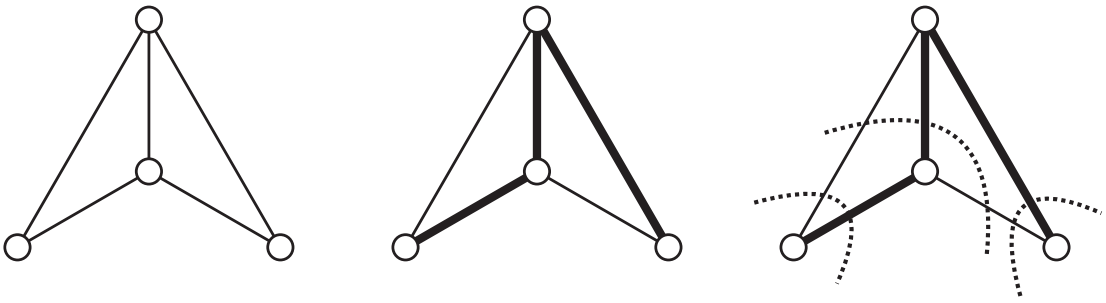
Minimum Fundamental Cut Basis Problem

Diploma Thesis
by
Anne Margarete Schwahn



Supervisor: Prof. Dr. Horst W. Hamacher

Department of Mathematics
Technical University of Kaiserslautern

August 2005

Die Mathematiker sind eine Art Franzosen:
Redet man zu ihnen, so übersetzen sie es in ihre Sprache,
und dann ist es alsbald ganz etwas anderes.
J. W. von Goethe

Contents

List of Figures

List of Tables

Affidavit

I, hereby, declare that the following diploma thesis "Minimum Fundamental Cut Basis Problem" has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

_____

Place, Date, Signature

Chapter 1

Preface

1.1 Overview

This thesis deals with a problem coming from graph theory. In a given graph, any spanning tree defines a fundamental cut basis. In the Minimum Fundamental Cut Basis Problem, (MinFCutB), the tree has to be chosen such that the weight of the corresponding basis is minimized. This problem is closely related to the Minimum Fundamental Cycle Basis Problem, (MinFCycB), which in turn has been investigated more thorough so far.

The necessary tools, e.g. definitions, theorems, and the relation to linear algebra, are introduced in Chapter 2. The NP-hardness of the problem is proved in Section 2.4. I present the idea of an application in Section 2.6. Several formulations for (MinFCutB) are given in Chapter 3. As the problem cannot be solved to optimality but for very small instances, we try to get the best lower and upper bounds possible. Lower bounds can be obtained by the relaxations in Chapter 4. The main focus of this thesis are the heuristics presented in Chapter 5. Finally, I will compare the performance of these algorithms in Chapter 6.

One incentive to solve the (MinFCutB) is the saving concerning computations. The cut basis contains information on all the cuts in the graph. Its corresponding tree can be stored in a highly compact manner.

1.2 Übersicht

Diese Diplomarbeit behandelt ein in der Graphentheorie angesiedeltes Problem. In einem gegebenen Graphen definiert jeder beliebige spannende Baum

eine fundamentale Schnittbasis. Im minimalen Fundamentalschnittproblem, (MinFCutB), wird ein Baum gesucht, der das Gewicht der entsprechenden Basis minimiert. Das Problem ist eng mit dem Fundamentalkreisproblem verwandt, das schon ausführlicher untersucht wurde.

Die notwendigen Vorkenntnisse, also Definitionen, Sätze und der Bezug zur Linearen Algebra werden in Kapitel 2 eingeführt. Die Zugehörigkeit zur Klasse der NP-schweren Probleme wird in 2.4 bewiesen, eine mögliche Anwendung findet sich in 2.6. Einige Formulierungen für (MinFCutB) werden in Kapitel 3 vorgestellt. Da das Problem nur für die kleinsten Instanzen optimal lösbar ist, wird versucht, bestmögliche untere und obere Schranken zu finden. Untere Schranken erhält man durch die Relaxierungen in Kapitel 4, obere durch die Heuristiken in Kapitel 5. Abschließend wird die Leistungsfähigkeit der vorgestellten Verfahren in Kapitel 6 untersucht.

Ein Anreiz zur Lösung des Problems liegt in der Programmierung. Eine Schnittbasis enthält Informationen über alle Schnitte im Graphen. Der entsprechende Baum kann kompakt gespeichert werden.

1.3 Thanks!

I want to take this opportunity to thank several people supporting me during the last months. First of all, I am very grateful to my parents for their love, trust, and money. I thank Prof. Dr. Horst W. Hamacher for his supervision, a wonderful topic, and some liberties as regards content. Special thanks go to Florentine Bunke for being an exceptional tutor. It was a pleasure to work with Prof. Francesco Maffioli from the Politecnico di Milano. And last but not least, sincere thanks are given to my family and friends for correction and distraction.

Chapter 2

Basics

2.1 Definitions and results in graph theory

2.1.1 Graphs, cuts and cycles

Graphs and matrices

Definition 2.1. An *undirected graph* $G = (V, E)$ consists of a nonempty set V of *vertices* and a set E of *edges*, which are defined by their respective endpoints. Let $n = |V|$ and $m = |E|$. A graph is called *finite* if $n, m < \infty$ and *simple* if it has neither loops nor parallel edges.

Definition 2.2. Two vertices are called *adjacent* if there is an edge connecting them. An *adjacency matrix* $A = A(G)$ of a given *graph* G is defined by its entries

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else.} \end{cases}$$

Definition 2.3. A vertex and an edge are called *incident* if the vertex is an endpoint of the edge. An *incidence vector* I_e, $e \in E$ has

$$i^{th} component = \begin{cases} 1 & \text{if } i \text{ endpoint of } e \\ 0 & \text{else.} \end{cases}$$

An *incidence matrix* $B = B(G)$ is an $n \times m$ matrix with columns $(I_e)_{e \in E}$. Given a vertex v, its *edge-neighbourhood* $\delta(v)$ is defined as all edges incident to v. The *degree* of v is the number of edges incident to v, i.e. $|\delta(v)|$.

Remark 2.4. The rows of the incidence matrix of a graph G are linearly dependent over $GF(2)$, as any row i can be represented as a linear combination of the other rows, namely $I_{i\bullet} = (\sum_{j \neq i} I_{j\bullet}) \mod 2$. Deleting any one row of the incidence matrix yields a linearly independent reduced incidence matrix having $n - 1$ rows.

Definition 2.5. A *subgraph* G' of G consists of $V' \subseteq V$ and $E' \subseteq E$, where the edges of E' are incident to the vertices of V'. It is *spanning* if $V' = V$.

Definition 2.6. A *path* P with *length* l^P in a graph is a sequence of vertices and edges

$$P = (v_0, e_1, v_1, e_2, \ldots, e_{l^P}, v_{l^P}),$$

where edge e_i is incident to vertices v_{i-1} and v_i. In case of simple graphs it can also be written as sequence of adjacent vertices

$$P = (v_0, v_1, \ldots, v_{l^P}).$$

A *simple path* is a sequence of distinct vertices and edges.

Cycles

Definition 2.7. A *cycle* is a closed path, i.e. $v_0 = v_{l^P}$. A graph G is *acyclic* if it does not contain any cycle. An *elementary cycle* is a closed simple path.

Remark 2.8. A cycle is a set of edges such that every vertex of the graph is incident with an even number of edges in this cycle. An elementary cycle is a connected cycle having at most two edges incident to any vertex. Thus cycles are the (possibly empty) union of edge-disjoint elementary cycles ([ALMM04]).

Definition 2.9. A graph is *connected* if every pair of vertices is connected by a path. It is *k-(vertex-)connected* if it is still connected after deleting any $k - 1$ vertices. Accordingly, a graph is *k-edge-connected* if it takes at least $k-1$ edges to disconnect it. A vertex and an edge whose removal disconnects the graph is called *cut-vertex* and *bridge*, respectively.

Remark 2.10. A graph is k-connected if and only if there exist at least k paths between any two vertices and these paths do not have any vertex in common except their endpoints. It is k-edge-connected if and only if there are at least k edge-disjoint paths between any pair of vertices.
Both connectivities are bounded from above by the smallest degree in the graph, $\min_{v \in V} |\delta(v)|$.

If there is a set of k edges whose removal disconnects the graph, one could choose one endpoint per edge and obtain a (not necessarily minimal) disconnecting set of vertices. Hence, the vertex-connectivity is not greater than the edge-connectivity.

Remark 2.11. Throughout this thesis a graph is considered to be undirected, simple, finite and biconnected if not stated otherwise.

Example 2.12. A simple graph with four vertices and five edges is depicted on the left of Figure 2.1.
Its corresponding matrices are:

$$AM(G) = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad IM(G) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$P = (v_1, v_3, v_2, v_4)$ is a simple path in G with length 3, $C = (v_1, v_3, v_4, v_1)$ is a cycle. G is biconnected. G' on the right is a subgraph of G, it is not spanning, as v_2 is not in G.

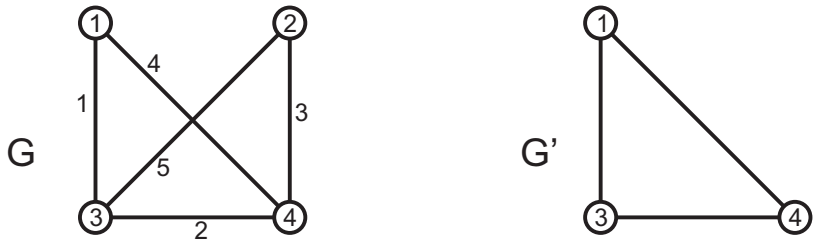


Figure 2.1: Graph and subgraph

Cuts

Definition 2.13. A *cut* in a connected graph G is a subset $Q \subset E$ such that $G \setminus Q := (V, E \setminus Q)$ is not connected. An *elementary cut* in G is a minimal cut Q, i.e. there is no smaller set being still disconnecting. It can be written as a partition of the nodes $Q = (U, V \setminus U)$, where $U \subseteq V$ and $(U, V \setminus U)$ is the set of all edges with one endpoint in U and the other one not U. U and $V \setminus U$ are called *shores*. A *nodal cut* is a cut disconnecting one vertex v_i from the other vertices, i.e. $Q = (\delta(v_i)) = (\{v_i\}, V \setminus \{v_i\})$.

Remark 2.14. An elementary cut can be referred to by the edges it contains as well as by the partition of vertices, depending on the context.

Example 2.15. On the left hand side in Figure 2.2, we see a cut Q_1 (dashed lines) disconnecting the shores $U = (v_1, v_3, v_4)$ and $V \setminus U = (v_2)$. An elementary cut Q_2 disconnecting these shores is shown on the right side.

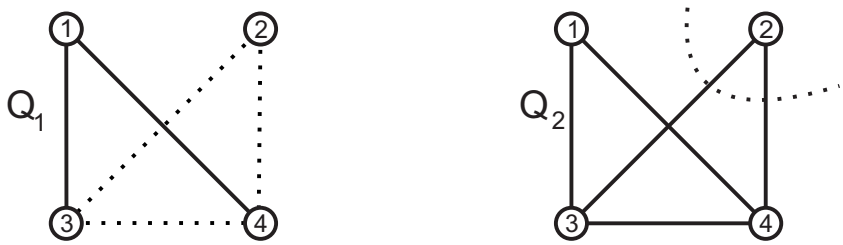


Figure 2.2: Cut and elementary cut

Cycles and cuts

Remark 2.16. Given a general graph G, there may be edges which are not contained in any cycle. If G consists of a simple non-closed path, none of its edges is contained in any cycle. On the other hand, there is never an edge in a graph which is not contained in any cut, since for any edge one can always define a partition of the vertices which has the endpoints of a this edge in different shores.

Theorem 2.17. *Given a graph G, the number of elements in the intersection of an arbitrary elementary cut and an arbitrary elementary cycle in G is even.*

Proof. e.g. in [SR61]: Let Q be an elementary cut, C an elementary cycle in G. Q partitions the graph into two shores whose vertex sets are denoted by U and $V \setminus U$. If Q and C have no elements in common, the theorem is proved. If their intersection is nonempty, then order the vertices of C such that successive vertices are joined by an edge of C. C contains vertices of both shores. Starting the cycle in U, we only get to the vertices of $V \setminus U$ by crossing the "border" from U to $V \setminus U$, an edge of Q. To get back to U again (which is necessary to close the cycle), the border is crossed again using another edge of the cut. This element cannot be the same as the first one since the cycle is elementary. In case of another "shore-switch" there are again two edges being contained in both, C and Q. □

2.1.2 Trees and fundamental cuts

Definition 2.18. A *tree* $T = (V', E')$ of G is an acyclic, connected subgraph of G. It is a *spanning tree* if $V' = V$. The edges of a given tree T are referred to as *branches*, the edges of the *co-tree* $G \setminus T$ as *chords*.

Remark 2.19. If not mentioned otherwise a tree is considered to be spanning throughout this thesis.

Definition 2.20. A *forest* $F = \{T_1, \ldots, T_K\}$ is a set of trees $T_k = (V_k, E_k)$ whose vertex sets are disjoint, i.e. the components of F are trees. It is a *spanning forest* if $V = \bigcup_{k=1}^{K} V_k$.

Definition 2.21. Given a tree T, a *fundamental cut* t_e is defined by a branch $e \in T$ and consists of the edges in G whose endpoints are in different shores of $T \setminus e$, the forest separated by e.

Remark 2.22. The fundamental cut defined by edge e_i will be denoted by t_{e_i} or t_i. Each fundamental cut contains exactly one branch of the spanning tree.

Example 2.23. Given a graph, every tree gives a set of fundamental cuts. In Figure 2.3, we see the fundamental cut defined by e_5. Removing e_5 from the tree (in bold lines), we get two shores, namely $X = (v_1, v_3, v_4)$ and $\overline{X} = (v_2)$. If we want to separate these shores in the underlying graph, the corresponding fundamental cut t_5 is $Q = (e_3, e_5)$.

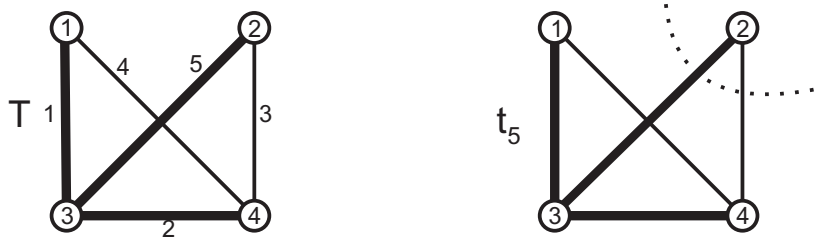


Figure 2.3: Tree and fundamental cut

Definition 2.24. The *nullity* (or *cyclomatic number*) ν of a connected graph G is given by $\nu = m - n + 1$. In general graphs it is $\nu = m - n + p$ where p is the number of connected components in G. The *rank* r of a connected graph G is given by $r = n - 1$, in the general case $r = n - p$.

Theorem 2.25. *[Properties of trees] Let a graph $G = (V, E)$ be given, $n = |V|$, then the following hold for any tree T (e.g. in [ALO93], [HK00]):*

1. *T consists of n-1 edges*
2. *T has at least 2 leaves (vertices with degree 1)*
3. *Any two vertices $u, v \in V$ of T are connected by a unique path, P_{uv}^T*
4. *The deletion of any branch $e = (u, v) \in T$ of T yields a disconnected graph consisting of two subtrees T_1 and T_2. Edges whose endpoints belong to different subtrees constitute the fundamental cut defined by e*
5. *Adding any edge $\neq e$ in this fundamental cut to T_1 and T_2 gives rise to another tree*
6. *There are r fundamental cuts corresponding to T (as T has r branches)*
7. *Adding any chord $f = (u, v) \in G \setminus T$ to T creates a unique cycle (*fundamental cycle*), namely $C_f = f + P_{uv}^T$*
8. *The deletion of any edge $\neq f$ in this fundamental cycle gives again a tree*
9. *There are ν fundamental cycles corresponding to T (as T has ν chords)*

Definition 2.26. The set of all fundamental cuts corresponding to a tree T of G is called *fundamental cut set.*

Theorem 2.27. *[Cayley's formula] There are n^{n-2} spanning trees in a complete graph K_n [WC04].*

Theorem 2.28. *The number of trees in a general graph G is given by the recursive formula*

$$\tau(G) = \tau(G - e) + \tau(G \setminus e),$$

where $G - e$ is the graph after deleting edge $e = (u, v)$, $G \setminus e$ is the graph where u and v are "merged" [WC04].

2.1.3 Minimum Fundamental Cut Basis Problem

Definition 2.29. A *weight function* $w : E \to \mathbb{R}_+$ assigns a *weight* w_{ij} to each edge $e = (i, j) \in E$ of a graph.

Definition 2.30. The *cut weight* of a cut t_e, w_{t_e}, is the total weight of the edges contained in the fundamental cut t_e. The *total cut weight* corresponding to a tree T, denoted by $TCW(T)$, is the sum of all cut weights, i.e. $TCW(T) := \sum_{e \in T} w_{t_e}$.

The problem considered in this thesis is called the *Minimum Fundamental Cut Basis Problem* (MinFCutB). In a given biconnected graph, we want to find a tree minimizing the total cut weight, i.e.

$$\min_{\text{T: T tree in G}} \sum_{e \in T} w_{t_e}.$$

2.1.4 Cut space/ Relation to linear algebra

This section deals with an operation on cuts and motivates the expression "basis" in (MinFCutB). Several results of linear algebra and graph theory can be interpreted as special cases of results in matroid theory. Fundamental cut/ cycle sets in the context of matroids are worked on in [GH02] and [Sys81]. It proves to be quite useful to have a closer look at matrices, vectors, linear independence and some more concepts of linear algebra. Those structures are of use concerning the implementations.

Incidence vectors, cut matrix and symmetric difference

Definition 2.31. For a subset Q of the set of edges E, the *(edge-)incidence vector* χ_Q is given by

$$(\chi_Q)_i = \begin{cases} 1 & \text{if } e_i \in Q \\ 0 & \text{else.} \end{cases}$$

The *vertex-incidence* vector χ_U for a subset U of the set of vertices V is defined analogously.

Cuts can be represented as edge-incidence binary vectors in $\mathbb{F}_2^m$ or vertex-incidence vectors in $\mathbb{F}_2^n$. $\mathbb{F}_2^m$ is another expression for the power set of edges, $P(E)$. If we express the weight function as a weight vector $w := (w_e)_{e \in E} \in \mathbb{R}_+^m$, the cut weight of a cut is equal to the product of its incidence vector and the weight vector.

Definition 2.32. Let G be a graph with edges $e_1, \ldots, e_m$ and cuts $Q_1, \ldots, Q_t$. The *cut matrix* $\mathfrak{Q} = (q_{ij}) \in Mat(t \times m, \mathbb{F}_2)$ of G is defined by

$$q_{ij} := \begin{cases} 1 & \text{if cut } Q_i \text{ contains edge } e_j \\ 0 & \text{else,} \end{cases}$$

i.e. the rows of $\mathfrak{Q}$ are the incidence vectors of the cuts of G. A *fundamental cut matrix* $\mathfrak{Q}_f$ of G is a special cut matrix where the rows correspond to a fundamental cut set [Bun06].

Definition 2.33. The *symmetric difference* Δ of two sets A and B is defined as $A \Delta B := (A \backslash B) \cup (B \backslash A)$. The symmetric difference of sets A_i, $i \in I$ is defined as $\underset{i \in I}{\Delta} A_i$. The symmetric difference Δ of two incidence vectors χ_{Q_1} and χ_{Q_2} is the mod 2 sum of these vectors, i.e. the addition of χ_{Q_1} and χ_{Q_2} in $\mathbb{F}_2^m$.

Definition 2.34. A set of cuts $Q_1, Q_2, \ldots, Q_k$ is called *independent* if none of these cuts can be represented by the symmetric difference of other cuts of this set, i.e. the corresponding incidence vectors are linearly independent in $\mathbb{F}_2^m$.

Example 2.35. The incidence vectors of the cuts in Figure 2.4 are:

$$Q_1 = (\ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ )$$
$$Q_2 = (\ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ )$$

These two cuts are combined by the symmetric difference and add up to

$$Q_3 = (\ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ )$$

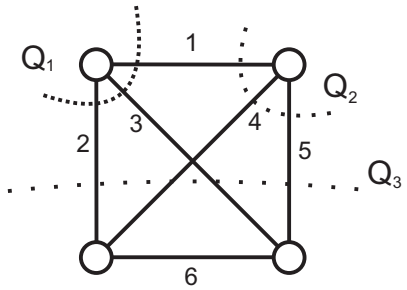


Figure 2.4: Incidence vectors and their symmetric difference

Cut Space

Definition 2.36. The set of cuts ($\emptyset$, elementary cuts and unions of edge-disjoint cuts) and its incidence vectors respectively constitute the *cut space* $\mathfrak{Cut}$.

Theorem 2.37. $\mathfrak{Cut}$ *together with the symmetric difference constitute a vector space over the two-element field* $\mathbb{F}_2$.

Proof. $\mathfrak{Cut}$ is closed under Δ, i.e. the symmetric difference of two elementary cuts is either again an elementary cut or a disjoint union of elementary cuts, i.e. a cut. If $Q_1 = (U, \overline{U})$, $Q_2 = (W, \overline{W})$ then $Q_1 \Delta Q_2 = (U \Delta W, V \backslash (U \Delta W))$. (see [Gib85]).
The symmetric difference is commutative and associative. The identity element is the zero vector, the inverse of a vector is the vector itself.
$\mathfrak{Cut}$ is closed under scalar multiplication with scalars 0 and 1. Scalar multiplication is associative, 1 is the multiplicative identity.
Furthermore, scalar multiplication distributes over scalar and vector addition respectively. □

Remark 2.38. $\mathfrak{Cut}$ is a linear subspace of the power set of edges, $P(E)$.

Cut Basis

Definition 2.39. A *cut basis* $\mathfrak{B}$ is an independent set of cuts which generate the whole cut space of a graph G. This is the case if their incidence vectors form a basis of $\mathfrak{Cut}$.

Theorem 2.40. *Let* $\mathfrak{B} = \{Q_1, \ldots, Q_{n-1}\}$ *be a cut basis of a graph* G *then every cut* Q *in* G *has a unique representation in* $\mathfrak{B}$ *as*

$$Q = \epsilon_1 Q_1 \Delta \epsilon_2 Q_2 \Delta \ldots \Delta \epsilon_{n-1} Q_{n-1}, \quad \epsilon_i \in \{0, 1\}.$$

[Bun06]

Theorem 2.41. *Given a graph* G *and a spanning tree* T*, the set of fundamental cuts corresponding to* T *constitutes a cut basis, the so-called fundamental cut basis of* G *defined by* T*. Denoting the fundamental cuts by* $t_1, \ldots, t_{n-1}$ *and their respective branches by* $e_1, \ldots, e_{n-1}$*, any cut* Q *has the unique representation:*

$$Q = \Delta_{e_i \in Q} t_i. \tag{2.1}$$

Proof. (e.g. in [Gib85]):
Independent: The set of fundamental cuts is independent as each cut t_i contains its "own" branch e_i. And the 1-entry in χ_{t_i} which corresponds to e_i cannot be generated by the combination of two or more other fundamental cuts having 0-entries for e_i. Consequently, a representation by fundamental cuts is unique if it exists.

Spanning: To show: Each cut Q in G can be expressed by a combination of the fundamental cuts $t_1, \ldots, t_{n-1}$ as in equation (2.1).
Assume $Q \neq \Delta_{e_i \in Q} t_i$, let $Q' := \Delta_{e_i \in Q} t_i$, then $Q \Delta Q' \neq \emptyset$. Now, Q is by definition a cut and Q' is a cut as it is the combination of cuts. Hence, $Q \Delta Q'$ is a cut as well. But Q and Q' contain exactly the same branches, consequently, their symmetric difference consists of nothing but chords. Any cut has to contain at least one branch of any tree. Otherwise, it does not disconnect the graph. This contradicts the assumption.
Consequently, $\{t_1, \ldots, t_{n-1}\}$ forms a basis of the cut space. □

Corollary 2.42. *The cut space of a graph with n vertices has dimension $n-1$.*

As the fundamental cuts corresponding to a spanning tree of a graph form a basis of the cut space, the problem is called Minimum Fundamental Cut *Basis* Problem (which is the same as Minimum Set of Fundamental Cuts). Another description of the (MinFCutB) is given in [Bun06]:

$$\min\{\sum_{j=1}^{n-1} w\chi_{Q_j} | \{Q_j\}_{j=1,\ldots,n-1} \text{ fundamental cut basis of } G\},$$

where $w := (w_e)$ is the weight vector.

Nodal cuts constitute a basis as well:

Lemma 2.43. *Given a graph G and vertices $v_i \in V$, $i = 1, \ldots, n$, any set of $n-1$ nodal cuts constitutes a cut basis, the so-called nodal cut basis of G. Denoting the nodal cuts by $Q_{v_1}, \ldots, Q_{v_n}$, a cut $Q = (U, V \setminus U)$ has unique representation: $Q = \Delta_{v_i \in U} Q_{v_i}$.*

Proof. see [Bun06] □

Remark 2.44. The incidence vectors $\chi_{Q(v_i)}$ of the nodal cuts are the rows of the incidence matrix. Hence, the cut space is the row space of the incidence matrix.

Example 2.45. In Figure 2.5, we see an example of a fundamental cut basis on the left and a nodal cut basis on the right. The incidence vectors of the fundamental cuts are:

$$I_{t_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} I_{t_3} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} I_{t_5} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

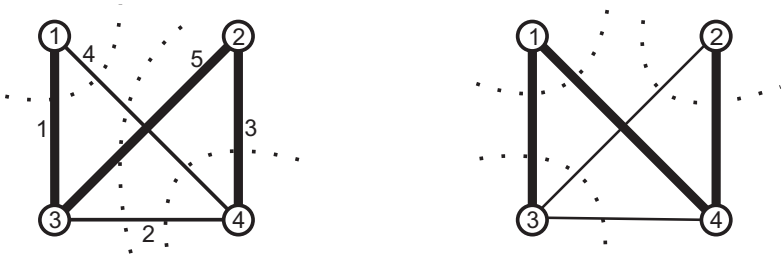


Figure 2.5: Fundamental cut basis and nodal cut basis

2.1.5 Fundamental cut sets and bases of the cut space

Remark 2.46. The set of fundamental cuts gives a basis of the cut space. However, the reverse does not hold, there are bases of the cut space which do not consist of fundamental cut sets, as can be seen in Example 2.47 [ALMM03a].

Example 2.47. This example shows a basis of a cut space which is not fundamental. The incidence vectors of the cuts depicted in Figure 2.6 are:

$$\begin{array}{lcccccccccc} Q_1 = (& 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\ Q_2 = (& 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 &) \\ Q_3 = (& 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 &) \\ Q_4 = (& 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 &) \\ Q_5 = (& 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 &) \end{array}$$

These five cuts (in a graph with six vertices) are independent of each other and thus constitute a basis. However, there is no spanning tree defining them, i.e. they do not build a fundamental cut set.

On certain conditions a cut basis is a fundamental cut basis as well (see [Sys79] or [Sys81] for fundamental cycles):

Theorem 2.48. *A cut basis $\mathfrak{B} = \{Q_1, \ldots, Q_k\}$ of G is a fundamental cut basis of G if and only if each cut $Q_i \in \mathfrak{B}$ contains an edge which does not belong to any other cut of the cut basis.*

Proof. in [BHMS05]

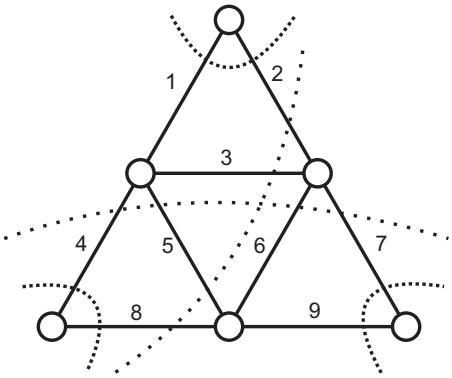


Figure 2.6: Non-fundamental basis of the cut space

$\Rightarrow$ Let $\mathfrak{B}$ be a fundamental cut basis of G, each cut $Q_i \in \mathfrak{B}$ is defined by a branch of the tree T, say e_i. This branch is not contained in any other cut of the cut basis.

$\Leftarrow$ Let $\mathfrak{B}$ be a cut basis of G where each cut $Q_i \in \mathfrak{B}$ contains at least one edge which is not contained in any other cut of the cut basis, one of these "unique" edges is denoted by e_i, E^* is the set of all e_i.
To show: E^* is a spanning tree generating $\mathfrak{B}$
As the dimension of the cut space is $n-1$, $\mathfrak{B}$ and thus also E^* have $n-1$ elements.
There is no cycle in E^*. Assume there is a cycle in E^* which includes e_i, then the intersection of cut Q_i and this cycle would contain only one element , namely e_i. An intersection with odd cardinality contradicts Theorem 2.17. Consequently, E^* is acyclic and constitutes a spanning tree.

□

2.1.6 Trees and cuts

In [SR61] one can find another definition and some more properties of cuts:

Definition 2.49. Given a graph $G = (V, E)$, an *elementary cut* is a subset of E such that removing these edges from G reduces the rank of G by 1, provided that no proper subset of this set reduces the rank of G by 1 when it is removed from G.

Remark 2.50. An elementary cut partitions a graph into two disconnected shores, i.e. the number of components p increases by 1, thus rank $r(= n - p)$ decreases by 1. Nodal cuts are not elementary in general, it may happen that they decrease the rank of a graph by more than one, as can be seen in Figure 2.7. The nodal cut defined by vertex v decreases the rank of the graph by 3.

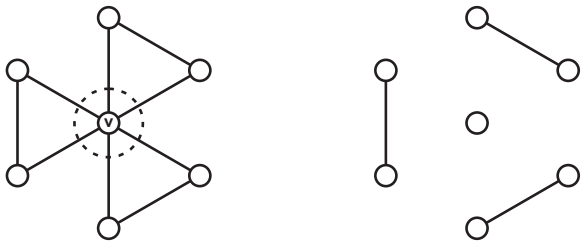


Figure 2.7: Nodal cut and resulting components

Theorem 2.51. *Given a graph G and a vertex $v \in V$, $\delta(v)$ is an elementary (nodal) cut provided that v is not a cut-vertex of G. Any bridge is an elementary cut (by itself).*

Theorem 2.52. *Every cut contains at least one branch of every tree T.*

Stronger version of Theorem 2.52 (elegant characterization of cuts):

Theorem 2.53. *Q is an elementary cut if and only if Q is a minimal set of edges which contain at least one branch of every tree.*

Theorem 2.54. *Let G be a graph, T a tree in G. For any branch $e \in T$, there is exactly one elementary cut (namely t_e) having only edge e in common with T [Jun94].*

2.1.7 Biconnectivity, cut-vertices and separability

As mentioned before, graphs are assumed to be biconnected throughout this thesis. What happens if a graph does not have to be biconnected but connected?

Dealing with fundamental cycles in general graphs, there may be edges which are not contained in any cycle (see Remark 2.16). Therefore, the cycle space does not cover all the edges [Lib05].

Example 2.55. Let G be the graph in Figure 2.8. The basis of the cut space consists of the vectors:

$$C_1 = (1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0)$$
$$C_2 = (0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1)$$

and it can easily be seen that edge (v_3, v_4) is not covered.

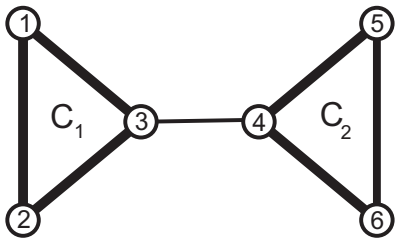


Figure 2.8: Fundamental cycles not covering the whole space

However, when it comes to fundamental cuts there is no edge which is not contained in any elementary cut (see again Remark 2.16), the cut space covers all the edges.

Theorem 2.56. *Given an arbitrary undirected graph G, the optimal spanning tree concerning (MinFCycB) (or spanning forest in case of an unconnected graph) is equal to the direct sum of the optimal spanning trees of the biconnected components of G [ALMM04].*

Conjecture 2.57. *"Preprocessing": When a general graph G has to be investigated, it can be decomposed by deleting bridges (particularly leaves and their associated edges). Any fundamental cut defined by a bridge contains no other edge than the bridge itself. The remaining components can be decomposed into biconnected components. That means that the potential cut-vertices are part of several biconnected components. It suffices to solve the (MinFCutB) in the distinct components. The solution tree of the original graph is build of the bridges and the solution trees of the components, the objective function value is the sum of the values of the components and the weights of the bridges.*

Example 2.58. The general graph on the left of Figure 2.9 can be decomposed by deleting edge e and splitting the connected component at the cut-vertex, this decomposition is depicted on the right of the figure. This example also visualizes another observation: Even if all the bridges of a graph are deleted, the remaining components are not necessarily biconnected. There can still be cut-vertices.

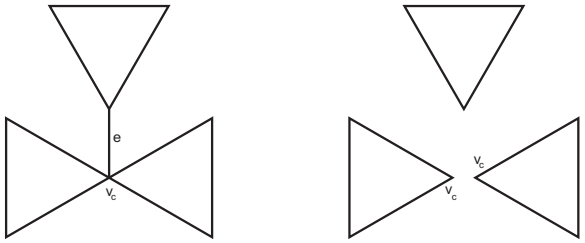


Figure 2.9: General graph and its decomposition

When it comes to programming and the simulation of test graphs the biconnectivity of a graph is not that easy to establish. One idea [Lib05] is to generate a graph $G = (V, E)$ and to add a Hamiltonian cycle, e.g. $C_H = \{v_1, v_2, \ldots, v_{n-1}, v_n, v_1\}$, to its edges. Then, any two vertices are connected by at least two vertex-disjoint paths, namely the two parts of C_H.

Biconnectivity and separability

There are some more concepts related to biconnectivity:

Definition 2.59. A graph G is *non-separable* if every subset of edges of G has at least two incident vertices in common with its complement. G is *separable* if it is not non-separable.
Given a connected and separable graph G and v_c is the single vertex in common between subgraph G' and its complement, then v_c is called *cut-vertex* of G.

The following theorem establishes the connection of the definitions above:

Theorem 2.60. *Let G be a connected graph, then:*

1. *G is non-separable if and only if it does not contain any cut-vertex.*
2. *v_c is a cut-vertex of G if and only if there exist two vertices v_a and v_b ($\neq v_c$) such that every path from v_a to v_b contains v_c.*

Proof. see [SR61] □

Corollary 2.61. *A graph G is biconnected if and only if it is non-separable.*

Proof.

G is biconnected
$\Leftrightarrow$ at least two vertices have to be removed to disconnect G
$\Leftrightarrow$ G has no cut-vertex
$\Leftrightarrow$ G is non-separable.

□

Remark 2.62. A graph whose vertices have degree at least 2 is not necessarily biconnected as can be seen in Figure 2.8.

2.2 Fundamental cycles

This section describes a problem which is similar to the (MinFCutB). Instead of cuts, it deals with cycles. Again, we have a biconnected, simple and undirected graph with a weight function, the *weight of a cycle* is the sum of the weights of its edges, the *total weight* of a cycle basis is the sum of the weights of all cycles in this basis. If each of the cycles of a cycle basis is uniquely defined by a chord (corresponding to a given spanning tree) and the unique path in the tree connecting the two endpoints of the chord, the cycle basis is *fundamental*.

A set of fundamental cycles constitutes a basis of the cycle vector space $\mathfrak{Cyc}$. It is defined over $\mathbb{F}_2$ as the null space of the incidence matrix of G. Thus, it has dimension $m - rank(B) = m - n + 1 = \nu$.

The problem of finding a set of minimum fundamental cycles in a given graph, (MinFCycB), has been investigated earlier and more detailed by quite a number of researchers from all over the world, e.g. Maciej M. Syslo, Narsingh Deo, Joseph Douglas Horton, Edoardo Amaldi, Giulia Galbiati, Leo Liberti, Francesco Maffioli and Nelson Maculan. Bases of fundamental cycles were first mentioned by Sunaram Seshu, Myril Reed [SR61] and Claude Berge [Ber62]. This section is meant to give a short overview over their work, as the (MinFCycB) is quite similar to the (MinFCutB) in many aspects. The NP-hardness of the (MinFCycB) will be proved in Section 2.4.

Beside the mathematical interest, another motivation for the research in this topic is its use in computations. For instance in the field of electrical networks, the amount of work depends on the cycle basis chosen. A (near-) optimal resp. short cycle basis may speed up the algorithm [Hor87].

2.2.1 Relation to fundamental cuts

The following properties concerning the relationship of cuts and cycles can be found in [SR61].

Theorem 2.63. *Given a graph G and a tree T,*

- *the fundamental cut defined by branch $e \in T$ contains exactly those chords of G for which e is in each of the fundamental cycles determined by these chords and*
- *the fundamental cycle defined by chord $f \in T$ contains exactly those branches of G for which f is in each of the fundamental cuts determined by these branches.*

The following characterization of the structure of cuts is the reverse of Theorem 2.17.

Theorem 2.64. *Given a graph G, any nonempty set Q of edges of G such that Q has an even number of elements in common with every cycle is an elementary cut or a cut.*

Lemma 2.65. *Let A and $B = B_1 \Delta \dots \Delta B_t$ be given. Then $|A \cap B|$ is odd $\Leftrightarrow |A \cap B_i|$ is odd for an odd number of B_i [Bun06].*

Corollary 2.66. *Let Q be a nonempty subset of E and assume that for every cycle C_i of a cycle basis $\mathfrak{B} = \{C_1, \dots, C_\nu\}$ of G, the intersection $|C_i \cap K|$ has even cardinality. Then Q is a cut of G [Bun06].*

Proof. Let C be an arbitrary cycle in G with basis representation $C = \Delta_{i=1}^{\nu} \epsilon_i C_i$ in $\mathfrak{B}$. Since $|C_i \cap Q|$ is even for every basis element C_i, by Lemma 2.65, $|C \cap Q|$ is also even. Hence Q is a cut by Theorem 2.64. □

Thus in order to model a cut Q, it is sufficient to require that the intersection of Q with the elements of an arbitrary basis of the cycle space should have an even number of elements.

The equivalence of (MinFCutB) and (MinFCycB) is given under certain conditions:

Lemma 2.67. *For a planar graph G the (fundamental) cut basis problem and the (fundamental) cycle basis problem are equivalent.*

Proof. Let G be a plane graph, G^* its dual. Then

$$\begin{aligned} & Q \subset E(G) \text{ is a cut in } G \\ \Leftrightarrow \quad & Q^* = \{e^* | e \in Q\} \subset E(G^*) \text{ cycle in } G^* \\ \Rightarrow \quad & \text{cut space of } G = \text{cycle space of } G^* \\ \Rightarrow \quad & \text{(F)CutB of G solvable as (F)CycB of } G^* \end{aligned}$$

[Bun06] □

Remark 2.68. Note that the dual of a planar graph may have multiple edges. This is not the case if each vertex has degree greater than two.

2.2.2 Algorithms

An algorithm to find a fundamental cycle set in a graph is given in [Pat69]. This procedure grows a spanning tree and simultaneously finds the cycles, its output is not intended to be minimal regarding the total cycle length. The algorithm has complexity $O(n^\gamma)$ where $2 \leq \gamma \leq 3$.
"Traditional" tree-growing algorithms (Static Degree Sort, Dynamic Degree Selection, "Unexplored" Edges, and Multi-point Breadth First Search) can be found in [DPK82].
In order to solve the (MinFCycB), several (meta-)heuristics are tested in [LAMM03] and [ALMM04], e.g. an adaption of Paton's algorithm, a local search, a variable neighbourhood search, and a tabu search. Another approach is the so called C-order heuristic which is based on the idea that a good spanning tree should contain vertices having big star sizes. The algorithm determines the barycenter of an undirected graph. It supposes that branches on the outside of the graph create longer fundamental cycles.

2.2.3 Applications

Application fields of short fundamental cycles are

- Periodic timetable planning – PESP Periodic Event Scheduling Problem, the number of integer variables in the problem can be minimized by means of a short fundamental cycle basis of a graph [Lie03]
- VLSI Design– Very Large Scale Integration (the process of placing thousands (or hundreds of thousands) of electronic components on a single chip[1])

[1] www.webopedia.com

- Electrical networks – cycles have been mentioned in this field for the first time by Kirchhoff [Kir47]
- Organic chemistry – coding of ring compounds [Led68] [Jr.65]
- Graph theory – determination of the isomorphism of graphs [CR77]
- Frequency analysis of computer programs [Knu68]
- Biopolymers [GS99]
- Electrical circuit testing [BP01]
- Generating minimal perfect hash functions [DKP95]

2.2.4 Minimum Cycle Basis Problem

If we do not claim fundamentality of the cycle basis, the resulting problem can be solved in polynomial time. J. D. Horton gives an $O(m^3n)$ algorithm of in [Hor87] and A.M.H. Gerards, J.C. De Pina and A. Schrijver in [GDPS02] improve the complexity to with $O(m^3 + mn^2logn)$.

The problem of finding a minimum cycle basis in regular matroids is solvable in polynomial time. Generalized to binary matroids it becomes NP-hard [GH02].

2.3 Tree graph

The following subsection is based on the work of M. M. Sysło ([Sys79], [Sys81], [Sys82]) who investigates fundamental cycles.

Definition 2.69. Given a graph G, the *tree graph* of G, $T(G)$, consists of the distinct spanning trees of G as vertices. Two vertices of $T(G)$ are joined by an edge if the corresponding spanning trees have $n-2$ edges in common. Two spanning trees are called *adjacent* if their corresponding vertices are adjacent in $T(G)$.

Remark 2.70. The tree graph is connected, i.e. for any T', T'' there exists a sequence $T_1, \ldots, T_k$ such that $T' = T_1$, $T'' = T_k$ and T_i is adjacent to T_{i+1} for all $i = 1, \ldots, k-1$. The subgraph of all minimum spanning trees (concerning the (MinFCutB)) does not have to be connected in general.

Definition 2.71. A *Hamiltonian cycle* is a cycle containing each vertex exactly once.

Theorem 2.72. *Every tree graph of a graph with at least 3 vertices contains a Hamiltonian cycle.*

Proof. (by Cummins [Cum66], Kishi and Kajitani) □

Example 2.73. In Figure 2.10 we can see a graph G, its corresponding tree graph $T(G)$, and a Hamiltonian cycle in $T(G)$ (bold dashed line).

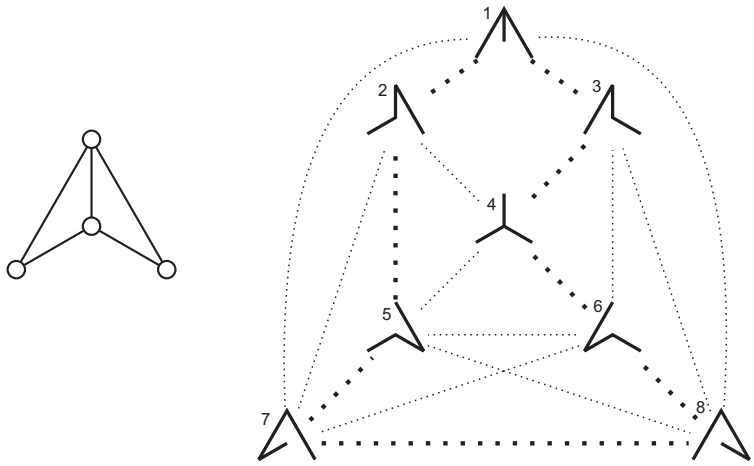


Figure 2.10: Graph, tree graph and Hamiltonian cycle

According to Theorem 2.72 it is possible to generate all spanning trees of G sequentially without repetition by using a transformation called "Edge Swap" (see Section 5.3.1). So the solution of (MinFCutB) could be found on the walk through the Hamiltonian cycle. This observation does not seem to be of great use in practice, as there are too many different spanning trees to be considered (see e.g. Cayley's formula, Theorem 2.27) and there is no information about the sequence of edge swaps in a Hamiltonian cycle.

Remark 2.74. Conjecture of Hubicka and Sysło (see e.g. [Deo79]):
A spanning tree T is a minimum fundamental cycle spanning tree in G if no spanning tree at distance 1 has total cycle length less than that of T. The distance between two spanning trees is defined as the number of edges present in one tree but not in the other. (The length of the shortest path connecting the corresponding vertices in $T(G)$.)

This conjecture is not true. Example 2.75 (given for fundamental cycles in [Deo79]) shows that the conjecture is also not true for fundamental cuts and their total cut weights.

Example 2.75. The spanning tree on the left of Figure 2.11 has total cut weight 17 and is thus a locally minimum spanning tree (as the neighbourhood has total cut weights 17, 18, 19), the right one is of distance 3 and has total cut weight 15.

Figure 2.11: Example disproving the Conjecture of Hubicka and Syslo

As the conjecture of Hubicka and Sysło does not hold, a locally optimal solution of (MinFCutB) or (MinFCycB) is, in general, not globally optimal. A local search by edge swaps (see Section 5.3.1) may get stuck before reaching optimality.

Correspondence of Spanning Trees and Fundamental Cut Sets

The tree graph also helps to visualize a point where fundamental cut sets and fundamental cycle sets cannot be treated analogously:

Remark 2.76. In Figure 2.10 we can observe two things:

- Adjacent spanning trees may generate the same fundamental cycle set (e.g. vertices 6 and 7) but that does not hold in general (vertices 7 and 8).
- A given fundamental cycle set can be generated by several different and non-adjacent spanning trees (vertices 1 and 4).

Consequently, there is no 1-1 correspondence between trees and fundamental cycle sets in general graphs unless the graph fulfills certain requirements:

Theorem 2.77. *Every pair of different spanning trees of G generates different fundamental cycle sets if and only if each 2-connected component of G is 3-edge connected [Sys82].*

Remark 2.78. Given a graph G and a tree T, it is possible that a branch e of T is element of only one fundamental cycle C. If we remove e from T and add the unique chord f of C to the tree, we obtain the same fundamental cycle set. Observe that in this case the graph is not 3-edge-connected as the removal of e and f disconnects the tree.

Lemma 2.79. *Given a simple graph G and a tree T, any edge $f = (u, v)$ of the co-tree $E \setminus T$ is contained in at least two fundamental cuts.*

Proof. The path connecting u and v in T, P_{uv}^T, has length greater than or equal to 2. The path exists, as there is a path in the tree between any pair of vertices. And it cannot have length 1, as this would imply a multiple edge which contradicts the condition of a simple graph. Edge f is contained in any fundamental cut defined by the edges on P_{uv}^T (see Lemma 2.63). Consequently, f is contained in at least two fundamental cuts. □

Theorem 2.80. *Given a general graph G, arbitrary spanning trees T_1 and T_2 generate different fundamental cut sets $\mathfrak{B}(T_1)$ and $\mathfrak{B}(T_2)$ if and only if the spanning trees are not the same.*

Proof. To show: $T_1 \neq T_2 \Leftrightarrow \mathfrak{B}(T_1) \neq \mathfrak{B}(T_2)$.

- $\Leftarrow$ Assume $T_1 = T_2$ then obviously $\mathfrak{B}(T_1) = \mathfrak{B}(T_2)$.
- $\Rightarrow$ Assume that $\mathfrak{B}$ is a given fundamental cut set and $\mathfrak{B}$ can be generated by different spanning trees.
  We know by Theorem 2.48 that each incidence vector has at least one 1-entry which is not contained in the other incidence vectors, the entry which corresponds to the branch defining the fundamental cut. There are two cases for the number of "unique" 1-entries:

  1. There is only one unique 1-entry in each incidence vector. In this case, the set of incidence vectors determines the tree completely. The basis cannot be generated by another spanning tree.
  2. Assume there is an incidence vector which has more than one unique 1-entry. In this case one could choose either of the corresponding edges and get different spanning tress generating $\mathfrak{B}$. But if we choose one of those edges as a branch for this fundamental

cut, the other one becomes a chord and is element of only one cut. This contradicts Lemma 2.79. Consequently, the incidence vector of any fundamental cut has exactly one 1-entry which is not contained in any other incidence vector of the fundamental cut set (case 1).

□

Remark 2.81. Note that the proof of Theorem 2.80 strengthens the statement of Theorem 2.48: Each cut of a fundamental cut basis contains exactly one edge which does not belong to any other cut of the cut basis.

Notation 2.82. For a given graph G, let $\tau(G)$ denote the number of distinct spanning trees in G, $\psi(G)$ the number of distinct fundamental cut sets and $\varphi(G)$ the number of distinct fundamental cycle sets.

Corollary 2.83.

$$\tau(G) = \psi(G) \geq \varphi(G)$$

Proof. According to Theorem 2.80, there is a 1-1 correspondence between spanning trees and fundamental cut sets, both sets have same cardinality. This establishes the equality.
The inequality follows from Remark 2.76, since the mapping from the set of spanning trees to the set of fundamental cycle sets may not be injective. □

2.4 Complexity

N. Deo conjectures in [Deo79] that finding a tree which generates a minimum set of fundamental cycles is NP-complete. He proves the NP-completeness of this problem in [DPK82]. Following his proof, it will be shown that (MinFCycB) is equivalent to (MinFCutB) in certain cases and that the ladder problem is NP-complete itself, this part is taken from [BHM05].

Remark 2.84. The following modification of (MinFCycB) is investigated in [Gal01] and proved to be NP-hard as well:

$$\min_{\text{T: T tree in G}} \max_{e \in T} w_{t_e}$$

The decision version of (MinFCycB) is formally stated as follows:
Given a graph $G = (V, E)$ and a positive integer L, is there a spanning tree T

of G such that the sum of the lengths of all fundamental cycles corresponding to T does not exceed L?

This problem will be polynomially reduced from the shortest-total-path-length-spanning-tree problem (STPLS), which is known to be NP-complete [JLRK78] and stated as follows:
Given a graph $G = (V, E)$ and a positive integer K is there a spanning tree T of G such that the sum over all pairs of vertices $u, v \in V$ of the path length in T from u to v is no more than K?

Before we start proving the theorem, we need the following definitions:

Definition 2.85. A *complete chain* between a pair of vertices u, v such that $(u, v) \notin E$ is defined as a set of vertices $(u, v, 1), (u, v, 2), \ldots, (u, v, n^4 - 1)$ and a set of edges $(u, (u, v, 1)), ((u, v, 1,), (u, v, 2)), \ldots, ((u, v, n^4 - 1), v)$.

Example 2.86. Figure 2.12 depicts a graph and its complete chain in dashed lines.

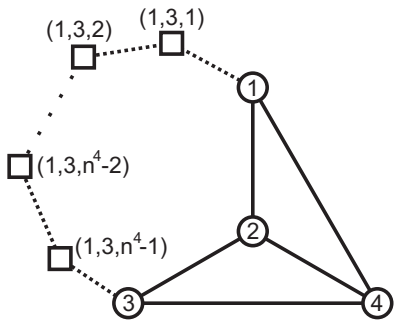


Figure 2.12: Graph with complete chain

Definition 2.87. A *1-off chain* is obtained by deleting exactly one edge from a complete chain.

Lemma 2.88. *The (STPLS) is polynomially reducible to the (MinFCycB).*

The following proof is given because the construction introduced will be useful concerning heuristic approaches.

Proof. Let $G = (V, E)$ be an undirected graph with n vertices and m edges. We want to find the tree which solves the (STPLS) in G.

To this end, a supergraph $H = (V_1, E_1)$ of G is constructed as follows:

$$\begin{aligned} V_1 &= V \cup \{(u,v,i) | (u,v) \notin E, i = 1,2,\ldots,n^4-1\} \\ E_1 &= E \cup \{(u,(u,v,1))\} \\ &\quad \cup \{((u,v,i),(u,v,i+1)) | i = 1,2,\ldots,n^4-2\} \\ &\quad \cup \{((u,v,n^4-1),v)\} \end{aligned}$$

for all pairs $u, v \in V$ which are not adjacent.
The number of unordered pairs of non-adjacent vertices is $r = \frac{n(n-1)}{2} - m$. Consequently, r complete chains with length $n^4 - 1$ have to be added in the supergraph H which thus has $n + r(n^4 - 1)$ vertices and $m + rn^4$ edges.
Now we consider two different spanning trees in H :

1. We take an arbitrary spanning tree T in G and add 1-off chains to all pairs of non-adjacent vertices. This spanning tree of H is denoted as T_1. The fundamental cycles in H with respect to T_1 are defined by the edges of the co-tree $E(G \setminus T)$ and the missing edge of each 1-off chain. Let l_{uv}^T denote the distance (number of edges in the path) between u and v in T, the total length of the fundamental cycle set in H w.r.t. T_1 is
$$L(T_1) = \sum_{(u,v)\in E(G\setminus T)} (l_{uv}^{T_1} + 1) + \sum_{(u,v)\notin E} (n^4 + l_{uv}^{T_1}).$$
Since $E(G\setminus T)$ has cardinality $m-n+1$ and the number of non-adjacent vertex pairs of G is r, the above expression can be rewritten as
$$L(T_1) = \sum_{(u,v)\in E(G\setminus T)} l_{uv}^{T_1} + (m-n+1) + rn^4 + \sum_{(u,v)\notin E} l_{uv}^{T_1}. \qquad (2.2)$$
The total path length P in T (from which T_1 is derived) between all pairs of vertices of G is
$$P = 2[\sum_{(u,v)\notin E} l_{uv}^{T_1} + \sum_{(u,v)\in E(G\setminus T)} l_{uv}^{T_1} + \sum_{(u,v)\in T} l_{uv}^{T_1}]. \qquad (2.3)$$
We note that $\sum_{(u,v)\in T} l_{uv}^{T_1} = n-1$, combine Equations (2.2) and (2.3) and get
$$L(T_1) = \frac{P}{2} + rn^4 + (m-n+1) - (n-1).$$

2. Let T_2 be a spanning tree in H that solves (MinFCycB). Let q be the number of complete chains in T_2.

Claim: $q = 0$
$G \setminus T$ now contains $m - n + 1 + q$ edges, T ($= T_2 \cap E$, does not span G in general) has $n - 1 - q$ branches, the number of non-adjacent vertex pairs in G still is r, the number of non-adjacent vertex pairs in G which are not joined by a complete chain (denoted by E') is $r - q$. Thus, we get the length of the fundamental cycle set w.r.t. T_2

$$L(T_2) = \sum_{(u,v)\in E(G\setminus T)} l_{uv}^{T_2} + (m-n+1+q) + (r-q)n^4 + \sum_{(u,v)\in E'} l_{uv}^{T_2}. \quad (2.4)$$

Let S be the total path length in T_2 between all pairs of vertices of G, i.e.

$$S = 2[\sum_{(u,v)\notin E} l_{uv}^{T_2} + \sum_{(u,v)\in E(G\setminus T)} l_{uv}^{T_2} + \sum_{(u,v)\in T} l_{uv}^{T_2}]. \quad (2.5)$$

Combining Equations (2.4) and (2.5), we obtain

$$L(T_2) = \frac{S}{2} + (r-q)n^4 + (m-n+1+q) - (n-1-q) - qn^4.$$

Note that E' is a subset of the edges corresponding to non-adjacent vertex pairs in G and $\sum_{(u,v)\in E'} l_{uv}^{T_2} = \sum_{(u,v)\notin E} l_{uv}^{T_2} - qn^4$.

By definition $L(T_2) \leq L(T_1)$

$$\begin{aligned} \Rightarrow \quad & \tfrac{S}{2} + (r-q)n^4 + (m-n+1+q) - (n-1-q) - qn^4 \\ \leq \quad & \tfrac{P}{2} + rn^4 + (m-n+1) - (n-1) \\ \Rightarrow S \quad \leq \quad & P + 4qn^4 - 4q \end{aligned}$$

Claim: The above inequality forces q to be zero.
Note that n^3 is an upper bound for P, as P is calculated over $n(n-1)$ pairs of vertices with respective path lengths at most n.
We calculate a lower bound for S, assume w.l.o.g. that the spanning tree T_2 has its q complete chains connected in star fashion. Hence we get:
$q = 1 \Rightarrow S \geq 2n^4 + 2(n-2)(n^4+1)$
$q = 2 \Rightarrow S \geq 8n^4 + 2(n-3)(n^4+1)$
$q > 2 \Rightarrow S \geq 2q^2n^4$.
With these bounds q cannot be greater than zero.
Consequently, T_2 cannot have any complete chains and furthermore $S \leq P$. That means the solution of the (MinFCycB) in H (or rather $T_2 \cap E$) solves the (STPLS) in G as well. Supergraph H can be constructed in polynomial time. Thus the (STPLS) problem is polynomially transformable to the (MinFCycB). □

Theorem 2.89. *Given graph G, it is an NP-complete problem to find a spanning tree T_{min} in G that solves (MinFCycB).*

Proof. Assume that we can find a solution of (MinFCycB) in polynomial time, then by Lemma 2.88 we could also get a solution of (STPLS) in polynomial time. This contradicts the fact that (STPLS) is NP-complete. □

Lemma 2.90. *If $w_e = 1$ for all $e \in E$, then (MinFCutB) is equivalent to (MinFCycB).*

Proof. An element $e \notin T$ which determines the fundamental cycle C_e belongs to all those fundamental cuts which are determined by the elements $e \in T$ contained in C_e (see Theorem 2.63).
$\Rightarrow$ Objective function:

$$Z_{MinFCutB}(T) = \sum_{e \in T} w_e + \sum_{e \notin T} w_e(|C_e| - 1)$$

If $w_e = 1$ for all $e \in E$:

$$\begin{aligned} Z_{MinFCutB}(T) &= n - 1 + \sum_{e \notin T} |C_e| - (m - n + 1) \\ &= 2n - 2 - m + Z_{MinFCycB}(T) \end{aligned}$$

$\Rightarrow$ optimal tree T for (MinFCycB) is also optimal for (MinFCutB). □

Theorem 2.91. *(MinFCutB) is NP-complete.*

Proof. Let P_1 be the decision version of (MinFCycB), P_2 of (MinFCutB): Is there a spanning tree T such that $Z_{MinFCycB}(T)$ resp. $Z_{MinFCutB}(T)$ is less than or equal to a given k?
We know by Theorem 2.89 that P_1 is NP-complete in the unweighted case and from Lemma 2.90 that $P_1 \propto P_2$ in the unweighted case. Hence, P_2 is NP-complete for the unweighted case, thus also for the more general weighted case. □

These results give rise for the search for heuristic procedures obtaining a near-optimal solution in reasonable running time.

2.5 "Sandwiching" the optimal solution

As the (MinFCutB) is NP-hard, optimal solutions for large instances cannot be obtained unless $\mathcal{P} = \mathcal{NP}$. Later in this thesis I will try to get feasible solutions (so called *primal* or *upper* bounds) and infeasible solutions (*dual* or *lower* bounds). Any pair of an upper and a lower bound defines a frame in which the optimal solution lies. This gap should be as small as possible to narrow down the optimal solution. The pairs can be used in a branch and bound- approach.

The following part is taken from www.wikipedia.org:
Branch and Bound is a general method to find the optimal solution of an NP-hard optimization problem. The method comes from A. H. Land and A. G. Doing [LD60].

The objective is to find a minimal value in a feasible region. Therefore, two tools are used:

- Branching: covering of the feasible region by several smaller feasible subregions, these subregions are the vertices of a tree structure, called branch-and-bound-tree
- Bounding: fast way of finding upper and lower bounds for the optimal solution within a feasible subregion

Now, in case of minimization, if the lower bound of one subregion is greater than the upper bound of another subregion, then the first one can be discarded. This step is called pruning. If upper and lower bound of a subregion match, the minimum within the corresponding subregion is found. In practice, the algorithm terminates after a given time. It might not give the optimal solution, but an interval as small as possible which contains the global optimum.

The algorithm works as effective as its components, the components are often specially designed for an applications.

2.6 Applications

When it comes to applications of (MinFCutB), nothing has been done so far.
One approach might be to look at a cut as a malfunction. Let us regard the

vertices as units trying to reach each other. The edge weight gives the absolute frequency of how often they need to connect (during a day, month...). There should be a path between each pair of vertices, but due to high construction costs cycles cannot be afforded. That means, we are looking for an acyclic, connected subgraph, a tree.
Telecommunications is one field in which networks are applied, however, there are often safety policies requiring more than a tree, e.g. biconnected networks. Another field is installation engineering, where tree-like structures are quite common due to the construction costs.
Every network is subjected to malfunctions. We assume that a malfunction is going to happen and this will affect *one* edge of the tree. The probability for an edge to be affected is uniformly distributed. This malfunction is a fundamental cut and the cut weight gives us the number of failing connections. Now the expected failure for a specific fundamental cut is the product of the probability $\frac{1}{n-1}$ and the cut weight. The expected failure for the whole tree is the sum of the single expected failures, i.e.

$$\sum_{e \in T} \frac{1}{n-1} w_{t_e} = \frac{1}{n-1} \sum_{e \in T} w_{t_e}.$$

The reader should recognize the second factor on the right hand side as the total cut weight. Thus, if we minimize the total cut weight, we minimize the expected failure as well (provided that we have a uniform distribution).

Example 2.92. Let us assume, we have four persons, the number of calls per week between each pair is known and shown in Figure 2.13. It is not cost-efficient to have a cycle in the telephone wires. So, a tree is needed. If a malfunction of one edge occurs, we have to add up all the edges that belong to its fundamental cut. This number of calls cannot be done. On the left-hand side, if $e = (v_1, v_3)$ is disturbed, then (v_2, v_4) and (v_3, v_4) do not work, too, i.e. we have a loss of ten calls. The probability of a branch to be affected is $\frac{1}{3}$. Hence, we have an expected failure of

$$\frac{1}{3} \times [w_{t_{13}} + w_{t_{14}} + w_{t_{24}}] = \frac{1}{3} \times [10 + 9 + 9] = 9\frac{1}{3}.$$

If we would choose the right tree, the expected failure is:

$$\frac{1}{3} \times [w_{t_{14}} + w_{t_{24}} + w_{t_{34}}] = \frac{1}{3} \times [9 + 9 + 15] = 11.$$

Under the assumption of a uniformly distributed failure, the left tree is more advantageous. The result would change if e.g. edge (v_3, v_4) is almost never subject to failures.

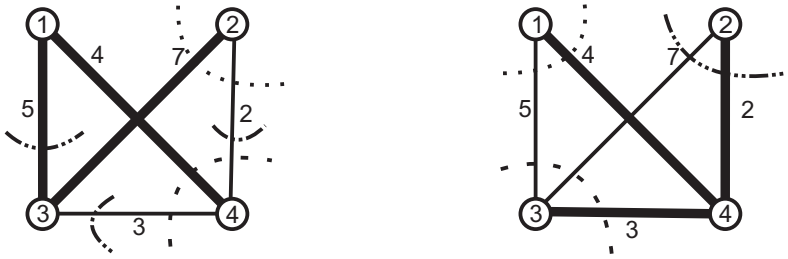


Figure 2.13: Telephone network and optimal solution

Chapter 3

Formulation of the problem

In order to get the optimal solution of (MinFCutB) and to estimate the quality of the heuristics, we need implementable functions. First, the objective function is modified, later on we have a closer look at the constraints.

3.1 New objective function

The core of the following lemma is similar to that of Theorem 2.63:

Lemma 3.1. *Let G be given, let T be any tree in G. Let $f = (u, v) \in E$ (might be chord as well as branch!) and P_{uv}^T the path in T joining u and v. Its length, l_{uv}^T, is equal to the number of edges on the path. Then the the number of fundamental cuts defined by T in which f is contained is equal to the length of the path in the tree which joins the endpoints of f, i.e.*

$$|\{e \in T : \ f \in t_e\}| = |\{e \in T : \ e \in P_{uv}^T\}| = l_{uv}^T. \tag{3.1}$$

Remark 3.2. Not only the number of elements of the two sets in Equation (3.1) is equal, even the elements itself are the same.

Remark 3.3. The length l_{uv}^T "counts" the edges on a path. It does not take their weights into account. However, in the unweighted case (all edge weights equal to 1) the weight of a cut is given by the number of edges.

Proof. To show:

$$e \in P_{uv}^T \quad \Leftrightarrow \quad f \in t_e$$

$\Rightarrow$ Let $e = (v_i, v_{i+1}) \in P_{uv}^T$. W.l.o.g. $l_{uv_i}^T < l_{uv_{i+1}}^T$. Then a cut at e partitions the tree and the corresponding vertices into $X = \{v = v_0, v_2, \ldots, v_i\} \cup Y$ and $\overline{X} = \{v_{i+1}, \ldots, v_{l_{uv}^T} = v\} \cup \overline{Y}$, where the vertices of Y and $\overline{Y}$ do not belong to P_{uv}^T and are not of relevance in this context. The endpoints of the edge $f = (u, v)$ hence belong to different shores, i.e. $f \in t_e$.

$\Leftarrow$ Let $e = (v_i, v_{i+1}) \notin P_{uv}^T$. Cutting at e does not separate the path, i.e. P_{uv}^T remains completely in one shore. Hence f (and its endpoints respectively) is in one shore and thus not included in t_e.

□

Example 3.4. Lemma 3.1 is visualized in Figure 3.1, the path connecting v_1 and v_2 has length 3. Edge (v_1, v_2) is contained in three fundamental cuts.

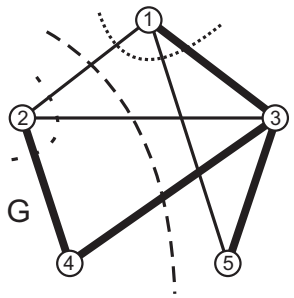


Figure 3.1: Visualization of Lemma 3.1

Example 3.5. If we calculate the total cut weight (all edge weights equal to 1) in Figure 3.2 according to the new objective function, we get the sum $l_{(1,5)}^T + l_{(3,5)}^T + l_{(4,5)}^T + (n-1) = 4+3+2+4 = 13$, where the last summand is the size of the tree itself or $l_{(1,3)}^T + l_{(3,4)}^T + l_{(4,2)}^T + l_{(2,5)}^T$ more precisely. If we use the original objective function, the sum is $w_{t_{(1,3)}} + w_{t_{(3,4)}} + w_{t_{(4,2)}} + w_{t_{(2,5)}} = 2+3+4+4 = 13$.

Theorem 3.6. *Let G be given, let T be any tree in G. Then*

$$\sum_{e \in T} w_{t_e} = \sum_{f=(u,v) \in E} l_{uv}^T \cdot w_f. \tag{3.2}$$

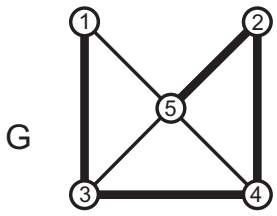


Figure 3.2: Calculating the total cut weight

Proof.

$$\sum_{e\in T} w_{t_e} = \sum_{e\in T} \sum_{f\in E: f\in t_e} w_f = \sum_{f\in E} \sum_{e\in T: f\in t_e} w_f = \sum_{f=(u,v)\in E} l_{uv}^T \cdot w_f,$$

where the first equality follows by the definition of cut weight and the last one by Lemma 3.1. □

Using this we obtain the following modified objective function:

$$\min_{\text{T: T tree in G}} \sum_{e=(u,v)\in E} l_{uv}^T \times w_e \tag{3.3}$$

Remark 3.7. The sum considers all edges, i.e. the branches have $l^T = 1$ and occur only once in the total cut weight. It does not sum over all pairs of vertices! That means, the length of a path in the tree is not relevant if its endpoints are not joined by a chord.

3.2 Constraints

Basically, we get one of the following linear programs, both of whom involve certain difficulties:

$$\begin{array}{lll} \min & \sum\limits_{e=(u,v)\in E} l_{uv}^T \cdot w_e & \\ \text{s.t.} & T & : \text{ tree} \\ & l_{uv}^T & : \text{ length of the path } \quad \forall\, e=(u,v)\in E \end{array}$$

or

$$\begin{array}{lll} \min & \sum_{e\in T} w_{t_e} & \\ \text{s.t.} & T & : \text{ tree} \\ & t_e & : \text{ fundamental cut defined by branch } e \quad \forall\, e\in T \end{array}$$

If not mentioned otherwise, the binary variables of the upcoming formulations take on the following values:

$$\begin{aligned} x_e^k = x_{uv}^k &= 1 \quad \Leftrightarrow \quad \text{edge } e=(u,v) \text{ is in cut } k \\ t_e^k &= 1 \quad \Leftrightarrow \quad \text{edge } e \text{ is branch in cut } k \\ y_e &= 1 \quad \Leftrightarrow \quad \text{edge } e \text{ is contained in the tree} \\ y_{uv} &= 1 \quad \Leftrightarrow \quad \text{the edge } \textit{directed} \text{ from } u \text{ to } v \text{ is contained in the tree} \end{aligned}$$

Essentially, all the formulations produce a similar solution vector giving the incidence vectors of the cuts and the edges of the tree, e.g.

$$(x_1^1, \ldots, x_m^1 \,|\, x_1^2, \ldots, x_m^2 \,|\, \ldots \,|\, x_1^{n-1}, \ldots, x_m^{n-1} \,|\, y_1, \ldots, y_m) \in \mathbb{B}^{n \cdot m}.$$

Tree-defining constraints

A tree can be defined by the following constraints:

$$\begin{aligned} \sum_{e \in E} y_e &= n-1 \\ \sum_{e \in E(S)} y_e &\leq |S| - 1 \quad \forall\, S \subset V,\, |S| \geq 2 \end{aligned}$$

Unfortunately, the second line produces exponentially many constraints.

Another set of $O(n^2)$ tree-defining constraints is:

$$\sum_{(u,v) \in E} y_{uv} = n-1 \tag{3.4}$$

$$level(v) \geq level(u) + 1 - n + n \cdot y_{uv} \quad \forall\, u, v \in V,\, u \neq v \tag{3.5}$$

$$\sum_{v \in V, v \neq u} y_{uv} = 1 \quad \forall\, u \in V,\, u \neq 1 \tag{3.6}$$

$$level(u) \in \mathbb{N} \quad \forall\, u \in V \tag{3.7}$$

The tree has $n-1$ edges, is acyclic and all its edges are directed towards the root (vertex 1). Note that $y_e = y_{uv} + y_{vu} \quad \forall\, e = (u,v) \in E$. This part is taken from an XPress Example.

"Cut-specific" constraints

Besides the tree-defining constraints, there are some more constraints showing up in several formulations.

Every edge is contained in at least one cut:

$$\sum_{k \leq r} x_e^k \geq 1 \qquad \forall\, e \in E$$

Every cut contains at least one edge, in biconnected graphs at least two:

$$\sum_{e \in E} x_e^k \geq 1 \qquad \forall\, k \leq r$$

The "Syslo-Condition" referring to Theorem 2.48 (see [Bun06]):

$$\sum_{k \leq r} x_e^k \leq 1 + (n-1)(1 - y_e) \qquad \forall\, e \in E$$

If an edge is part of the tree, it cannot be contained in more than one fundamental cut. On the other hand, it has to be contained in at least one cut, hence, it is part of exactly one cut.

Formulation with shores

The following formulation is presented in [Bun06] with slight modifications:

$$\min \quad \sum_{k \leq r} \sum_{(u,v) \in E} w_{uv} x_{uv}^k \tag{3.8}$$

$$\text{s.t.} \quad \sum_{(u,v) \in E} y_{uv} = n - 1 \tag{3.9}$$

$$level(v) \geq level(u) + 1 - n + n \cdot y_{uv} \quad \forall\, u, v \in V,\, u \neq v \tag{3.10}$$

$$\sum_{v \in V,\, v \neq u} y_{uv} = 1 \quad \forall\, u \in V,\, u \neq 1 \tag{3.11}$$

$$\sum_{k \leq r} x_{uv}^k \geq 1 \quad \forall\, (u,v) \in E \tag{3.12}$$

$$\sum_{(u,v) \in E} t_{uv}^k = 1 \quad \forall\, k \leq r \tag{3.13}$$

$$t_{uv}^k \leq x_{uv}^k \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.14}$$

$$\sum_{k \leq r} t_{uv}^k \leq y_{uv} + y_{vu} \quad \forall\, (u,v) \in E \tag{3.15}$$

$$\sum_{h \leq r} x_{uv}^h + r t_{uv}^k - 2 x_{uv}^k \leq r - 1 \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.16}$$

$$\sum_{u \in V} z_u^k \geq 1 \quad \forall\, k \leq r \tag{3.17}$$

$$x_{uv}^k \leq z_u^k + z_v^k \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.18}$$

$$x_{uv}^k \leq 2 - z_u^k - z_v^k \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.19}$$

$$-x_{uv}^k \leq z_u^k - z_v^k \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.20}$$

$$-x_{uv}^k \leq z_v^k - z_u^k \quad \forall\, (u,v) \in E,\, k \leq r \tag{3.21}$$

$$x_{uv}^k,\, y_{uv},\, t_{uv}^k,\, z_u^k \in \{0,1\} \quad \forall\, (u,v) \in E,\, u \in V,\, \forall\, k \leq r \tag{3.22}$$

$$level(u) \in \mathbb{N} \quad \forall\, u \in V \tag{3.23}$$

The z-variable takes on the following values:

$$z_u^k = \begin{cases} 1 & \text{if vertex } u \text{ is lying on shore } U_k \text{ of the cut } (U_k, V \setminus U_k) \\ 0 & \text{else.} \end{cases}$$

Each cut has exactly one branch (3.13), a branch of a cut has to be contained in this cut (3.14) and in the tree (3.15), (3.16) is the "Syslo-condition". Constraints (3.18) – (3.21) guarantee that an edge is element of a cut if and only if its endvertices are lying in different shores of the cut.

The formulation has $O(mn)$ variables and $O(mn)$ constraints. It yields feasible results even if either the y- or the t-constraints are deleted. In the latter case $\sum_{(u,v)\in E} x_{uv}^k \quad \forall\, k \leq r$ and $\sum_{k\leq r} x_{uv}^k \leq 1 + r(1 - y_{uv} - y_{vu}) \quad \forall\, (u,v) \in E$ have to be added. The decrease in the number of constraints and variables leads to higher computation times.

Formulation with cycles

According to Corollary 2.66 a cut can be modelled by the requirement that the intersection of this cut with any element of an arbitrary cycle basis has even cardinality. In order to get a cycle basis for a given graph the algorithm of Paton can be used (see [Pat69] and Section 2.2.2). The cycles can be used as an input for the following formulation (given in [Bun06]):

$$\min \quad \sum_{k\leq r} \sum_{e\in E} w_e x_e^k \tag{3.24}$$

$$\text{s.t.} \quad \sum_{k\leq r} x_e^k \geq 1 \qquad \forall\, e \in E \tag{3.25}$$

$$\sum_{e\in E} x_e^k \geq 2 \qquad \forall\, k \leq r \tag{3.26}$$

$$\sum_{e\in F} x_e^k - \sum_{e\in D\setminus F} x_e^k \leq |F| - 1 \qquad \forall \text{ cycles } D \text{ of cycle basis,}$$

$$\forall\, F \subset D,\ |F| \text{ odd},\ \forall\, k \tag{3.27}$$

$$\sum_{(u,v)\in E} y_{uv} = n - 1 \tag{3.28}$$

$$level(v) \geq level(u) + 1 - n + n \cdot y_{uv} \qquad \forall\, u, v \in V,\ u \neq v \tag{3.29}$$

$$\sum_{v\in V, v\neq u} y_{uv} = 1 \qquad \forall\, u \in V,\ u \neq 1 \tag{3.30}$$

$$y_e = y_{uv} + y_{vu} \qquad \forall\, e = (u,v) \in E \tag{3.31}$$

$$\textstyle\sum_{k\leq r} x_e^k \leq 1 + r(1 - y_e) \qquad \forall\, e \in E \tag{3.32}$$

$$x_e^k,\ y_e,\ y_{uv} \in \{0,1\} \qquad \forall\, e, (u,v) \in E,\ k \leq r \tag{3.33}$$

$$level(u) \in \mathbb{N} \qquad \forall\, u \in V \tag{3.34}$$

Constraint (3.27) can be approximated by the following set of constraints:

$$\sum_{e\in F} x_e^k - \sum_{e\in D\setminus F} x_e^k \leq 0 (= |F|-1) \quad \forall \text{ cycles } D \text{ of cycle basis}, \\ \forall\, F \subset D, |F| = 1, \forall\, k \leq r \tag{3.35}$$

$$\sum_{e\in D} x_e^k \leq 2 \qquad \forall \text{ cycles } D \text{ of cycle basis} \tag{3.36}$$

Now, let the cycle basis be defined by:

$$c_e^l = \begin{cases} 1 & \text{if edge } e \text{ is contained in cycle } l \\ 0 & \text{else,} \end{cases}$$

We can rewrite the above conditions as follows:

$$c_f^l x_f^k - \sum_{e\in E,\, e\neq f} c_e^l x_e^k \leq 0 \quad \forall\, f \in E,\, l \leq \nu,\, k \leq r \tag{3.37}$$

$$\sum_{e\in E}{}^{m} c_e^l x_e^k \leq 2 \qquad \forall\, l \leq \nu,\, k \leq r \tag{3.38}$$

The above constraints ensure that a cycle and a cut have none or two edges in common. This holds if the fundamental cuts and cycles are defined by the same spanning tree, as their intersection cannot contain other edges than the cycle-defining chord and the cut-defining branch. It does not hold for any choice of a cycle and a cut, not even for fundamental cuts and cycles, see Figure 3.3. It depicts two different trees, a fundamental cycle defined by the dashed chord on the left side and a fundamental cut on the right side. They have four edges in common.

Hence, if we use a fundamental cycle basis as an input, the constraints may cut off feasible fundamental cuts. This formulation yields at least an upper bound for the (MinFCutB).

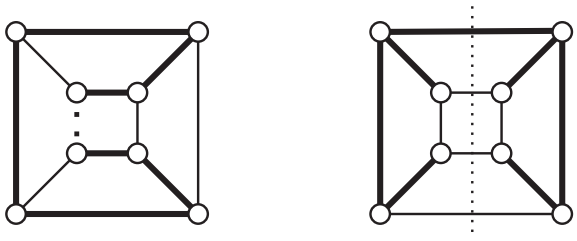


Figure 3.3: Fundamental cycle and cut

Modification of a (MinFCycB) formulation

It is possible to use formulations for (MinFCycB) (e.g. given in [LAMM03]) with slight modifications of the objective function and new constraints modelling the path length. The modified formulation solves the (MinFCutB), the instance has to be biconnected.

$$\begin{aligned}
\min \quad & \sum_{e\in E} w_e \cdot l_e && && (3.39)\\
\text{s.t.} \quad & l_e \geq 1 && \forall\, e \in E && (3.40)\\
& l_e \geq \sum_{f\in E} z_{ef}^k - t_e^k && \forall\, e \in E,\, k \leq \nu && (3.41)\\
& z_{ef}^k \leq t_e^k && \forall\, e, f \in E,\, k \leq \nu && (3.42)\\
& z_{ef}^k \leq x_f^k && \forall\, e, f \in E,\, k \leq \nu && (3.43)\\
& z_{ef}^k \geq t_e^k + x_f^k - 1 && \forall\, e, f \in E,\, k \leq \nu && (3.44)\\
& \sum_{k\leq\nu} x_e^k \geq 1 && \forall\, e \in E && (3.45)\\
& \sum_{e\in E} x_e^k \geq 3 && \forall\, k \leq \nu && (3.46)\\
& \sum_{e\in E} t_e^k = 1 && \forall\, k \leq \nu && (3.47)\\
& \sum_{e\in E,\, k\leq\nu} t_e^k = \nu && && (3.48)\\
& \sum_{h\leq\nu} x_e^h + \nu t_e^k - 2x_e^k \leq \nu - 1 && \forall\, e \in E,\, k \leq \nu && (3.49)\\
& t_e^k \leq x_e^k && \forall\, e \in E,\, k \leq \nu && (3.50)\\
& \sum_{e\in\delta(j)} x_e^k \leq 2 && \forall\, j \in V,\, k \leq \nu && (3.51)\\
& \sum_{e\in\delta(j):e\neq h} x_e^k \geq x_h^k && \forall\, h \in \delta(j),\, j \in V,\, k \leq \nu && (3.52)\\
& x_e^k,\, t_e^k,\, z_{ef}^k \in \{0,1\} && \forall\, e, f \in E,\, k \leq \nu && (3.53)\\
& l_e \in \mathbb{N} && \forall\, e \in E && (3.54)
\end{aligned}$$

The output of this formulation are incidence vectors of the fundamental cycles and their respective chords, namely:

$$x_e^k = \begin{cases} 1 & \text{if edge } e \text{ is contained in cycle } k \\ 0 & \text{else,} \end{cases}$$

$$t_e^k = \begin{cases} 1 & \text{if edge } e \text{ is chord in cycle } k \\ 0 & \text{else.} \end{cases}$$

The z-variable is used to linearize the constraints:

$$z_{ef}^k = t_e^k x_f^k \begin{cases} 1 & \text{if edge } e \text{ is chord and edge } f \text{ is edge in cycle } k \\ 0 & \text{else.} \end{cases}$$

There is no explicit information about the cuts. The tree is defined by those edges e which are not chord of any cycle, i.e. $\sum_{k \leq \nu} t_e^k = 0$.

Constraints (3.45) to (3.52) implicitly constitute a spanning tree and define the fundamental cycles and their chords. Every edge is contained in at least one cycle (3.45), every cycle consists of at least three edges (3.46), every cycle has exactly one chord (3.47), every chord is contained in only one cycle (3.49), if an edge is chord in a cycle it has to be part of this cycle too (3.50), every vertex has at most two edges of any cycle incident to it (3.51) and if an edge of a cycle is incident to a vertex, there has to be at least one other edge of this cycle being incident to this vertex (3.52). The combination of Constraints (3.51) and (3.52) ensures that any vertex has either zero or two edges of any cycle incident to it.
Constraints (3.40) and (3.44) model the path length for each edge and can be interpreted as follows: If edge e is a branch of the tree, then $t_e^k = 0 \quad \forall\, e, k$, i.e. $z_{ef}^k = 0 \quad \forall\, f \in E$ and thus, $l_e \geq 1$. As we deal with a minimization problem, $l_e = 1$, which is the length of the path in the tree connecting the endpoints of a branch. On the other hand, if $t_e^k = 1$ for some e, k, then $z_{ef}^k = 1 \quad \forall\, f \in E : x_f^k = 1$ and thus, $l_e \geq \sum_{f \in E} x_f^k - 1$. This gives the number of edges in cycle k reduced by one, i.e. the length of the path connecting the endpoints of edge e. Again, the objective function chooses the smallest l_i possible. Due to the linearization in (3.42) - (3.44), the number of variables and constraints, $O(nm\nu)$, is significantly higher than it is in the formulation with shores.

Chapter 4

Relaxations and lower bounds

The heuristics to be presented later on yield feasible, yet suboptimal solutions, i.e. they provide upper bounds of the (MinFCutB). In order to sandwich the optimal solution, it is useful to know lower bounds as well. To this end we relax certain constraints of the (MinFCutB), e.g. the fundamentality or the binary variables.

4.1 LP- Relaxation

Given the IP-Formulation with shores, the binary variables are relaxed to be in the interval $[0, 1]$. The resulting linear program is easier to solve than the integer program. But in general its solution is not feasible. If all binary variables are relaxed, the bounds obtained in numeric tests are rather weak. The reason for this is that each edge does not need to occur in more than one cut. So, if we add up the "shares" of an edge being in the different cuts, we get exactly one, i.e. $\sum_{k \leq r} x_e^k = 1 \quad \forall e \in E$. This does not even come close to (MinFCutB).

The LP-relaxation of the modified (MinFCycB)-formulation does not yield better results.

4.2 Minimum Cut Basis Problem

If the demand for fundamentality of the cuts is neglected, the remaining problem is referred to as *Minimum Cut Basis Problem*. Even though the algorithm of De Pina et al. works quite well concerning the Minimum Cycle

Basis Problem, it becomes rather complex applied to the Minimum Cut Basis Problem. In this case the algorithm of Gomory and Hu has to be preferred. The following section is based on the results of [GH61], [KV00], and [Gus90].

Definition 4.1. Let G be an undirected graph, w a weight function on its edges. For any pair of vertices $u, v \in V$, the minimum weight of a cut separating u and v is denoted by λ_{uv}. A tree T_{GH} is a $Gomory - Hu$ or *cut tree* for (G, w) if $V(T_{GH}) = V(G)$ and $\lambda_{uv} = \min_{e \in P_{uv}} w_{t_e}$ for all $u, v \in V(G)$. P_{uv} is the path in the Gomory-Hu tree, t_e is the fundamental cut in G defined by e with respect to T_{GH}.

Remark 4.2. The weights of the minimal cuts between any pair of vertices is the same in both the Gomory-Hu tree and the original graph. Each edge of this Gomory-Hu tree corresponds to the minimal cut in the graph separating its two endpoints [GH02].

The output of the algorithm of Gomory and Hu is a tree whose $n - 1$ edges define cuts constituting a basis. The core of the procedure is to choose any pair of vertices, find the minimum cut which separates them, and contract the corresponding shores to so called *super-vertices*. This procedure is iterated through the vertices of the contracted graph. The algorithm terminates as soon as $n - 1$ cuts are found, i.e. any pair of vertices is separated by a cut.

Any undirected graph possesses a Gomory-Hu tree, the algorithm works correctly and has complexity $O(n^4)$. It can be proved (see [Bun06]) that the cut tree yields a feasible and optimal solution of the Minimum Cut Basis Problem, consequently, the non-fundamental version of (MinFCutB) is polynomially solvable. Relaxing fundamentality yields lower bounds.

The algorithm of Gomory and Hu is rather difficult to implement. The reason is that sets of vertices have to be contracted to super-vertices in order to avoid crossing cuts. Two cuts $(U, \overline{U})$ and $(W, \overline{W})$ *cross* each other if all of the shores $(U \cap W)$, $(U \cap \overline{W})$, $(\overline{U} \cap W)$, and $(\overline{U} \cap \overline{W})$ are nonempty.

Lemma 4.3. *Any tree defines a set of non-crossing cuts.*

Proof. Let $e, f \in T$, $e \neq f$. The deletion of e separates the graph into two vertex sets U and $\overline{U}$, the deletion of f into W, $\overline{W}$. The endpoints of f lie in one shore of the cut defined by e, wlog $f \in U$. Hence, one shore of the cut defined by f also lies in U, wlog $W \subset U$. It follows that $W \cap \overline{U} = \emptyset$ and by this, the cuts defined by different branches of the tree do not cross. □

Remark 4.4. It is important that the cuts defined by the algorithm of Gomory and Hu are non-crossing. If they crossed, they would not define

a tree by Lemma 4.3. On the other hand, there are $n-1$ cuts and edges, respectively. So the Gomory-Hu output would not be connected. In this case it would not represent the minimum cuts between any pair of vertices in the graph.

In order to get a cut tree, I use the algorithm of D. Gusfield. It does not contract vertices, yet obtains non-crossing cuts. This approach includes the following algorithm of Edmonds and Karp (see e.g. [KV00]):

Algorithm 4.5. *Algorithm of Edmonds and Karp*
Input: undirected graph $G = (V, E)$, weight function $w : E \to \mathbb{R}_+$, vertices s and t
Output: maximum flow and minimum cut between s and t

1. *Initialize $f(e) := 0$ for all edges e*
2. *Find a shortest flow-augmenting path P in the residual graph G_f, if none exists STOP*
3. *Set $\gamma := \min_{e \in P} w_f(e)$. Increase the flow f along P by γ, decrease w_f along P by γ. Go to 2.*

This algorithm is intended for capacity functions which are decreased as the flow is augmented. In our case we deal with the weight function and use it analogously. If $w_f(e)$ becomes zero for an edge e, e cannot be part of a flow-augmenting path anymore. The algorithm terminates when t cannot be reached by s anymore. The vertices which can be reached by s and s itself now constitute the s-side of the cut. The flow value is equal to the sum of all γ.

To put it short, the algorithm of Gusfield starts with a star tree rooted at vertex 1 which is predecessor to all vertices. It chooses $n-1$ vertices in order from 2 to n. In each iteration the algorithm computes the minimum cut between this vertex s and its predecessor t. Vertices having predecessor t and lying on the s-side of the cut do now have s as predecessor. If the predecessor of t also lies on the s-side, s is now linked to the predecessor of t and t itself linked to s. This procedure makes sure that the final tree defines a set of non-crossing cuts.

Algorithm 4.6. *Cut Tree (Algorithm of Gusfield)*
Input: undirected graph $G = (V, E)$, weight function $w : E \to \mathbb{R}_+$
Output: cut tree T
Initialize $p[v] := 1$ for all vertices $v \in V$

for all $s := 2 \dots n$
    Compute a minimum cut between vertices s *and* $t := p[s]$ *in* G
    X *denotes the* s*-side of the cut,* $f(s,t)$ *the maximum flow (Edmonds-Karp)*
    $fl[s] := f(s,t)$
    for all $i := 1 \dots n$
        if *(*$i \neq s$ *and* $i \in X$ *and* $p[i] = t$*)* **then** $p[i] := s$
    if *(*$p[t] \in X$*)* **then** $p[s] := p[t]$, $p[t] := s$, $fl[s] := fl[t]$, *and* $fl[t] := f(s,t)$
$(s, p[s])$ *are branches of the tree for all vertices* $s = 2, \dots, n$

The correctness of this procedure is proved in [Gus90]. As the algorithm of Edmonds and Karp has complexity $O(nm^2)$, Gusfield's algorithm has $O(n^2m^2)$.

Chapter 5

Heuristics and upper bounds

5.1 "Basics"

5.1.1 Pairwise shortest paths

In order to calculate path lengths and objective function values, we can use the algorithm of Floyd and Warshall (see [Flo62]). Its output is a *distance matrix D* giving the distance between any pair of vertices and a so called *predecessor matrix Pred* to backtrack the corresponding paths.

Algorithm 5.1. *Algorithm of Floyd and Warshall*
Input: adjacency matrix A(G)
Output: distance matrix D, predecessor matrix Pred
for all $i := 1, \ldots, n$
  for all $j := 1, \ldots, n$
    if $a_{ij} = 1$ **then** $d_{ij} := 1$, $pred_{ij} := i$
    else $d_{ij} := \infty$, $pred_{ij} := \infty$
for all $k := 1, \ldots, n$
  for all $i := 1, \ldots, n$
    if $d_{ik} < \infty$ **then**
      for all $j := 1, \ldots, n$
        if $d_{kj} < \infty$ **then**
          if $d_{ij} > d_{ik} + d_{kj}$ **then** $d_{ij} := d_{ik} + d_{kj}$, $pred_{ij} := pred_{kj}$

The algorithm has complexity $O(n^3)$.

5.1.2 Incidence vectors

The algorithms of Section 5.2 have a tree T as output. So far, there is no explicit information about the fundamental cuts. One possibility is to determine for each branch e the shores in $T \setminus e$ and the edges having their endpoints in different shores (see [Gib85]).

I used the following approach in the implementation, it is based on Theorem 2.63. The output of the algorithm is the $(n-1) \times m$ fundamental cut matrix $\mathfrak{Q}_f$.

Algorithm 5.2. *Transforming trees to cut matrices*
Input: graph G, tree T
Output: $\mathfrak{Q}_f$
Calculate Pred(T) by the algorithm of Floyd and Warshall
for all $i := 1, \ldots, n-1$, $j = 1, \ldots, m$ *initialize* $q_{ij} := 0$
Enumerate the edges such that the branches have numbers from 1 to n-1 and the chords from n to m
for all $i := 1, \ldots, n-1$ *set* $q_{ii} := 1$
for all $j := n, \ldots, m$
  backtrack the path P in T connecting the endpoints of edge j by Pred(T)
  for all *edges* $i \in T$
    if $i \in P$ **then** *set* $q_{ij} := 1$

The complexity of this procedure is determined by the algorithm of Floyd and Warshall, i.e. $O(n^3)$.

5.1.3 Calculating the total cut weight

In order to get the objective function value of the (MinFCutB), the cut-vectors are multiplied by the weight vector and the respective products are summed up. This can be done in $O(n^2)$ time.

If the incidence vectors are not of importance, the total cut weight of a tree can be calculated in $O(n^3)$ time. We calculate the pairwise shortest paths in the tree by means of the algorithm of Floyd and Warshall and use them in equation (3.3).

5.1.4 Unweighted graphs

Given an unweighted graph with n vertices and m edges, each of the $n-1$ fundamental cuts contains at most $m-n+1$ chords, hence we have an upper

bound to the total cut weight of $(n-1)(m-n)$. This upper bound cannot be exceeded by any spanning tree.

The following is based on the calculations for fundamental cycles in [DPK82]: Let K_n be an unweighted and complete graph with n vertices, we calculate the total cut weight with respect to a

- Hamiltonian tree T_h:

$$TCW(T_h) = \sum_{i=1}^{n-1} i \cdot (n-i),$$

  as there are two cuts of $n-1$ elements, two cuts of $2(n-2)$ elements and so on,

- star tree T_s:

$$TCW(T_s) = (n-1) \cdot (n-1),$$

  as each of the $n-1$ fundamental cuts contains $n-1$ edges.

Noting that $TCW(T_h) \geq TCW(T_s)$ for all n, we get $(n-1)^2$ as an upper bound for the total cut weight of unweighted complete graphs. Does this bound also hold for arbitrary unweighted graphs?

5.2 Initial solutions

The modified objective function (3.3) consists of weight parameters and lengths of paths in the tree. It seems reasonable to look at them in detail considering the minimization problem. This will be done in Subsections 5.2.1 and 5.2.2, where solutions are build from scratch, i.e. we have *primal-dual* algorithms yielding a feasible solution after their last iteration. The output of each of these tree-growing algorithms is a tree fulfilling certain "substitute requirements". These requirements can be solved quite fast and more easily than the (MinFCutB), yet they should match its objective function as much as possible. Unfortunately, their solution can be far from optimal. Another approach based on the minimum cut tree is presented in Subsection 5.2.4.

5.2.1 Heavy Tree

This section focuses on the weight parameters. We have seen in (3.2) that any edge $e = (u, v)$ (strictly speaking its weight) is added as often in the total cut weight as $P^T_{(u,v)}$ is long.

Remark 5.3. When it comes to the frequency of how often an edge occurs in the objective function, two observations can be made:

- The weight of every edge has to occur at least once in the sum, as the edge joins two vertices who are again separated by at least one cut.
- The weight of a branch appears only once, as the length of the path in the tree joining its endpoints is one.

It is desirable that "heavy" edges appear in the total sum as less as possible, i.e. they should be part of the tree. So we are looking for a tree with maximum weight or a *maximum spanning tree*. This can be done using the algorithm of Kruskal such that the tree weight is maximized. Basically, the procedure grows a maximum spanning tree out of a forest. In each iteration the heaviest edge is added to the spanning forest if no loop is caused by this addition. The idea for this construction is given in [Kru56], implementation details in [AHU74].

Algorithm 5.4. *Heavy Tree (Algorithm of Kruskal)*
Input: undirected graph $G = (V, E)$*, weight function* $w : E \to \mathbb{R}$
Output: maximum spanning tree T
Initialize the "isolated node forest" $T := \emptyset$
for all $i := 1, \ldots, n \quad p(i) := i$
Sort the edges in non-increasing order in a list
While $|T| < n - 1$
  find the maximal element $e = (v_1, v_2)$ *in the list*
  if $p(v_1) \neq p(v_2)$ **then**
    if $p(v_1) \geq p(v_2)$ **then**
      for all *vertices* v *for which* $p(v) = p(v_1)$
        set $p(v) := p(v_2)$
      else for all *vertices* v *for which* $p(v) = p(v_2)$
        set $p(v) := p(v_1)$
    $T := T \cup \{e\}$
  delete e *from the list*

Remark 5.5. The algorithm works correctly [ALO93] in $O(m \log n)$ time [CT76]. Dealing with sparse graphs it works much better than the algorithm of Prim ($O(n^2)$), in nearly complete graphs Prim has to be preferred [HK00].

Example 5.6. Kruskal's procedure is demonstrated by means of Figure 5.1. The graph G is depicted on the left, the heavy tree T corresponding to G on the right. The edges $(1, 3)$ and $(2, 3)$ are discarded because they would form

loops if they are selected for the tree. The total cut weight corresponding to T is

$$w_{t_{(1,2)}} + w_{t_{(1,5)}} + w_{t_{(3,5)}} + w_{t_{(4,5)}} = 12 + 22 + 20 + 7 = 61.$$

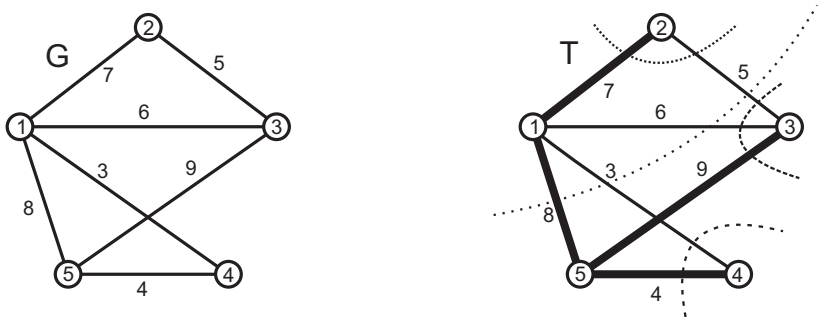


Figure 5.1: Visualization of the Heavy Tree algorithm

Example 5.7. Kruskal does not solve the (MinFCutB) optimal if the edge weights are relatively close to each other as can be seen in Figure 5.2. In this case the next Section 5.2.2 yields better results. The total cut weight of the heavy tree on the left is 57, this result can be improved by an edge swap (as depicted on the right side) to 51. The technique of edge swaps will be explained in Section 5.3.1.

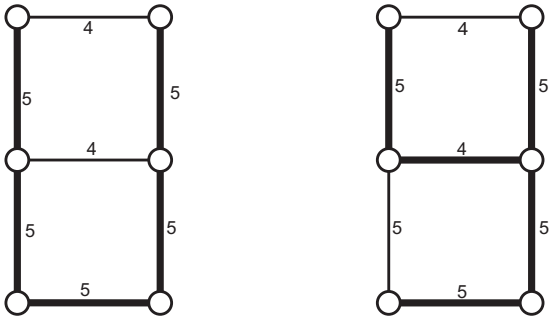


Figure 5.2: Heavy tree and an improvement regarding total cut weight

Remark 5.8. The Heavy Tree approach behaves worst in case of unweighted complete graphs. In this case its output can be a Hamiltonian tree which would be the worst solution as well as an optimal star tree (see Section 5.1.4). In complete graphs with five vertices the Hamiltonian tree is 25% worse than the star tree, it is already 100% worse in graphs with ten vertices.

Remark 5.9. A *dual* greedy algorithm shrinks a given graph to a tree by iteratively deleting edges with minimum weight such that the resulting graph is still connected [Sch03]. This procedure yields the same result.

Remark 5.10. The minimum/ maximum spanning tree is applied in the field of networks where e.g. the amount of wire has to be minimized, cable TV etc. Furter references for applications are listed by Graham and Hell [GH85].

5.2.2 Short Tree

Suppose all weights are equal, then the objective function reduces to

$$\min \sum_{f=(u,v)\in G} l_{uv}^T, \tag{5.1}$$

i.e. in this case we want to minimize the sum of the length of all paths connecting edges of G in T. Hence, we are looking for a tree with short paths. To this end a concept of location theory will be used:

Definition 5.11. Given a graph $G = (V, E)$ and denoting the shortest distance between two vertices u and v by d_{uv}, the *median problem* is defined as follows:

$$\min_{v\in V} \sum_{u\in V} d(v, u). \tag{5.2}$$

A vertex v^* that solves this optimization problem is called *median*.
A similar problem is the *center problem*:

$$\min_{v\in V} \max_{u\in V} d(v, u). \tag{5.3}$$

The solution of this problem is called *center*.

Remark 5.12. The problems of finding minimum spanning trees or shortest path trees are quite different, e.g. regarding the objective functions. More precise, the weight of a branch is taken into account exactly once in the objective function of the first problem and at least once in the latter one [ALO93].

Remark 5.13. Definition 5.11 is a simplification of the classic p-median model in location theory:

$$\min_{X\subset V,\, |X|=p} \sum_{i=1}^{n} w_i \cdot \min_{x\in X} d(x, v_i).$$

In this case w_i is the demand of customers in vertex v_i which is to be satisfied by p facilities located in vertex set X. This demand has to be satisfied at minimum cost where cost is measured by total weighted distance from facility to customer [Sch04]. In our context, the vertices are not assigned any weight and there is only one facility to be located.

Algorithm 5.14. *Short Tree*
Input: undirected graph $G = (V, E)$, weight function $w : E \to \mathbb{R}$
Output: shortest path spanning tree T

1. *Run the algorithm of Floyd and Warshall on the unweighted graph to get D(G) and Pred(G)*
2. *Calculate $dist(v) = \sum_{u \neq v} d_{uv}$ for all $v \in V$*
3. *Find $v^* \in V$ that solves $\min_{v \in V} dist(v)$*
4. *Develop a tree rooted at v^* with shortest paths to all other vertices*

This procedure can be varied using the center in the second step, i.e. calculate $dist(v) = \max_{u \neq v} d_{uv}$. The center minimizes the length of the longest path in the tree.

Remark 5.15. The tree can be developed by means of the predecessor matrix (which is calculated in the algorithm of Floyd and Warshall). We look at the row corresponding to v^*. The entry of a column j is by definition $pred_{v^*j}$ and we add the edge $(pred_{v^*j}, j)$ to the tree for all columns $j \neq v^*$.

Example 5.16. Algorithm 5.14 can be visualized by means of the graph of Example 5.6 shown again on the left of Figure 5.3. The matrices corresponding to G are:

$$D(G) = \begin{pmatrix} \infty & 1 & 1 & 1 & 1 \\ 1 & \infty & 1 & \infty & \infty \\ 1 & 1 & \infty & \infty & 1 \\ 1 & \infty & \infty & \infty & 1 \\ 1 & \infty & 1 & 1 & \infty \end{pmatrix}, \quad P(G) = \begin{pmatrix} - & 1 & 1 & 1 & 1 \\ 2 & - & 2 & - & - \\ 3 & 3 & - & - & 3 \\ 4 & - & - & - & 4 \\ 5 & - & 5 & 5 & - \end{pmatrix}$$

After one iteration of Floyd-Warshall, we get the following output:

$$D(G) = \begin{pmatrix} \infty & 1 & 1 & 1 & 1 \\ 1 & \infty & 1 & 2 & 2 \\ 1 & 1 & \infty & 2 & 1 \\ 1 & 2 & 2 & \infty & 1 \\ 1 & 2 & 1 & 1 & \infty \end{pmatrix}, \quad P(G) = \begin{pmatrix} - & 1 & 1 & 1 & 1 \\ 2 & - & 2 & 1 & 1 \\ 3 & 3 & - & 1 & 3 \\ 4 & 1 & 1 & - & 4 \\ 5 & 1 & 5 & 5 & - \end{pmatrix}$$

This gives $d(1) = 4$, $d(2) = 6$, $d(3) = 5$, $d(4) = 6$ and $d(5) = 5$. Hence, we choose vertex 1 as median.
For the development of the tree we have a look at the first line of the predecessor matrix: All the entries are equal to 1, i.e. we add the edges $(1, j)$ for $j = 2, \ldots, 5$ to the tree which can be seen on the right of Figure 5.3
The total cut weight is:

$$w_{t_{(1,2)}} + w_{t_{(1,3)}} + w_{t_{(1,4)}} + w_{t_{(1,5)}} = 12 + 20 + 7 + 21 = 60.$$

The objective function value has improved compared to the one obtained by means of the Heavy Tree, where the total cut weight was equal to 61.

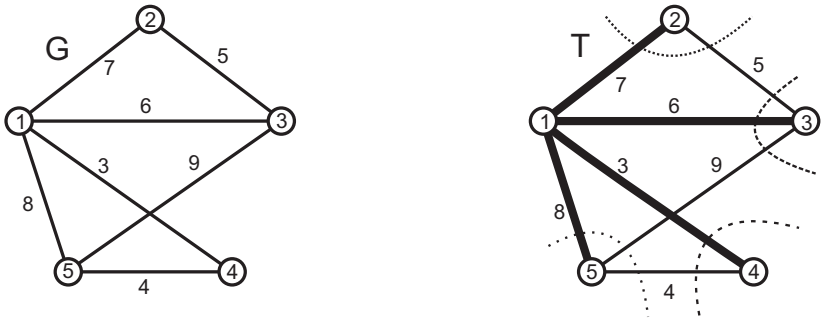


Figure 5.3: Visualization of the Short Tree algorithm

The complexity of the algorithm is determined by the algorithm of Floyd-Warshall, i.e. $O(n^3)$. In general, the solution is not unique.

Example 5.17. The median algorithm can behave bad if the weights are spread, see e.g. Figure 5.4. Let M be a large number. The median tree on the left side yields an objective function value of $6 + 2M$, the tree on the right side has total cut weight of $7 + M$, an absolute difference of $M - 1$.

Remark 5.18. The algorithm neglects the fact that not all path lengths are relevant for the objective function of (MinFCutB), namely the path lengths between vertices which are not adjacent in the graph. If the edge weights are relatively wide spread they are not taken into account adequately by the short tree.

Remark 5.19. There are other ways to find a short tree:

1. choose the most vital links as branches (see Remark 5.20)

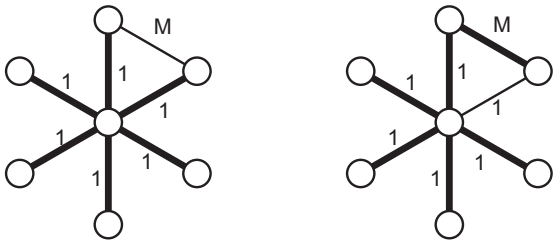


Figure 5.4: Non-optimal Median Tree

2. choose edges into a tree which are contained in many cycles of the underlying graph (see Remark 5.21 and Example 5.22)

3. construct a shortest total path length spanning tree (see Section 2.4), [WCT00]

4. choose branches which are incident to vertices with high degree

   build a tree rooted at the vertex with highest degree and with shortest paths from the root to all the other vertices

   select all edges incident to the vertex with highest degree, then select the edges incident to the vertex with second highest degree (without forming loops)...

5. construct trees with many leaves [KW91]

6. construct trees with minimum diameter [HLCW91]

Remark 5.20. An edge is called *vital arc* if its deletion strictly increases the length of the shortest path joining two vertices. It is a *most vital arc* if its deletion increases the path length by the maximal amount. There are algorithms of $O(nm)$ to identify such an arc [ALO93].

Remark 5.21. It is of use to determine the number of cycles an edge is contained in. A further information would be the length of these cycles.

Example 5.22. On the left tree of Figure 5.5, chord e is contained in five fundamental cuts and chord f in nine. This can easily be seen looking at the length of the paths connecting their endpoints. Each of the branches is cut once, so in the case that all edges have weight 1, we have a total cut weight of 23. However, on the right side, the cut weight is only 19, as both chords are contained in five fundamental cuts each.

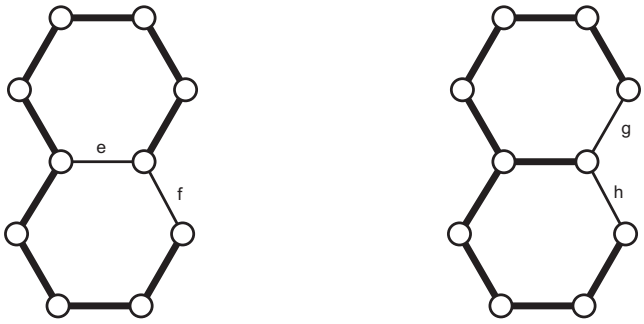


Figure 5.5: Shortening cycles

5.2.3 Combination of Heavy and Short Tree

The preceding algorithms can be combined in different ways:

- They can be used to reduce a given graph. In a first step, this hybrid procedure determines the Heavy Tree and the Short Tree. None but their edges are used as input for a formulation of Chapter 3. This approach reduces the number of constraints and variables.
- A bicriterial approach can be done. E.g. one looks for the shortest tree in a set of heavy trees.
- The algorithms can be used to get an input for genetic algorithms.
- A graph can be reduced to its heaviest edges (such that this heavy subgraph is connected) among which one looks for the Short Tree.
- The longest path in a heavy tree can be shortened.

5.2.4 Feasible Cut Tree

It has already been mentioned that the polynomial algorithm of Gomory and Hu yields an optimal solution of the Minimum Cut Basis Problem (see Section 4.2). The edges of the Gomory-Hu tree are not necessarily edges of the original graph, hence, it may not give a fundamental and feasible solution of the (MinFCutB).

The following algorithm is a *dual* one, i.e. it starts with an infeasible solution and modifies it until it becomes feasible.

Algorithm 5.23. *Feasible Cut Tree*
Input: undirected graph $G = (V, E)$*, weight function* $w : E \to \mathbb{R}$
Output: tree T
Calculate a cut tree T_{GH} *by the algorithm of Gusfield*
Determine the set of edges of T_{GH} *which are not contained in* G*,* $X := E(T_{GH}) \setminus E(G)$
Initialize $T := T_{GH}$
while $X \neq \emptyset$
Choose $e \in X$
Determine the shores U *and* V *arising by the deletion of* e *from* T_{GH}
Execute an edge swap with e *and an edge* $f \in (U, V)$*,* $T := T \setminus e \cap f$
$X := X \setminus e$

Given edge $e \in X$ there are different criteria to choose edge f for the edge swap:

- calculate the change in total cut weight for each edge in Q_e and execute the best edge swap possible,
- choose the heaviest edge in Q_e, according to the theory of Heavy Tree,
- choose f such that it is contained in a minimum number of fundamental cuts, thus the good solution is changed as slightly as possible.

5.3 Improving initial solutions

The solutions obtained by the preceding tree growth algorithms may be far from optimal. Therefore, we work at the initial solutions to obtain improvements of the objective function value. The algorithms in this section are so called *primal* algorithms, i.e. they improve solutions which are already feasible and maintain feasibility throughout this process.

5.3.1 Edge Swap

In this section a technique will be explained which can be used in a local search frame. It changes an initial solution (e.g. a heavy or a short tree) by just two edges. The results of this section are given by [ALMM03b] and [ALMM04], except for Remark 5.30.

Definition 5.24. Given a graph G and a spanning tree T, an edge exchange is defined by a branch e and a chord f. This so called *edge swap* $\pi = (e, f)$ is realized by removing e from and adding f to the tree. π is defined on the set $\mathcal{T}$ of all spanning trees of G.

Remark 5.25. An edge swap π is well-defined if and only if $f \in t_e$. In this case we get another spanning tree, $\pi T = T'$.

Remark 5.26. As mentioned before, spanning trees are related to fundamental cut bases. We define a mapping $m : \mathcal{T} \to \mathcal{F}$ where $\mathcal{T}$ is the set of spanning trees and $\mathcal{F}$ is the set of fundamental cut bases. This mapping is bijective for the cuts, see Theorem 2.80. Therefore swapping edges in a cut with cardinality two can improve the objective function value as opposed to the (MinFCycB) (see [ALMM03a]).

Performing an edge swap, how does the total cut weight change? The following theorems help to implement edge swaps, e.g. in a local search framework, more efficiently.

Theorem 5.27. *Let a graph G and a tree T be given. Let a branch e and a chord $f \in t_e$ define an edge swap $\pi = (e, f)$ which yields a new tree $T' = T \backslash e \cup f$. There are three cases of modified fundamental cuts defined by $h \in T'$:*

1. *The new fundamental cut defined by f, t'_f consists of the same edges as the old fundamental cut defined by e, i.e. $t'_f = t_e$.*
2. *If f is not contained in t_h then $t'_h = t_h$.*
3. *If f is contained in t_h then $t'_h = t_h \triangle t_e$.*

Proof. To show: T' is a spanning tree
T' has the same number of edges as T, namely $n - 1$. Furthermore, it is still connected, because e partitions T into two shores and f joins these two shores by the definition of fundamental cuts.
$\Rightarrow T'$ is acyclic and thus a tree.
The proofs for the modified fundamental cuts are given in [ALMM04]. □

Example 5.28. Theorem 5.27 is pictured in Figure 5.6. The edges e and f are swapped. The edges which belong to the fundamental cut defined by e are the same edges as in the cut defined by f (*1.*). The fundamental cut defined by h_2 does not change (*2.*). We get t'_{h_3} by means of the symmetric difference $t_{h_3} \triangle t_e$ (*3.*).

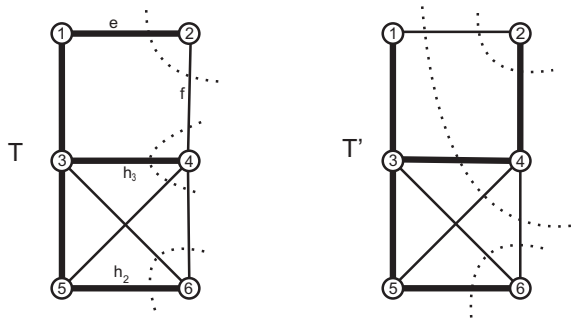


Figure 5.6: Edge Swap

Corollary 5.29. *Keeping the denotations of Theorem 5.27, the cut weights change as follows:*

1. $w_{t'_f} = w_{t_e}$
2. $f \notin t_h \Rightarrow w_{t'_h} = w_{t_h}$
3. $f \in t_h \Rightarrow w_{t'_h} = w_{t_h} + w_{t_e} - 2 \cdot \sum_{g \in t_h \cap t_e} w_g$

Proof. In the third case the sum is subtracted twice, as its edges are considered in both fundamental cuts and do not appear in the modified fundamental cut at all. □

Remark 5.30. The third case is the only one yielding a change of the total cut weight. In this case the change of a single cut weight is:

$$w_{t'_h} - w_{t_h} = w_{t_e} - 2 \cdot \sum_{g \in t_h \cap t_e} w_g. \tag{5.4}$$

We know that f is in the fundamental cuts defined by e and h. Thus we get

$$\sum_{g \in t_h \cap t_e} w_g \geq w_f \tag{5.5}$$

and combining Equations (5.4) and (5.5):

$$w_{t'_h} - w_{t_h} \leq w_{t_e} - 2 \cdot w_f. \tag{5.6}$$

Note that the right hand side of (5.6) does not depend on the fundamental cut h containing f. An edge swap is advantageous for the (MinFCutB) if the change in total cut weight is negative, i.e. if the left hand side of (5.6) is

smaller than 0. This is definitely the case if we can find a chord whose weight is at least half of the weight of a fundamental cut containing it ($w_f \geq \frac{1}{2} w_{t_e}$). Checking this condition is not as complex as checking the sum of all the common edges in Equation (5.4).

Example 5.31. The median solution of Example 5.16 can be improved by an edge swap. On the left of Figure 5.7 the median solution is depicted. The cut weight of $t_{(1,4)}$ is 7. The only chord contained in this cut has weight 4, which is more than half of the cut weight. An edge swap $\pi = (e, f)$ is performed with $e = (1, 4)$ and $f = (4, 5)$. This gives a total cut weight of:

$$w_{t_{(1,2)}} + w_{t_{(1,3)}} + w_{t_{(4,5)}} + w_{t_{(1,5)}} = 12 + 20 + 7 + 20 = 59.$$

The objective function value has improved by 1.

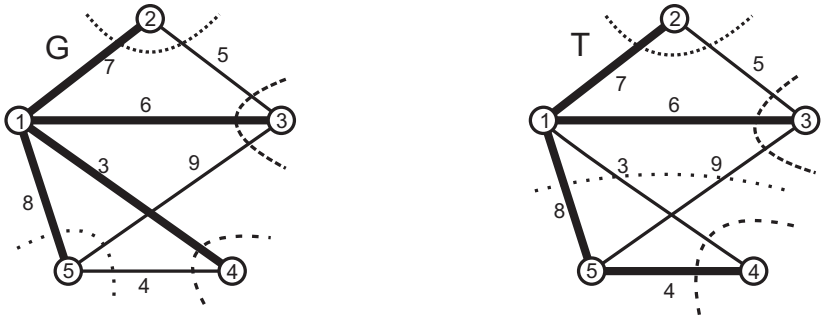


Figure 5.7: Median Tree and an improvement by means of an edge swap

Remark 5.32. One advantage of an edge swap is the saving in computations based on the results in this chapter. It is not necessary to recalculate all the single cut weights. Moreover, applying repeated edge swaps, it is possible to obtain all spanning trees of a graph out of any initial spanning tree, the whole "space" can be explored (see Section 2.3 on tree graphs).

5.3.2 Local Search

The above transformation can be used in a local search framework as will be described in the following.

Notation 5.33. Let $P = \{(e, f) | e \in T \text{ and } f \in t_e\}$ be the set of all edge swaps (any one changes the fundamental cut basis). For all $\pi \in P$ let $TCW(\pi T)$ be the cost of of the new fundamental cut basis. The cost reduction is denoted by $\Delta_\pi = TCW(T) - TCW(\pi T)$.

Given an initial spanning tree T, it makes sense to choose π in P such that $TCW(\pi T)$ is minimal. The corresponding edge swap is performed on T resulting in a new spanning tree T'. This move is iterated until no further improvement by means of edge swaps can be obtained. Summarizing, the costs of all cut bases whose spanning trees are adjacent to the initial spanning tree have to be calculated, the minimum over these costs has to be found and the new fundamental cuts have to be defined [ALMM03a].

Algorithm 5.34. *Local search algorithm ([ALMM04])*
Input: initial spanning tree
Output: locally optimal spanning tree
repeat
  $\Delta_{opt} := 0$
  initialize π_{opt} *to the identity*
  for all $e \in T$
    for all $f \in t_e$, $f \neq e$ *(*)*
      $\pi := (e, f)$
      if $\Delta_\pi \geq \Delta_{opt}$ **then**
        $\pi_{opt} = \pi$
        $\Delta_{opt} = \Delta_\pi$
  if π_{opt} *is not the identity* **then** $T := \pi_{opt} T$
until π_{opt} *is the identity*

Observe that this algorithm does not execute an arbitrary advantageous edge swap but the best edge swap possible. It terminates with a local optimum (there is no better spanning tree of distance one), hence a "good" initial spanning tree is advantageous.

Remark 5.35. The complexity of the algorithm is $O(m^2 n^2)$ if it is implemented straightforward without the differential calculations shown in Section 5.3.1 ([ALMM03b]).

According to Remark 5.30, loop $(*)$ can be substituted by:
for all $f \in t_e$, $f \neq e$ and $w_f \geq \frac{1}{2} w_{t_e}$.

5.4 Metaheuristics

5.4.1 Variable Neighbourhood Search

The local search algorithm as described before may get stuck in a local optimum. In order to look for better solutions, we have to leave the corresponding

solution even if we have to accept worse solutions to begin with. The core of the following algorithm is to comb increasingly larger neighbourhoods/ mutations of a locally optimal solution. This procedure is iterated until a given termination condition (e.g. neighbourhood size) is met. It jumps from an incumbent solution to another one as soon as this constitutes an improvement.

Algorithm 5.36. *Variable Neighbourhood Search*
Input: Locally optimal tree T, neighbourhood size k_{max}
Output: improved tree T
Set $k := 1$
repeat
    Execute k random edge swaps on T to obtain tree T'
    Apply local search with T' as input, T'' denotes the local optimum
    if $TCW(T'') < TCW(T)$ **then** $T := T''$, $k := 1$
    else *set* $k := k + 1$
until $k > k_{max}$

The variable neighbourhood search is described in detail in [HM03]. It can be applied in many fields and proves to be quite effective dealing with combinatorial optimization. Due to its restriction to a local search and a systematic mutation the VNS is simple to understand and implement. The idea is that many variables in a locally optimal tree are already at their optimal value, consequently they are kept on the search for a better solution. The random generation instead of a deterministic rule avoids cycling.

5.4.2 Genetic algorithm

A genetic algorithm imitates nature and its "survival of the fittest". It starts with a *population* of solutions and calculates their *strength* concerning the objective function. The strongest members of the population are combined to get *offspring*, this offspring is added to the population and the procedure is repeated until a certain stopping criterion is met:

Algorithm 5.37. *Genetic search (Idea)*
Input: 10 trees
Output: improved tree
repeat
    Calculate the total cut weight of the trees
    Take the best four of the trees
    Combine these four trees to get six new solutions

Add the new solutions to the best four
until *no improvement is obtained*

The points to be considered are the input, the combination, and the stopping criterion:
As an input, we can take arbitrary trees as well as Heavy Trees, locally optimal trees etc.
The combination should be done in such a way that the common edges of two trees are in their offspring as well. They have proved to be good in total cut weight. To make this forest a spanning tree missing edges can be added randomly or chosen alternately out of the symmetric difference.
The stopping criterion could be a maximum number of iterations or a minimal improvement to be reached.

Chapter 6

Running the algorithms

The algorithms of Chapters 4 and 5 are now tested for their effectiveness dealing with different types of graphs. The results of the (meta-)heuristics and relaxations are compared to each other and to the optimal solution in case of small instances. Furthermore, this chapter includes implementation details.

6.1 How to get graphs

The graphs to be investigated vary in the following characteristics:

- number of vertices n

  5 vertices (for which optimal solutions can be obtained)

  10 to 100 vertices

- density p

  graphs with density p from 0.25 to 1

  complete graphs

- weight span w

  unweighted graphs

  small or large weight span

  disjoint weight intervals

The random graphs are simulated for different combinations of the parameters n, p, and w. The generator starts with a set of n vertices and creates an edge between a pair of vertices with probability p. The edge weights are randomized in the weight span. In case of disjoint weight intervals, I give the probability of an edge weight to be in a certain interval. For some relaxations and heuristics, the graph has to be connected or even biconnected.

Another idea is to randomize graphs with n vertices and density p on a plane. The edge weight is equal to the Euclidean distance between its adjacent vertices. This is a *Euclidean graph*.

6.2 Special graphs

Complete graphs

In case of complete graphs the Gomory-Hu tree is always feasible for the (MinFCutB) as its edges are contained in the graph.

Mesh graphs

Mesh graphs are known to pose quite a challenge to the (MinFCycB). The problem is their uniformity in vertex degree and weights. Therefore it becomes difficult to choose an edge in this symmetrie [MAG+03].

Planar graphs

Definition 6.1. A graph is *planar* if it can be embedded into the plane such that no edges intersect.

Definition 6.2. The *dual graph* G^* of a planar graph G has its vertices corresponding to the faces in G and the vertices of G^* are joined by an edge if the faces in G are adjacent.

Planar graphs can be dualized. Note that the dual depends on the representation of the planar graph. To avoid multiple edges in the dual graph each vertex has at least degree three. A planar graph and its dual are depicted in Figure 6.1

The (MinFCutB) can be solved as (MinFCycB) in the dual graph and vice versa. That means that we can use the best available algorithm for cut or cycle bases in planar graphs.

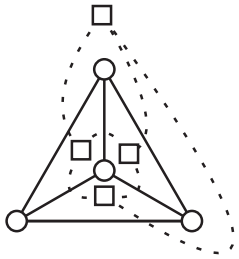


Figure 6.1: Planar graph and its dual

Cactus graphs

Definition 6.3. A *cactus* is a connected graph where each edge is contained in at most one cycle.

Any pair of cycles in a cactus is edge-disjoint. An example of a cactus is depicted in Figure 6.2.

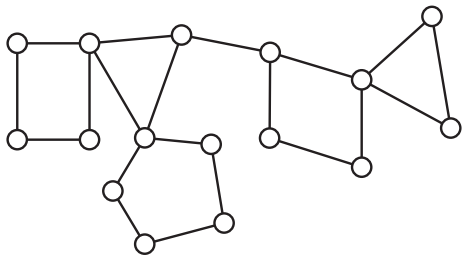


Figure 6.2: Cactus graph

The Heavy Tree algorithm solves the (MinFCutB) in a cactus. This is true as the cycles in a cactus can be treated independently. The total cut weight of each cycle is equal to the tree weight and the product of the weight of the single chord with the path length, e.g. in a cycle with n edges and chord f, we get the total cut weight $\sum_{e \in T} w_e + (n-1)w_f$. Hence, it is best to choose the edge with lowest weight as chord.

6.3 Numerical results

Implementation details

All the tests have been carried out on an AMD Athlon machine with 256 GB RAM running Windows XP. The source code has been written in Xpress and compiled with IVE.

6.3.1 Different IP-formulations

Four formulations of Chapter 3 have been implemented, the formulation with shores, complete, without y- or t-constraints, and the modified (MinFCycB)-formulation. In order to get the best one possible, I compared the time needed for the global search. Table 6.1 shows the mean values for different parameters calculated with ten instances each. All the formulations find the optimal solution quite fast, the time depends on the relaxation used in the branch and bound scheme.

	Shores	Shores without y	Shores without t	Cycles
n=4, p=1	1.3748	22.2231	> 120.0	4.3534
n=5, p=0.6	15.2631	7.1994	16.2805	3.9415

Table 6.1: Mean time for global search in seconds

The modified (MinFCycB)-formulation behaves best in average. However it is not used in the following tests as it requires biconnectivity. For the following tests, I used the formulation without y-constraints.

6.3.2 Comparison

The quality of the introduced heuristics is tested for different combinations of the input parameters graph size n, density p, weight span w, disjoint weight intervals w_1 and w_2, probability for the weight to be in the first interval p_1, and maximal neighbourhood size k. Twenty random graphs have been generated for each combination of input parameters. The abbreviations used in the tables are the following:

VNS	Variable Neighbourhood Search
LB	Quality of the lower bounds (Cut Tree/ LP-Relaxation)
FCT	Feasible Cut Tree
HT	Heavy Tree
MT	Median Tree
CT	Center Tree

The duality gaps are obtained by division of the initial solution and VNS solution resp. by the cut tree. The figures shown in the tables are the mean values of the duality gaps of parameter variation.

Graph structure

		Duality gap Heuristic				Duality gap VNS			
n	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
5	1.339	1.003	1.049	1.091	1.094	1.001	1.001	1.001	1.001
10	1.715	1.042	1.375	1.021	1.043	1.026	1.021	1.017	1.017
15	1.819	1.101	1.74	1.08	1.165	1.073	1.117	1.065	1.113
20	1.865	1.088	1.978	1.058	1.158	1.071	1.131	1.052	1.127
25	1.891	1.161	2.323	1.098	1.205	1.123	1.139	1.091	1.184
30	1.912	1.121	2.505	1.096	1.194	1.107	1.133	1.09	1.175
40		1.127		1.1					
60		1.169		1.105					
80		1.19		1.126					
100		1.188		1.135					

Table 6.2: Influence of the graph size ($p = 0.75$, w in [1,10], $k = 5$)

Table 6.2 demonstrates the influence of the graph size on the advantageousness of the heuristics. The larger the graph the worse the behaviour of the heuristics which becomes especially apparent at the heavy tree heuristic. This is due to the increasing importance of having short paths which is not accounted for in the heavy tree. In the worst case the maximum spanning tree is a Hamiltonian path.
The median tree yields the best results for the heuristic and the variable neighbourhood search except for instances with five vertices where the feasible cut tree and the heavy tree are to be preferred. The large graphs have not been tested with VNS due to the computation times. However, the median tree can be obtained pretty fast and shows a passable duality gap of 13.5%.

		Duality gap Heuristic				Duality gap VNS			
p	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
0.25	1.586	1.16	1.141	1.274	1.344	1.044	1.046	1.045	1.05
0.5	1.777	1.196	1.507	1.173	1.256	1.109	1.103	1.099	1.124
0.75	1.819	1.101	1.74	1.08	1.165	1.073	1.117	1.065	1.113
1.0	1.839	1.0	1.762	1.014	1.014	1.0	1.008	1.014	1.014

Table 6.3: Influence of the density ($n = 15$, w in [1,10], $k = 5$)

		Duality gap Heuristic				Duality gap VNS			
p	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
0.5	1.169	1.0	1.009	1.112	1.111	1.0	1.0	1.0	1.0
0.75	1.339	1.003	1.049	1.091	1.094	1.001	1.001	1.001	1.001
1.0	1.472	1.0	1.059	1.096	1.096	1.0	1.0	1.001	1.002

Table 6.4: Influence of the density ($n = 5$, w in [1,10], $k = 5$, $k = 0$ for $p = 0.5$)

The influence of the graph density is shown in Tables 6.3 and 6.4. We observe that the LP-relaxation gives worse solutions with increasing density. This corresponds to the observation that each cut contains just one branch in the relaxation which is far from reality in case of dense graphs. The feasible cut tree yields good solutions, in complete graphs as a matter of course the optimal one. The initial short trees become better, the heavy tree worse with increasing density. Furthermore, we can observe that the heavy tree yields better solutions than the short trees in graphs with only a few vertices.

Weight structure

		Duality gap Heuristic				Duality gap VNS			
w	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
$[1, 10]$	1.819	1.101	1.74	1.08	1.165	1.073	1.117	1.065	1.113
$[1, 100]$	1.808	1.119	1.702	1.094	1.167	1.08	1.118	1.072	1.11
$[1, 1000]$	1.814	1.11	1.638	1.076	1.161	1.075	1.108	1.062	1.114
$[1, 10000]$	1.816	1.099	1.742	1.072	1.139	1.07	1.094	1.057	1.099

Table 6.5: Influence of the weight ($n = 15$, $p = 0.75$, $k = 5$)

The weight intervals tested in Table 6.5 do not influence the advantageousness of the different heuristics. The best solutions are obtained by the median tree, the second best solutions by the feasible cut tree. The heavy tree solution does not yield an acceptable duality gap.

		Duality gap Heuristic				Duality gap VNS			
w_2	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
[91, 100]	1.703	1.152	1.288	1.144	1.261	1.039	1.046	1.057	1.075
[991, 1000]	1.594	1.131	1.264	1.245	1.29	1.026	1.023	1.076	1.069
[9991, 10000]	1.478	1.083	1.183	1.319	1.489	1.023	1.024	1.096	1.103

Table 6.6: Influence of the weight ($n = 15$, $p = 0.75$, $k = 5$, w_1 in [1,10], $p_1 = 0.75$)

For the tests shown in Table 6.6 an edge weight was randomized in the first interval with probability 75% and in the second interval with 25%. The two intervals have been "torn apart". With increasing distance of the intervals the short trees are less and the heavy tree is more efficient. Heavy tree and feasible cut tree show the best behaviour if improved by variable neighbourhood search.

		Duality gap Heuristic				Duality gap VNS			
p_1	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
0.0	1.829	1.077	1.84	1.08	1.166	1.07	1.133	1.074	1.133
0.45	1.801	1.115	1.754	1.073	1.154	1.063	1.086	1.06	1.105
0.5	1.795	1.061	1.654	1.05	1.081	1.041	1.064	1.045	1.059
0.55	1.774	1.121	1.608	1.078	1.1	1.051	1.063	1.053	1.085
0.6	1.743	1.172	1.433	1.138	1.217	1.07	1.058	1.087	1.099
0.65	1.695	1.102	1.421	1.115	1.169	1.033	1.05	1.062	1.071
0.7	1.707	1.079	1.363	1.104	1.144	1.025	1.043	1.054	1.052
0.75	1.594	1.131	1.264	1.245	1.29	1.026	1.023	1.076	1.069
0.8	1.406	1.139	1.103	1.372	1.395	1.01	1.007	1.08	1.055
0.85	1.219	1.072	1.048	1.609	1.719	1.005	1.005	1.054	1.032
0.9	1.156	1.015	1.03	1.655	1.761	1.002	1.004	1.027	1.016
0.95	1.114	1.024	1.106	1.633	1.746	1.012	1.014	1.013	1.019
1.0	1.819	1.101	1.74	1.08	1.165	1.073	1.117	1.065	1.113

Table 6.7: Influence of the weight spread ($n = 15$, $p = 0.75$, w_1 in [1,10], w_2 in [991,1000], $k = 5$)

The last investigation concerning the influence of the weight is shown in Table 6.7. The probability of the weight to be in the first of two intervals has been modified. Except for the cases with probability 0 and 100%, the heavy tree behaves better and the short trees behave worse for increasing probability of a lower weight. The advantageousness between heavy and short tree has a break even at probability 75% .

Neighbourhood size

		Duality gap Heuristic				Duality gap VNS			
k	LB	FCT	HT	MT	CT	FCT	HT	MT	CT
0	1.802	1.091	1.637	1.073	1.157	1.065	1.085	1.062	1.128
5	1.819	1.101	1.74	1.08	1.165	1.073	1.117	1.065	1.113
10	1.81	1.111	1.682	1.093	1.189	1.076	1.094	1.073	1.099
15	1.82	1.081	1.703	1.045	1.104	1.048	1.065	1.037	1.045

Table 6.8: Influence of the neighbourhood size ($n = 15$, $p = 0.75$, w in [1,10])

Enlarging the size of the neighbourhood to be combed improves the VNS-duality gaps (see Table 6.8) but has to be paid off by higher computation times.

Summary

Summing up the above results the feasible cut tree is the most reliable heuristic. The center tree is never better than the median tree. The median tree yields good results except for very small and sparse graphs and spread weight intervals. In this cases the heavy tree is more efficient. Overall, the worst case solutions are much better of the median than of the heavy tree. The VNS-duality gap of the respectively best procedure is always below 10% with a neighbourhood size of only five.

Chapter 7

Conclusion

7.1 What has been done

Some new observations concerning the differences between the cut and the cycle problem can be found in the first part of this thesis. Formulations for the Minimum Fundamental Cut Basis Problem and a new objective function have been introduced. Its optimal solution has been approached from above and below. Relaxing fundamentality has proved to yield reasonable lower bounds. When it comes to upper bounds, there is no optimal heuristic. Graph size, density and the weight function have been identified as factors influencing the advantageousness of the heuristics. The feasible cut tree has shown the most stable performance under varied input parameters. If the initial solution is improved by means of the variable neighbourhood search the best duality gap does not exceed 10% .

7.2 Further work

(MinFCutB) in directed graphs

Lagrangian Relaxation

The Lagrangian Relaxation can be used to reduce the number of constraints. The reduced subproblem can be solved faster than the LP-Relaxation. If the solution of the subproblem violates constraints of the original problem, the corresponding constraints are added to the subproblem cutting off the infeasible solution. The Lagrangian Relaxation can be solved exactly for

minor instances. For larger ones there are heuristics to find good Lagrangian coefficients.

Greedy Heuristic

A greedy heuristic for the (MinFCutB) should always add the next best feasible cut to a cut basis. Therefore we need a set of cuts and a ranking, i.e. an ordering corresponding to the cut weight. In theory one can generate all cuts by means of vertex partitions, i.e. binary vectors in $\mathbb{B}^n$. There are 2^n binary vectors. Are there ways to construct smaller sets which contain a nearly optimal cut basis?

Short Tree

There may be other ways to find a short tree, e.g. those listed in Remark 5.19.

Does it make sense to interpret the vertex degree as weight and leaves are chosen such that their weight is as low as possible? This ensures that the barycenter of a graph is not a leaf.

The median solution considers the paths between any pair of vertices, not only between adjacent pairs of vertices. Is there an approach to exclude the paths between non-adjacent vertices?

"Traditional" algorithms

There are several spanning tree growth algorithms for (MinFCycB)given in [DPK82], the implementation is explained in [Hub03]. Can these procedures be adapted to the cut problem?

Local search

The local search procedure has a high complexity. Is it possible to accelerate the search? The sampling method has proved to yield good results for the cycle problem (see [ALMM04]). Is there a similar procedure for the cuts? Is there another modification of an incumbent than the edge swap?

Taboo Search

The taboo search is currently under investigation in the Politecnico di Milano for the cycle problem.

Gomory-Hu

It has already been mentioned that Gomory-Hu yields optimal solutions in complete graphs. One can look for ways to construct complete graphs out of given graphs. This should happen in such a way that the auxiliary edges are not part of the solution.

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Massachusetts, 1974. (MAT Aho).

[ALMM03a] E. Amaldi, L. Liberti, N. Maculan, and F. Maffioli. Local search for the minimum fundamental cycle basis problem. EUME, Antwerpen, 2003.

[ALMM03b] E. Amaldi, L. Liberti, N. Maculan, and F. Maffioli. The minimum fundamental cycle basis problem: A new heuristic based on edge swaps. SYMOPIS, Herceg Novi, 2003.

[ALMM04] E. Amaldi, L. Liberti, N. Maculan, and F. Maffioli. Efficient edge-swapping heuristics for finding minimum fundamental cycle bases. WEA, Angra dos Reis, Brazil, 2004.

[ALO93] R. K. Ahuja, Magnanti T. L., and J. B. Orlin. *Network Flows*. Prentice Hall, New Jersey, 1993. (MAT Ahuj).

[Ber62] C. Berge. *The theory of graphs and its applications*. Wiley, New York, 1962.

[BHM05] F. Bunke, H. W. Hamacher, and F. Maffioli. On minimum (fundamental) cut bases of a graph. ORBEL19, Louvain-la-Neuve, 2005.

[BHMS05] F. Bunke, H. W. Hamacher, F. Maffioli, and A. Schwahn. On the minimum (fundamental) cut basis problem. Report in Wirtschaftsmathematik, 2005.

[BP01] A. Brambilla and A. Premoli. Rigorous event-driven (red) analysis of large-scale nonlinear rc circuits. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 48:938–946, 2001.

[Bun06] F. Bunke. *Minimal Circuit Bases of Matroids*. PhD thesis, TU Kaiserslautern, 2006.

[CR77] D. B. Corneil and R. C. Read. The graph isomorphism disease. *Journal on Graph Theory*, 1:339–363, 1977.

[CT76] D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM Journal on Computing*, 5:725–742, 1976. (INF Z 1436).

[Cum66] R. L. Cummins. Hamiltonian circuits in tree graphs. *IEEE Transactions on Circuit Theory*, 13:82–90, 1966.

[Deo79] N. Deo. Minimum length-fundamental cycle set. *IEEE Transactions on Circuits and Systems*, 26:894–895, 1979. (ELT Z 2653).

[DKP95] N. Deo, N. Kumar, and J. Parsons. Minimum-length fundamental-cycle set problem: New heuristics and an empirical investigation. *Congressus Numerantium*, 107, 1995.

[DPK82] N. Deo, G. M. Prabhu, and M. S. Krishnamoorthy. Algorithms for generating fundamental cycles in a graph. *ACM Transactions on Mathematical Software*, 8:26–42, 1982.

[Flo62] R. W. Floyd. Algorithm 97, shortest path. *Communications ACM*, 5:345, 1962. (INF Z 1415).

[Gal01] G. Galbiati. On min-max cycle bases. *Proceedings ISAAC 2001 in: Algorithms and Computations*, LNCS 2223:116–123, 2001.

[GDPS02] A. M. H. Gerards, J. C. De Pina, and A. Schrijver. *Shortest Circuit Bases of Graphs*. 2002. to appear.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of SIAM*, 9:551–570, 1961. (MAT Z 1054).

[GH85] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7:43–57, 1985.

[GH02] A. Golynski and J. D. Horton. A polynomial time algorithm to find the minimal cycle basis of a regular matroid. SWAT 2002, 2368:200–209, Scandinavia, 2002.

[Gib85] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge, 1985. (MAT Gibb).

[GS99] P. M. Gleiss and P. F. Stadler. Relevant cycles in biopolymers and random graphs. Fourth Slovene International Conference in Graph Theory, 1999.

[Gus90] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19:143–155, 1990. (INF Z 1436).

[HK00] H. W. Hamacher and K. Klamroth. *Lineare und Netzwerk-Optimierung*. Vieweg, Braunschweig, 2000.

[HLCW91] J. M. Ho, D. T. Lee, C. H. Chang, and C. K. Wong. Minimum diameter spanning trees and related problems. *SIAM Journal on computing*, 20:987–997, 1991. (INF Z 1436).

[HM03] P. Hansen and N. Mladenovic. Variable neighbourhood search. *Handbook of Metaheuristics*, 2003. (MAT Hand).

[Hor87] J. Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal of Computing*, 16:358–366, 1987. (INF Z 1436).

[Hub03] M. Huber. Implementation of algorithms for sparse cycle bases of graphs, 2003. Project report.

[JLRK78] D. S. Johnson, J. D. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8, 1978.

[Jr.65] E. Sussenguth Jr. A graph theoretical algorithm for matching chemical structures. *Journal of Chemical Documentation*, 5:36–43, 1965.

[Jun94] D. Jungnickel. *Graphen, Netzwerke und Algorithmen*. B.I., Mannheim, 1994. (MAT Jun).

[Kir47] G. Kirchhoff. über die auflösung der gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer ströme geführt wird. *Annalen der Physik und Chemie*, 72:495–508, 1847.

[Knu68] D. E. Knuth. *The Art of Computer Programming Vol. 1.* Addison-Wesley, Massachusetts, 1968.

[Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the AMS*, 7:48–50, 1956. (MAT Z 1046).

[KV00] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, 2000. (MAT Kort).

[KW91] D. J. Kleitman and D. B. West. Spanning trees with many leaves. *SIAM Journal on discrete mathematics*, 4:99–106, 1991. (MAT Z 1318).

[LAMM03] L. Liberti, E. Amaldi, N. Maculan, and F. Maffioli. Mathematical models and a constructive heuristic for finding minimum fundamental cycle bases. Manuscript, 2003.

[LD60] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems econometrica. 28:497–520, 1960.

[Led68] J. Lederberg. Topology of molecules. *Mathematical Sciences*, pages 37–51, 1968.

[Lib05] L. Liberti. Mail, April 2005.

[Lie03] C. Liebchen. Finding short integral cycle bases for cyclic timetabling. Internal report, TU Berlin, 2003.

[MAG+03] F. Maffioli, E. Amaldi, G. Galbiati, L. Liberti, and N. Maculan. On the problem of finding miniumum fundamental cycle bases. ISMP Copenhagen, 2003.

[Pat69] K. Paton. An algorithm for finding a fundamental set of cycles of a graph. *Communications of the ACM*, 12:514–518, 1969. (INF Z 1415).

[Sch03] A. Schrijver. *Combinatorial Optimization*. Springer, Heidelberg, 2003. (MAT Schrij).

[Sch04] M. Schröder. Competitive location theory: Models and algorithms, 2004. Lecture Notes.

[SR61] S. Seshu and M. B. Reed. *Linear Graphs and Electrical Networks*. Addison-Wesley, Massachusetts, 1961. (EIT 608/008).

[Sys79] M. M. Sysło. On cycle bases of a graph. *Networks*, 9:123–132, 1979.

[Sys81] M. M. Sysło. On some problems related to fundamental cycle sets of a graph. *Theory of Applications of Graphs*, pages 577–588, 1981.

[Sys82] M. M. Sysło. On the fundamental cycle set graph. *IEEE Transactions on Circuits and Systems*, 29:136–138, 1982. (ELT Z 2653).

[WC04] B. Y. Wu and K. M. Chao. *Spanning Trees and Optimization Problems*. Chapman & Hall, 2004.

[WCT00] B. Y. Wu, K.-M. Chao, and C. Y. Tang. Approximation algorithms for the shortest total path length spanning tree problem. *Discrete Applied Mathematics*, 105:273–289, 2000. (MAT Z 1284).