do not occur in any mincut k-partition and thus can be ignored. To reduce the space requirement of the input sequence (M_i) of mincuts and to speed up the generation, we do not admit a mincut $M \in \mathcal{M}$ to the sequence which cannot be in a mincut k-partition. Further, our generation framework offers the flexibility to be selective and accept only a subset of $\mathcal{M}$ as mincuts that constitute a $\mathcal{K} \in \mathcal{M}_k$.

4.1. Size-based reduction

If we do not insist on the complete set $\mathcal{M}_k$, we can use parameters $n_{\min}$ and $n_{\max}$ to exclude all $M \in \mathcal{M}$ with $|M| < n_{\min}$ or $|M| > n_{\max}$ from the sequence (M_i). This allows for the suppression of $\mathcal{K} \in \mathcal{M}_k$ having cells with unbalanced sizes. A $\mathcal{K} \in \mathcal{M}_k$ containing an $M \in \mathcal{M}$ covering most of V can be suppressed by a small $n_{\max}$. If we insist on the generation of the complete set $\mathcal{M}_k$, we set $n_{\min} := 1$ and $n_{\max} := n$. Obviously, the minimum and maximum size of a mincut in the built sequence of mincuts may be larger than $n_{\min}$ and smaller than $n_{\max}$, respectively.

After this first pass, it may be possible to drop further mincuts from (M_i). Let $s_{\min}$ ($s_{\max}$) be the sum of the sizes of the $(k-1)$ smallest (largest) mincuts in the current sequence. An $M \in \mathcal{M}$ with $|M| + s_{\min} > n$ or $|M| + s_{\max} < n$ cannot be in a mincut k-partition and can therefore be dropped from (M_i).

4.2. Structure-based reduction

As opposed to the criteria above, we now identify mincuts of G that cannot be in a mincut k-partition of G by exploiting the structure of $\mathcal{M}$ represented by $\mathcal{H}$.

Theorem 4. *Let $k > 2$ and M be a mincut of G described by the edges (c_i, c_j) of cycle $\mathcal{C}$ of the cactus $\mathcal{H}$. If the number of nodes of $\mathcal{H}$ on $\mathcal{C}$ between* head(c_j) *and* tail(c_i) *(including* head(c_j) *and* tail(c_i)*) is strictly smaller than $(k-1)$ and if all these nodes are non-empty, then M cannot be in a mincut k-partition.*

Proof. The situation is visualized in Fig. 3 where the non-empty nodes n_1 to n_r may be cutnodes $(r < k - 1)$.

(i) M is not a double mincut.

We have to use mincuts described by pairs of edges of $\mathcal{C}$ to cover the vertices of G contained in n_1 to n_r by mincuts of G not intersecting with M. The cardinality of a mincut partition containing M is maximized by choosing the r beads represented by $\mathcal{C}$ that do not intersect with M. Hence, M cannot be in a mincut τ-partition with $\tau \geq k$.

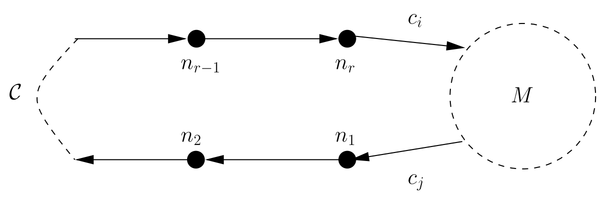

Fig. 3. The location of M in $\mathcal{H}$ forbids M to be in a mincut k-partition of G.

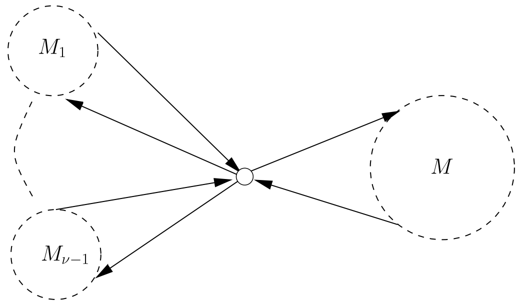

Fig. 4. An empty cutnode giving rise to an $M \in \mathcal{M}$ that cannot be in a $\mathcal{K} \in \mathcal{M}_k$.

(ii) M is a double mincut.

Then $\mathrm{head}(c_i) = \mathrm{tail}(c_j)$ is an empty 2-junction node and the vertices in n_1 to n_r can be covered by $\overline{M}$ which can be described by a pair of cycle edges not belonging to $\mathcal{C}$. We end up with $\{M, \overline{M}\} \in \mathcal{M}_2$. The other way to obtain a mincut partition containing M is detailed in (i). □

We call a mincut of G described by a pair (c_i, c_j) of edges of $\mathcal{H}$ with $\mathrm{tail}(c_i) = \mathrm{head}(c_j)$ a *blade* of $\mathrm{tail}(c_i)$. A ν-junction node of $\mathcal{H}$ has exactly ν blades ($\nu \geq 1$).

Corollary 5. *A blade $M \in \mathcal{M}$ of a non-empty node of $\mathcal{H}$ cannot be in a mincut k-partition of G with $k > 2$.*

Theorem 6. *If $M \in \mathcal{M}$ is a blade of an empty ν-junction node of $\mathcal{H}$ where $\nu > k$, then M cannot be in a mincut k-partition of G with $k > 2$.*

Proof. Fig. 4 depicts the situation ($\nu > k$).

Each $M_j \in \mathcal{M}, j = 1, \ldots, \nu - 1$, is a blade of the empty ν-junction node of $\mathcal{H}$. Apart from the unacceptable mincut 2-partition $\{M, \overline{M}\}$, there is no mincut partition containing M with fewer than the $\nu > k$ mincuts $M_1, \ldots, M_{\nu-1}, M$. □

5. A fast cactus-based test for the intersection of mincuts

In the generation algorithm, a mincut M of G considered for insertion into a PMP of G has to be tested for intersection with the mincuts already in the PMP. We could keep an n-element array updated which indicates the vertices covered by a PMP, but the obvious $O(|M|)$ test is too slow in practice. One can do better by using a sorted input sequence (M_i) of mincuts together with $\mathcal{H}$.

Because intersection tests have to be performed very frequently, tests detecting an intersection after having scanned through nearly a complete mincut are very detrimental to the overall performance. Intuitively, it is desirable to know where to look in a tested mincut for collisions with a PMP. This section gives an answer.

Keeping Remark 1 in mind, let us denote the unique mincut of $\mathcal{H}$ inducing M by M'. Whenever we insert a mincut M into a PMP, we *lock* M' in $\mathcal{H}$, i.e., we mark the nodes of M' as occupied. We unlock M' when we remove M from a PMP or from a mincut k-partition during backtracking in the search tree.

Lemma 7. *If M_a and M_b are two disjoint mincuts of G with $M_a \cup M_b \neq V$, then the two mincuts M'_a and M'_b of $\mathcal{H}$ are also disjoint.*

Proof. Since $M_a \cap M_b = \emptyset$ we know that $M'_a \cap M'_b$ can only contain empty nodes and we cannot have $M'_a \subset M'_b$ or $M'_b \subset M'_a$. Now, assume $M'_a \cap M'_b \neq \emptyset$.

If both M'_a and M'_b are obtained by removing edges of a given cycle $\mathcal{C}$ of $\mathcal{H}$, then also $M'_a \cap M'_b$ can be obtained by removing two edges of $\mathcal{C}$. However, $M'_a \cap M'_b$ induces a mincut of G included in $M_a \cap M_b = \emptyset$ and we have a contradiction.

If, on the other hand, M'_a and M'_b are obtained by removing edges of different cycles of $\mathcal{H}$, then, since $M'_a \cap M'_b \neq \emptyset$, we see that $M'_a \cup M'_b$ covers all nodes of $\mathcal{H}$. Because $\mathcal{H}$ is canonical, the only possibility is that $M'_a \cap M'_b$ consists of a single empty node and M'_a, M'_b induce two complimentary double mincuts M_a, M_b with $M_a \cup M_b = V$. Again a contradiction is obtained.

Consequently, there is no possibility for M'_a to intersect with M'_b. □

If $M_a \cap M_b = \emptyset$ but $M_a \cup M_b = V$, then M'_a and M'_b may intersect. This happens exactly in the situation where M_a and M_b are complementary double mincuts, i.e., when M_a and M_b are the two blades of an empty 2-junction node.

A mincut M' inducing M may be smaller (cactus nodes can contain several vertices) or larger (cactus nodes can be empty) than M. We show that we can focus on a 1- or 2-element subset of M' if we want to test whether M intersects with a mincut in a PMP. Let M be described by the pair (c_i, c_j) of cycle edges of $\mathcal{H}$. We call the subset $\{\text{head}(c_i), \text{tail}(c_j)\}$ of M' the *anchor* of M and denote it by $A(M)$. For example, in Fig. 1, we have $A(\{4, 18\}) = \{\text{head}(c_3)\}$.

Theorem 8. *Let $\{M_j \in \mathcal{M} \mid j \in J\}$ be a PMP of G and let exactly the pairwise disjoint mincuts $M'_j, j \in J$, of $\mathcal{H}$ be locked in the cactus $\mathcal{H}$ of G. Let M be a mincut of G with $|M| \le |M_j|$ and $M \cup M_j \ne V$ for $j \in J$. Then M intersects with some $M_j, j \in J$, in the PMP if and only if $A(M)$ contains a locked node of $\mathcal{H}$.*

Proof. Since, due to its definition, a PMP does not cover V, Lemma 7 implies that the cuts M'_j are indeed pairwise disjoint. The requirement $M \cup M_j \ne V$ excludes the situation where the PMP consists of a single double mincut having M as complement. The claimed equivalence does not hold in this situation. Let us prove both directions.

Let $j_0 \in J$ be such that $M \cap M_{j_0} \ne \emptyset$. We see that $M' \cap M'_{j_0}$ contains a non-empty node of $\mathcal{H}$. There are the following three cases.

(1) If $M' \subset M'_{j_0}$, then $A(M) \subset M'$ obviously contains a locked node.
(2) If $M'_{j_0} \subset M'$, then $|M_{j_0}| \le |M|$. Since also $|M| \le |M_{j_0}|$, we have $M = M_{j_0}$ and therefore $M' = M'_{j_0}$ so that $A(M) \subset M'$ contains a locked node.
(3) If neither of the two mincuts M' and M'_{j_0} is included in the other, then, since $M \cup M_{j_0} \ne V$, it is impossible that M' and M'_{j_0} are obtained by removing edges of different cycles of $\mathcal{H}$. Therefore, M and M_{j_0} are represented by the same cycle $\mathcal{C}$ of $\mathcal{H}$ and $M \cap M_{j_0} \ne \emptyset$ is a mincut (induced by $M' \cap M'_{j_0}$) represented by $\mathcal{C}$. We see that $A(M)$ contains a node on $\mathcal{C}$ locked by M'_{j_0}.

Now let $j_0 \in J$ be such that $A(M) \cap M'_{j_0} \ne \emptyset$. We show that M intersects with M_{j_0}. We have $M \cup M_{j_0} \ne V$. If $M \cap M_{j_0} = \emptyset$ then by Lemma 7 $M' \cap M'_{j_0} = \emptyset$ and, as $A(M) \subset M'$, also $A(M) \cap M'_{j_0} = \emptyset$. This is a contradiction, and hence M and M_{j_0} intersect. □

That is, after having checked the small anchor $A(M)$ with $|A(M)| \in \{1, 2\}$ in constant time, we know for sure whether the usually much larger mincut M intersects with a mincut which is already in the PMP. Theorem 8 is a major reason why our algorithm for generating $\mathcal{M}_k$ is efficient. We improved on the obvious $O(|M|)$ test for intersection and ended up with an $O(1)$ test.

The requirement $M \cup M_j \ne V$ in Theorem 8 is no restriction for us since we are not interested in generating mincut 2-partitions (they could easily be extracted from $\mathcal{H}$). An M with $M \cup M_j = V$ would be ruled out by the size-based criterion (1) before a test for intersection is performed. Note the importance of $|M| \le |M_j|$ which holds in our algorithm due to the sorted sequence (M_i).

There is more to the fast test for intersection than only using a sorted input sequence (M_i). An $M \in \mathcal{M}$ described by (c_i, c_j) is a union of pairwise disjoint beads. If we walk through M' in $\mathcal{H}$ via depth-first-search starting at head(c_i), we may dive into a large bead building up M which does not intersect with a PMP. We may detect an intersection only after we have switched to a further bead building up M. Traversing M' via breadth-first-search or walking along the cycle from head(c_i) to tail(c_j) is an improvement, but directly checking $A(M) = \{\text{head}(c_i), \text{tail}(c_j)\} \subset M'$ in constant time is the best we can do.

6. Efficiency of the generation algorithm in practice

We report about computational results demonstrating the efficiency of our algorithm in practice. We implemented the algorithm in C and used the compiler `gcc` version `3.3` with optimization option `-O3`. All computational results reported in this paper were obtained on a Linux PC with a single 2800 MHz Intel Pentium 4 processor and 4 GByte main memory.

Table 1 provides data for test graphs produced by a branch-and-cut algorithm for the TSP. They are equal to the graphs in Table 2 (see Section 9 for details on their origin). The number of vertices and edges of the graphs are listed. The set of double mincuts of G is denoted by $\mathcal{D} \subset \mathcal{M}$. Cactus construction times are given in CPU seconds where $t^{\text{W}}_{\mathcal{H}}$ and $t^{\text{F}}_{\mathcal{H}}$ are the times required by Wenger's [27] and Fleischer's [23] algorithm, respectively. The number of cycles (including 2-cycles) of the cacti ($\#\mathcal{C}$), the maximum length of a cycle ($h_{\max}$), and the number of empty nodes ($\#\emptyset$) are listed. We denote the length (after reduction) of the input sequence (M_i) of mincuts by ℓ_k. The generation of $\mathcal{M}_k$ took t_k CPU seconds where t_k includes building the input sequence of mincuts and the exhaustive search, but not the cactus construction.

The test graphs are sparse. Their number of mincuts is large for the relatively short cycles of the cacti and double mincuts appear frequently. The cactus construction algorithm of Fleischer [23] is slower than the one of Wenger [27], but works for more general graphs. We report results for the range $k = 6, \dots, 30$. The rationale will become clear in the sections below. The length ℓ_k of the input sequence is provided for the extreme values $k = 6$ and $k = 30$. We have $\ell_k < |\mathcal{M}|$ for G_1 to G_8. Roughly speaking, t_k tends to increase with increasing n and k, but much slower than the time complexity $O(n^{2k+1})$ in Theorem 1 suggests. The test graphs have a large number of mincut k-partitions.

Note that, for the test graphs, a minimum k-cut algorithm would return a single element of $\mathcal{M}_k$. To the best of our knowledge, the only deterministic minimum k-cut algorithm for $k > 6$ is due to Goldschmidt and Hochbaum [5] and

Table 1
Computation of all mincut k-partitions of graphs.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
n	50	91	190	402	566	3724	4190	5178
m	83	152	324	676	959	6231	6854	8493
$\|\mathcal{M}\|$	190	304	650	1412	1808	12 202	13 466	16 438
$\|\mathcal{D}\|$	8	32	48	102	66	662	774	946
$t^{\mathrm{W}}_{\mathcal{H}}$	<0.01	<0.01	0.01	0.01	0.02	0.35	0.43	0.66
$t^{\mathrm{F}}_{\mathcal{H}}$	0.01	0.01	0.15	0.07	0.14	10.15	13.21	20.40
$\#\mathcal{C}$	35	65	131	294	396	2475	2759	3423
$h_{\max}$	6	4	5	8	6	9	8	6
$\#\emptyset$	14	25	44	94	83	624	674	820
ℓ_6	118	141	309	693	775	5421	5742	6834
ℓ_{30}	92	155	337	681	786	5425	5643	6731
t_6	<0.01	<0.01	<0.01	0.01	0.01	0.43	0.48	0.67
t_7	<0.01	<0.01	<0.01	0.01	0.01	0.44	0.45	0.62
t_8	<0.01	<0.01	<0.01	0.01	0.01	0.40	0.41	0.54
t_9	<0.01	<0.01	<0.01	0.01	0.01	0.43	0.38	0.51
t_{10}	<0.01	<0.01	<0.01	0.01	0.01	0.52	0.35	0.50
t_{15}	0.04	0.01	0.04	0.03	0.03	2.87	0.72	1.17
t_{20}	0.53	0.35	1.15	0.24	0.33	48.16	6.97	12.42
t_{25}	2.22	5.65	41.54	1.65	1.46	2349.70	106.93	136.55
t_{30}	3.51	171.76	2821.34	98.37	2.39	82 035.86	5561.97	2873.42
$\|\mathcal{M}_6\|$	92	23	110	415	230	1584	1344	1304
$\|\mathcal{M}_7\|$	160	35	154	503	270	1865	1606	1569
$\|\mathcal{M}_8\|$	299	48	230	593	324	2387	1958	1864
$\|\mathcal{M}_9\|$	517	62	337	659	420	3324	2390	2253
$\|\mathcal{M}_{10}\|$	848	86	470	718	547	4731	2875	2749
$\|\mathcal{M}_{15}\|$	8341	1062	2984	1118	1845	28 045	7778	8965
$\|\mathcal{M}_{20}\|$	62 957	15 057	55 471	8397	5366	167 647	70 832	57 956
$\|\mathcal{M}_{25}\|$	328 016	168 030	578 757	48 068	46 288	2135 525	538 886	550 916
$\|\mathcal{M}_{30}\|$	986 938	1241 266	5632 322	203 597	108 378	34 942 944	4047 932	5103 056

Table 2
Mincut k-partitions for TSP support graphs.

Name	G_1 gr120	G_2 lin318	G_3 att532	G_4 pr1002	G_5 pr2392	G_6 usa13509	G_7 d15112	G_8 d18512
ℓ'_6	118	92	216	478	456	3224	3293	3979
ℓ'_{30}	92	155	337	616	628	4286	4010	4721
t'_6	<0.01	<0.01	<0.01	<0.01	<0.01	0.33	0.40	0.58
t'_7	<0.01	<0.01	<0.01	<0.01	<0.01	0.30	0.39	0.54
t'_8	<0.01	<0.01	<0.01	<0.01	<0.01	0.29	0.35	0.46
t'_9	<0.01	<0.01	<0.01	<0.01	<0.01	0.31	0.31	0.43
t'_{10}	<0.01	<0.01	<0.01	<0.01	<0.01	0.34	0.30	0.39
t'_{15}	0.04	0.01	0.03	0.02	0.02	1.48	0.54	0.92
t'_{20}	0.53	0.35	1.05	0.23	0.29	24.63	5.48	10.97
t'_{25}	2.28	5.86	39.65	1.60	1.30	1051.56	94.41	120.60
t'_{30}	3.40	175.42	2694.55	79.35	2.05	33430.70	3435.61	2758.23
$\|\mathcal{M}'_6\|$	55	20	79	176	124	975	987	1102
$\|\mathcal{M}'_7\|$	100	31	109	177	127	1175	1096	1283
$\|\mathcal{M}'_8\|$	178	43	165	200	173	1526	1349	1488
$\|\mathcal{M}'_9\|$	283	56	237	257	240	1951	1707	1737
$\|\mathcal{M}'_{10}\|$	478	81	331	322	331	2535	2107	2052
$\|\mathcal{M}'_{15}\|$	5829	948	2349	772	1106	12 913	5966	7117
$\|\mathcal{M}'_{20}\|$	47 354	13 052	36 756	6394	4173	84 931	53 921	49 479
$\|\mathcal{M}'_{25}\|$	251 093	145 949	368 150	39 282	35 387	847 946	409 578	439 153
$\|\mathcal{M}'_{30}\|$	763 634	1076 808	4004 798	182 606	79 486	11 134 216	3156 787	3871 092

requires $O(n^{k^2/2-3k/2+4})$ minimum (s,t)-cut computations. Therefore, our algorithm works for much larger k than nearly all minimum k-cut algorithms and can generate the complete set $\mathcal{M}_k$, for $k > 6$.

To enlarge the test bed, we experimented with random graphs G of the type described in Nagamochi et al. [29] and Jünger et al. [2] where these graphs served in computational studies assessing mincut algorithms. Apart from very rare exceptions, we have $|\mathcal{M}| = 2$ for such a graph—whatever parameter settings determining some graph properties are used. That is, the cactus $\mathcal{H}$ consists of exactly 2 nodes joined by a single 2-cycle so that $|\mathcal{M}_2| = 1$ and $\mathcal{M}_k = \emptyset$ for $k > 2$.

7. Application to the traveling salesman problem

We now describe how mincut k-partitions can be applied in the branch-and-cut approach to the TSP. We only give a sketch of this approach here and refer to [30] for a detailed description.

Let $G = (V_n, E_n, c)$ be the complete undirected graph with n nodes and edge weights c_e, for every edge $e \in E_n$. The *traveling salesman problem (TSP)* consists of finding a simple cycle containing every node (a so-called *tour*) such that the sum of its edge weights is as small as possible.

The TSP is a classical combinatorial optimization problem and it is NP-hard. With binary variables x_e stating whether edge e is in the tour or not, the TSP can be formulated as linear 0/1 program as follows.

$$\begin{aligned}\min\ & \sum_{e\in E_n} c_e x_e \\ & x(\delta(v)) = 2, \quad \text{for all } v \in V_n, \\ & x(\delta(S)) \geq 2, \quad \text{for all } S \subset V_n,\ |S| \geq 3,\ |S| \leq n-2, \\ & x_e \in \{0, 1\}, \quad \text{for all } e \in E_n.\end{aligned}$$

Here $\delta(S)$ denotes the set of edges with exactly one end in a cut S ($\delta(v)$ being the abbreviation for $\delta(\{v\})$) and $x(F)$ stands for $\sum_{e\in F} x_e$, for an edge set F. The equations $x(\delta(v)) = 2$ are called *degree constraints* and the inequalities $x(\delta(S)) \geq 2$ are called *subtour elimination constraints*.

The *traveling salesman polytope STSP*(n) is defined as the convex hull of all feasible solutions of this integer program, or seen differently, of the characteristic 0/1-vectors of tours in the complete graph. Theoretically the TSP now amounts to minimizing the (linear) weight function c over STSP(n). To this end a description of STSP(n) with linear equations and inequalities is needed. There has been a lot of research on this polyhedral description and many classes of inequalities have been found. Most of them, however, are too complicated to be used in practice.

The branch-and-cut approach works with linear relaxations of STSP(n). A basic relaxation, the so-called *subtour relaxation* is obtained by replacing $x_e \in \{0, 1\}$ by $0 \leq x_e \leq 1$ in the above IP and provides a lower bound on the minimum tour length. Usually the optimum solution of this linear program will be fractional. A simple branch-and-bound algorithm would now split the problem into subproblems, e.g., by fixing variables to 0 or 1, and proceed this way. In contrast, branch-and-cut tries to first strengthen the LP relaxation by adding further inequalities which are known to be valid for STSP(n).

The characteristic situation in branch-and-cut is that the current LP relaxation has been solved giving a non-integral solution x^*. For strengthening the relaxation the so-called *separation problem* has to be addressed now. It consists of finding a linear inequality which is violated by x^*, but satisfied by all points of STSP(n). *Separation algorithms* identify such inequalities and, by adding them to the LP, improve the relaxation. Since STSP(n) is not known completely, it may be the case that no such separating inequality can be found and one has to resort to some kind of branch-and-bound. The success of a branch-and-cut approach heavily depends on the possibilities to generate violated inequalities before having to start to branch.

Note that, despite the exponential number of inequalities, the subtour relaxation is polynomially solvable via the ellipsoid method. In practice, however, it is solved very efficiently in short time and for large problem instances by generating subtour constraints in a cutting-plane fashion and by using the simplex algorithm for optimizing the linear programs.

We will show that our algorithm can give an interesting contribution to solving the separation problem for large problems where standard separation algorithms might be too time-consuming to be applicable in practice. Here an approach comes into play which allows for using mincut k-partitions. The idea is to "shrink" the current x^* to a significantly smaller $\bar{x}$ representing the relevant properties of x^*, search separating inequalities for $\bar{x}$ and "lift" them back to the original problem.

So assume that the following situation is given. The current relaxation has been solved and x^* is its optimum solution. The fractional vector x^* defines the *TSP support graph* $G = (V, E, w)$ where $V = V_n$, $E = \{e \in E_n \mid x^*_e > 0\}$, and $w_e = x^*_e$, for $e \in E$. We assume that all linear constraints of the IP formulation, i.e., the so-called *degree equations* and *subtour elimination constraints* are satisfied. So, in a TSP support graph every vertex v is a mincut of weight 2 because $w(\delta(v)) = 2$ and every cut $S \subset V$ has weight $w(\delta(S)) \geq 2$.

Let $S, T \subset V$ be disjoint cuts. We denote the set of edges with one end in S and one in T by $(S : T)$ and abbreviate $w((S : T))$ to $w(S : T)$. To *shrink* a cut $S \subset V$ we remove S and $\delta(S)$, add a new vertex s replacing S, and for all $v \in V \setminus S$ with $w(S : \{v\}) > 0$ introduce a new edge $\{s, v\}$ with weight $w(S : \{v\})$. Shrinking a cut S in G results in a graph denoted by G/S. If $\mathcal{S}$ is a collection of pairwise disjoint cuts in G, then the graph obtained by shrinking these cuts is denoted by $G/\mathcal{S}$. Shrinking a mincut k-partition $\mathcal{K} \in \mathcal{M}_k$ results in a k-vertex graph $G/\mathcal{K}$.

We require the concept of safe shrinking due to Padberg and Rinaldi [31]. In [31] sufficient conditions for 1- and 2-shrinkability are given where all involved sets S and T are mincuts.

A cut $S \subset V$ in an n-vertex TSP support graph G is called 1-*shrinkable* if the weight vector of G/S is outside STSP$(n-|S|+1)$ given that the weight vector of G is outside STSP(n), i.e., S is 1-shrinkable if there is a TSP cutting-plane after shrinking S given that there was one before.

Analogously, a pair $\mathcal{S} = \{S, T\}$ of disjoint cuts in G is called 2-*shrinkable* if there is a TSP cutting-plane for $G/\mathcal{S}$ given that there is one for G.

Repeatedly shrinking 1-shrinkable sets and 2-shrinkable pairs is widely applied to reduce redundancy in TSP support graphs. Cuts computed for a shrunk graph are lifted to the original space. We could shrink any PMP or mincut partition in a

TSP support graph to obtain a smaller TSP support graph. This shrinking is not necessarily safe, i.e., the weight vector may move inside the TSP polytope corresponding to the small graph.

The two separation methods below require small shrunk graphs. The required number of vertices usually cannot be reached by safe shrinking alone. In both methods k-cuts in a support graph are shrunk to obtain small k-vertex graphs.

In the *small instance relaxation* (SIR) method [38], linear descriptions of polytopes associated with small problem instances are scanned in search for inequalities violated by $\overline{x}^*$ where $\overline{x}^*$ is the weight of a small k-vertex graph. The linear description of STSP(k) by inequalities is trivial for $k < 6$ and not completely known for $k > 10$. For the SIR method described in [32,33] the range of k is 6 to 10.

The local cut method of Applegate et al. [34,35] tries to compute a cutting-plane separating a vector $\overline{x}^*$ from STSP(k) by applying linear programming and polyhedral theory where k ranges from 6 to about 30 or 40.

Both methods search for cuts of the form $f^T\overline{x} \geq f_0$ with $f \geq 0$ (called tight-triangular form in [36]). To violate $f^T\overline{x} \geq f_0$ by $\overline{x}^*$ it is advantageous to have a low total weight $\sum \overline{x}^*_e$. The lowest possible total weight in a k-vertex graph obtained from a TSP support graph is achieved by shrinking a mincut k-partition. In this sense mincut k-partitions are the ideal k-cuts to be shrunk. (Neither of the two methods relies on $\overline{x}^*$ being the weight vector of a TSP support graph.) Shrinking a mincut k-partition is not always safe, therefore we try many of them. In our experiments they were indeed superior to other k-cuts [33].

8. The generation in the TSP case

A TSP support graph emerging in a branch-and-cut run usually has numerous mincut k-partitions. To keep the number manageable, we apply safe shrinking before we start generating mincut k-partitions. In [31] the following sufficient condition for 1-shrinkability is proven.

Theorem 9. *Let $G = (V, E, w)$ be a TSP support graph and $S \subset V$ a mincut with $2 \leq |S| \leq |V| - 2$. Let $t \in V \setminus S$ with $w(\{t\} : S) = 1$ and $W := V \setminus (S \cup \{t\})$. If the weight vector of G/W is in STSP $(|V| - |W| + 1)$, then S is 1-shrinkable.*

We call an edge having weight 1 a *1-edge*. A *1-path* in a TSP support graph is an inclusionwise maximal simple path consisting of 1-edges not spanning V. Using Theorem 9 we see that 1-paths can safely be shrunk to single 1-edges by repeatedly shrinking 1-edges. This is important since 1-paths in G form cycle segments in $\mathcal{H}$ and long cycles in $\mathcal{H}$ result in an exploding number of mincut k-partitions.

All sets S with $2 \leq |S| \leq 3$ and the property that there is a $t \in V \setminus S$ with $w(\{t\} : S) = 1$ are 1-shrinkable. The condition in Theorem 9 is automatically satisfied and such a set S can be detected by simple means. In the literature, larger sets S are not shrunk. They are harder to find and checking the condition is time-consuming. We note that all candidates S for Theorem 9 can be extracted from the cactus by scanning its cycles for beads $\{t\}$. For an h-cycle of $\mathcal{H}$ with $h \geq 3$, the weight in G between consecutive beads of $\mathcal{H}$ is 1 and between non-consecutive beads it is 0 (from Lemma 10). For $4 \leq |S| \leq 7$ the condition can, e.g., be checked by scanning the known complete linear descriptions of STSP(k) for $6 \leq k \leq 9$. Considering S with $4 \leq |S| \leq 7$ usually leads to a slight further reduction compared to only shrinking S with $2 \leq |S| \leq 3$, see [33].

A *1-square* in a support graph is a pair $\{S, T\}$ of 1-edges with $w(S : T) > 0$. A 1-edge in a TSP support graph is a mincut. To reduce redundancy further, we also shrink all 2-shrinkable 1-squares satisfying the sufficient condition of [31] and note that candidates can be detected using the cactus.

When we generate $\mathcal{M}_k$ for a TSP support graph G, we assume that no further S with $2 \leq |S| \leq 3$ or 1-square can safely be shrunk as described above.

Safe shrinking can help in a further way to suppress undesirable $\mathcal{K} \in \mathcal{M}_k$. In a TSP branch-and-cut context we start with $k = 6$. Only if the k-vertex support graphs obtainable by shrinking mincut k-partitions in G have been processed, we move on to the next higher k. Let us be satisfied with some instead of all cutting-planes per k-vertex graph $G^<$. Assume that $G^<$ can be safely shrunk to $G^\lll$. If there is a cutting-plane for $G^<$, then also for $G^\lll$.

The graph $G^<$ can be ignored if some kind of safe shrinking is applicable:

(i) If $G^\lll$ has less than 6 vertices, then there is neither a cutting-plane for $G^\lll$ nor for $G^<$ (degree equations and variable bounds remain satisfied when mincuts are shrunk in TSP support graphs). $G^<$ can therefore be ignored.
(ii) If $G^\lll$ has at least 6 vertices, then we may be able to find cutting-planes for $G^\lll$. However, for $G^\lll$ cutting-planes have already been searched since $G^\lll$ has already been processed earlier. Therefore, $G^<$ can be ignored.

We speed up the generation of mincut k-partitions by skipping exactly those $\mathcal{K} \in \mathcal{M}_k$ yielding graphs $G^< = G/\mathcal{K}$ having a 1-path of length at least 2.

First, our algorithm can be modified such that it backtracks as soon as three pairwise disjoint and consecutive mincuts represented by the same cycle would be chosen. This would result in a 1-path of length at least 2 in $G^<$. In particular, this skips $\mathcal{K} \in \mathcal{M}_k$ resulting in a tour when shrunk. ($G/\mathcal{K}$ is a tour if all $M \in \mathcal{K}$ are represented by the same cycle of $\mathcal{H}$.) Second, the following theorem shows how the input sequence (M_i) of mincuts can be reduced further in the TSP case.

Two cuts in G are said to *cross* if they intersect, neither of the two is included in the other, and their union does not cover V. We will use a well-known result on crossing cuts.

Lemma 10. *If $A, B \in \mathcal{M}$ cross in G, then the following holds.*

(i) *$M^1 := A \cap \overline{B}$, $M^2 := A \cap B$, $M^3 := \overline{A} \cap B$, and $M^4 := \overline{A \cup B}$ are mincuts.*

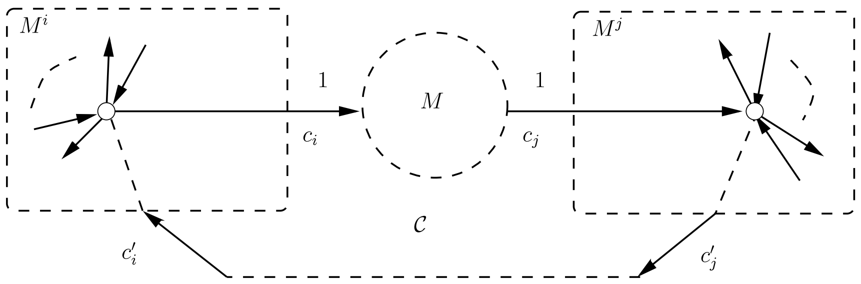

Fig. 5. Shrinking a $\mathcal{K} \in \mathcal{M}_k$ with $M \in \mathcal{K}$ produces a 1-path of length at least 2.

(ii) $w(M^1 : M^3) = w(M^2 : M^4) = 0$.
(iii) $w(M^1 : M^2) = w(M^2 : M^3) = w(M^3 : M^4) = w(M^4 : M^1) = \frac{\lambda}{2}$.

Theorem 11. *Let $k > 2$ and M be a mincut of the TSP support graph G described by the pair (c_i, c_j) of edges of $\mathcal{H}$. Let* $\text{tail}(c_i) \neq \text{head}(c_j)$ *and* $\text{tail}(c_i)$ *be either non-empty or an empty ν-junction node with $\nu > k - 1$. Let the same be true for* $\text{head}(c_j)$. *Then $G/\mathcal{K}$ has a 1-path of length at least 2 for every mincut k-partition $\mathcal{K} \in \mathcal{M}_k$ containing M.*

Proof. Fig. 5 depicts an exemplary situation. Let $M \in \mathcal{K} \in \mathcal{M}_k$. Since $k > 2$, the mincut of G described by (c_j, c_i) covering $\overline{M}$ cannot be in $\mathcal{K}$.

If $\text{head}(c_j)$ is non-empty, then we have to cover the vertices of G contained in $\text{head}(c_j)$ by a mincut M^j of G described by c_j and a further edge $c'_j \neq c_i$ of cycle $\mathcal{C}$. The other mincuts covering the vertices in $\text{head}(c_j)$ intersect with M.

Now assume that $\text{head}(c_j)$ is an empty ν-junction node with $\nu > k - 1$. Let $B \in \mathcal{M}$ be the bead represented by $\mathcal{C}$ at the head of c_j. We could cover B by a mincut (c_j, c'_j). If we want to cover B by other mincuts not intersecting with M, we need at least $\nu - 1$ of them, as they correspond to the $\nu - 1$ blades of $\text{head}(c_j)$ not intersecting with M. In addition to these $\nu - 1$ mincuts, we also have M in $\mathcal{K}$ and at least one more mincut since $\text{tail}(c_i) \neq \text{head}(c_j)$. Since $\nu + 1 > k$, we have to cover B by a mincut (c_j, c'_j) which we denote by M^j.

For $\text{tail}(c_i)$ the argument is analogous. We have to cover the bead of G at the tail of c_i by a mincut (c'_i, c_i) denoted by M^i where c'_i follows c'_j on $\mathcal{C}$.

The two mincuts (c'_i, c_j) and (c_i, c'_j) cross in G. Using Lemma 10 we obtain $w(M^i : M) = w(M : M^j) = \frac{\lambda}{2} = 1$ and $G/\mathcal{K}$ contains a 1-path of length at least 2. The vertex resulting from shrinking M is an inner vertex of the 1-path. □

The cutnodes of cacti of TSP support graphs are empty. They are often ν-junction nodes with $\nu > k$ and Theorem 6 excludes many mincuts from (M_i).

9. Computational results for the TSP

The test graphs G_1 to G_8 in Table 2 are the same graphs as in Table 1. They are TSP support graphs obtained at the root node of the branch-and-cut tree for problem instances from the TSP benchmark library TSPLIB ([37]). Subtour elimination constraints have been separated exactly and in addition simple comb inequalities [31] heuristically. All 1-shrinkable sets S with $2 \leq |S| \leq 3$ and 2-shrinkable 1-squares satisfying the sufficient conditions in [31] have recursively been shrunk to obtain G_i.

The figures in Table 2 are for the generation algorithm which skips exactly the mincut k-partitions yielding a shrunk graph with a 1-path of length 2 or longer (compare Section 8). Due to Theorem 11, the length ℓ'_k of the input sequence (M_i) is smaller than ℓ_k in Table 1. In theory, ℓ'_k can be quadratic in n. In practice, ℓ'_k seems to be linear in n for safely shrunk graphs. The generation times t'_k of the algorithm modified for the TSP skipping some undesirable partitions are, due to the added backtracking criterion, sometimes slightly higher than t_k in Table 1, but usually clearly smaller. We have $\mathcal{M}'_k \subset \mathcal{M}_k$. No time-consuming separation algorithm has to be called for a small k-vertex graph corresponding to a mincut k-partition in $\mathcal{M}_k \setminus \mathcal{M}'_k$. The range of mincut 30-partitions generated per second was 333 for G_6 to 224,598 for G_1.

By applying all criteria for safe shrinking, many of the small k-vertex graphs corresponding to $\mathcal{M}'_k$ can be skipped instead of calling separation algorithms for them. The saved separation time usually outweighs the time required for checking safe shrinkability by far.

Remark 3. If there is a candidate for a 1-shrinkable set S of any size in Theorem 9, then there is a 1-shrinkable set. If S is not 1-shrinkable, then W is. We do not have to check whether a weight vector is in the TSP polytope.

In a TSP support graph, a mincut k-partition usually contains mincuts of size 1, i.e., if we exclude all mincuts of size 1 from the input sequence (M_i), only very few mincut k-partitions are found.

This paper suggests an algorithm for generating the set of all partitions of the vertex set of a graph with positive edge weights into a fixed number k of globally minimum weight cuts. It is an exhaustive search algorithm in which the cactus representation of mincuts helps in several ways to make the search time and space efficient. For fixed k, the running time is polynomial in the number of vertices. But, more importantly, the algorithm is efficient in practice which is demonstrated by computational results.

It was also shown how mincut k-partitions can help in the small instance relaxation and local cut methods to find cutting-planes for the TSP. In both methods many k-cuts in support graphs are shrunk and in some sense mincut k-partitions are the ideal k-cuts. For TSP support graphs, the desirable mincut k-partitions can be generated particularly fast.

Minimum and near-minimum k-cuts are generalizations of mincut k-partitions. An efficient deterministic generation of such k-cuts remains a challenge.

References

[1] C.S. Chekuri, A.V. Goldberg, D.R. Karger, M.S. Levine, C. Stein, Experimental Study of minimum cut algorithms, in: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 1997, pp. 324–333.
[2] M. Jünger, G. Rinaldi, S. Thienel, Practical performance of efficient minimum cut algorithms, Algorithmica 26 (2000) 172–195.
[3] E.A. Dinitz, A.V. Karzanov, M.V. Lomonosov, On the structure of a family of minimal weighted cuts in a graph, in: A.A. Fridman (Ed.), Studies in Discrete Optimization, Nauka, Moscow, 1976, pp. 290–306. original article in Russian, English translation from www.loc.gov or www.nrc.ca/cisti.
[4] D.R. Karger, C. Stein, A new approach to the minimum cut problem, J. ACM 43 (4) (1996) 601–640.
[5] O. Goldschmidt, D.S. Hochbaum, A polynomial algorithm for the k-cut problem for fixed k, Math. Oper. Res. 19 (1) (1994) 24–37.
[6] H. Saran, V.V. Vazirani, Finding k-cuts within twice the optimal, SIAM J. Comput. 24 (1) (1995) 101–108.
[7] S. Kapoor, On minimum 3-cuts and approximating k-cuts using cut trees, in: LNCS, vol. 1084, Springer, 1996, pp. 132–146.
[8] L. Zhao, H. Nagamochi, T. Ibaraki, Approximating the minimum k-way cut in a graph via minimum 3-way cuts, J. Comb. Optim. 5 (4) (2001) 397–410.
[9] D.S. Hochbaum, D.B. Shmoys, An $O(|V|^2)$ algorithm for the planar 3-cut problem, SIAM J. Algebraic Discrete Methods 6 (1985) 707–712.
[10] X. He, An improved algorithm for the planar 3-cut problem, J. Algorithms 12 (1) (1991) 23–37.
[11] M. Burlet, O. Goldschmidt, A new and improved algorithm for the 3-cut problem, Oper. Res. Lett. 21 (5) (1997) 225–227.
[12] H. Nagamochi, T. Ibaraki, A fast algorithm for computing minimum 3-way and 4-way cuts, Math. Program., Ser. A 88 (2000) 507–520.
[13] H. Nagamochi, S. Katayama, T. Ibaraki, A faster algorithm for computing minimum 5-way and 6-way cuts in graphs, J. Comb. Optim. 4 (2) (2000) 151–169.
[14] M.S. Levine, Fast randomized algorithms for computing minimum {3, 4, 5, 6}-way cuts, in: Proceedings of the 11th Annual ACM-SIAM Symposium On Discrete Algorithms, 2000, pp. 735–742.
[15] Y. Kamidoi, S. Wakabayashi, N. Yoshida, A divide-and-conquer approach to the minimum k-way cut problem, Algorithmica 32 (2002) 262–276.
[16] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, M. Yanakakis, The complexity of multiterminal cuts, SIAM J. Comput. 23 (4) (1994) 864–894.
[17] W.-C. Yeh, A simple algorithm for the planar multiway cut problem, J. Algorithms 39 (1) (2001) 68–77.
[18] G. Calinescu, H. Karloff, Y. Rabani, An improved approximation algorithm for multiway cut, J. Comput. Syst. Sci. 60 (3) (2000) 564–574.
[19] D.R. Karger, P. Klein, C. Stein, M. Thorup, N.E. Young, Rounding algorithms for a geometric embedding of minimum multiway cut, in: Proceedings of the 31th annual ACM Symposium on Theory of Computing, ACM Press, 1999, pp. 668–678.
[20] W.H. Cunningham, L. Tang, Optimal 3-terminal cuts and linear programming, in: G. Cornuéjols, R.E. Burkard, G.J. Woeginger (Eds.), 7th International IPCO Conference, Graz, Austria, June 9–11, 1999, Proceedings, in: LNCS, vol. 1610, Springer, 1999, pp. 114–125.
[21] D. Naor, V.V. Vazirani, Representing and enumerating edge connectivity cuts in $\mathcal{RNC}$, in: F. Dehne, J.-R. Sack, N. Santoro (Eds.), Algorithms and Data Structures, 2nd Workshop, WADS'91, Ottawa, Canada, August 1991, Proceedings, in: LNCS, vol. 519, Springer, 1991, pp. 273–285.
[22] A. De Vitis, The cactus representation of all minimum cuts in a weighted graph, Tech. Rep. 454, Istituto di Analisi dei Sistemi ed Informatica (IASI), Roma, Italy, 1997.
[23] L. Fleischer, Building chain and cactus representations of all minimum cuts from hao-orlin in the same asymptotic run time, J. Algorithms 33 (1) (1999) 51–72.
[24] H. Nagamochi, Y. Nakao, T. Ibaraki, A fast algorithm for cactus representations of minimum cuts, Japan J. Ind. Appl. Math. 17 (2) (2000) 245–264.
[25] H. Nagamochi, T. Kameda, Canonical cactus representation for minimum cuts, Japan J. Ind. Appl. Math. 11 (3) (1994) 343–361.
[26] A.V. Karzanov, E.A. Timofeev, Efficient algorithm for finding all minimal edge cuts of a nonoriented graph, Cybernetics 22 (1986) 156–162. Plenum Publishing Corporation. Translated from Kibernetika, 2, 1986, pp. 8–12 (in Russian).
[27] K.M. Wenger, A new approach to cactus construction applied to tsp support graphs, in: W.J. Cook, A.S. Schulz (Eds.), Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, Proceedings, in: LNCS, vol. 2337, Springer, 2002, pp. 109–126.
[28] K.M. Wenger, Kaktus-Repräsentation der minimalen Schnitte eines Graphen und Anwendung im Branch-and-Cut Ansatz für das TSP, Diploma thesis, University of Heidelberg, 1999.
[29] H. Nagamochi, T. Ono, T. Ibaraki, Implementing an efficient minimum capacity cut algorithm, Math. Program. 67 (1994) 325–341.
[30] D. Applegate, R. Bixby, V. Chvátal, W. Cook, The Traveling Salesman Problem: A Computational Study, in: Princeton Series in Applied Mathematics, 2006.
[31] M. Padberg, G. Rinaldi, Facet identification for the symmetric traveling salesman polytope, Math. Program. 47 (1990) 219–257.
[32] G. Reinelt, K.M. Wenger, Small instance relaxations for the traveling salesman problem, in: D. Ahr, R. Fahrion, M. Oswald, G. Reinelt (Eds.), Operations Research Proceedings 2003, Selected Papers of the International Conference on Operations Research, OR 2003, Springer, 2004.
[33] K.M. Wenger, Generic cut generation methods for routing problems, Ph.D. Thesis, University of Heidelberg, Shaker (2003).
[34] D. Applegate, R. Bixby, V. Chvátal, W. Cook, TSP cuts which do not conform to the template paradigm, in: M. Jünger, D. Naddef (Eds.), Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions, in: LNCS, vol. 2241, Springer, 2001, pp. 261–303.
[35] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Implementing the Dantzig–Fulkerson–Johnson algorithm for large traveling salesman problems, Math. Program., Ser. B 97 (2003) 91–153.
[36] D. Naddef, G. Rinaldi, The graphical relaxation: A new framework for the symmetric traveling salesman polytope, Math. Program. 58 (1) (1993) 53–88.
[37] TSPLIB, A library of benchmark instances for the traveling salesman problem, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95.
[38] T. Christof, G. Reinelt, Combinatorial optimization and small polytopes, Top 4 (1) (1996) 1–64. ISSN 1134–5764.