technische universität
dortmund

Master's thesis

Efficient Algorithms for the Maximum Common Subgraph Problem in Partial 2-Trees

Florian Kurpicz

April 7, 2014

Supervisors:
Prof. Dr. Petra Mutzel
Dipl.-Inf. Nils Kriege

Faculty of Computer Science
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.tu-dortmund.de

Contents

Chapter 1

Introduction

A maximum common subgraph of two graphs G and H is a graph isomorphic to subgraphs of both G and H such that there is no greater graph isomorphic to any subgraph of G and H. In this thesis, the size of these common subgraphs is measured by the number of vertices of the graph. The size of a maximum common subgraph is an indicator for the similarity of those graphs. Since graphs are a widespread data structure which can represent objects and their relations, they are used in many areas, like chemistry[25, 19] or pattern recognition[9].

The maximum common subgraph problem denotes the problem of finding the size of a maximum common subgraph of two graphs. The problem is well studied and there exist lot of results regarding the complexity in different graph classes. It is known to be **NP**-complete if the maximum common subgraph is not required to be connected. Therefore, the problem considered in this thesis always requires a common subgraph to be connected. Also, it is known to be solvable in polynomial time in trees [1], in connected almost trees of bounded degree [2] and if one input graph is a partial k-tree with bounded degree and the other input graph has a bounded number of spanning-trees[30].

In 2013 two papers [4, 3] regarding the complexity of the maximum common subgraph problem have been published. For two graph classes, the outerplanar graphs with bounded degree and partial 11-trees with degree greater than five, they show that the maximum common subgraph problem is either solvable in polynomial time or is **NP**-hard. There is still a huge gap in the hierarchy of graph classes, for which it is unknown whether the maximum common subgraph problem is solvable in polynomial time.

In this thesis, the maximum common subgraph problem in partial 2-trees is considered. Partial 2-trees are a direct superclass of the outerplanar graphs and it is known that there are restrictions for which the maximum common subgraph problem in partial 2-trees can be solved in polynomial time [20]. Different restrictions regarding the input graphs and feasible common subgraphs are considered and algorithms solving the problem in polynomial time

with respect to these restrictions are presented. Also, it is shown that there are restrictions which are not sufficient to guarantee the existence of a polynomial time algorithm.

This thesis is organized as follows. In Chapter 2 a basic introduction of the used definitions, notations and graph theory is given. The focus is set to graph decompositions, starting with the tree decomposition and a stricter normalized tree decomposition which can be applied to all graphs. Then SP-trees are introduced as data structure emphasizing the series and parallel components of a 2-connected series-parallel graph. Since partial 2-trees are mainly considered in this thesis which are not necessarily 2-connected, the BC-tree is introduced as data structure highlighting the 2-connected components of a graph. These 2-connected components are then associated with a SP-tree. These graph decompositions and the SP-trees and BC-trees in particular are then extensively used in the following chapters as basic data structure the algorithms work on.

In Chapter 3 it is shown that the maximum common subgraph problem in partial 2-trees is **NP**-hard even if all vertices except one in each graph has bounded degree. To do so, a polynomial-time reduction from the numerical matching with target sums problem to the maximum common subgraph problem in partial 2-trees is defined.

Then in the following chapters, different restrictions of the problem are discussed. First in Chapter 4 both the input graphs and the feasible common subgraphs are restricted to be 2-connected and it is shown that under this restriction the problem is solvable in polynomial time. Then in Chapter 5 the block-and-bridge preserving maximum common subgraph problem in partial 2-trees is considered. Therefore, the feasible common subgraphs are required to be block-and-bridge preserving. It is shown that the approach used for 2-connected graphs can be extended and then applied to this problem. Last, in Chapter 6 only the input graphs are restricted to have a bounded number of chordless cycles. It is shown that the problem can also be solved in polynomial time in these graphs.

Finally in Chapter 7 these results are summarized and resulting open problems are mentioned.

Chapter 2

Preliminaries

In this chapter the mathematical and graph theoretical concepts, needed in the further course, are discussed. Also the notation used in this thesis is explained. The first section deals with fundamental definitions regarding graphs. The second section introduces different graph decompositions, while the third section is about partial 2-trees and equivalent graph classes and their characteristics. In the last section different complexity classes and methods to proof the membership of problems to these classes are discussed.

2.1 Graphs

A *graph* $G = (V, E)$ is a pair consisting of a finite set V of *vertices* and a finite set E of *edges*. The set of edges consists of pairs of vertices, thus $E \subseteq\in V \times V$. There are two types of graphs: *directed* graphs and *undirected* graphs. If a graph is directed, the pairs are ordered. Otherwise, if the graph is undirected the pairs are unordered. Unless stated otherwise, undirected graphs are considered. If there is an edge $e = (u, v) \in E$ then u and v are *incident* to e also u is *adjacent* to v and vice versa. The *degree* of a vertex $v \in V$ is the number of edges incident to v and denoted by $\deg(v)$. For each graph G, $V(G)$ is denoting the set of vertices of the graph and $E(G)$ is denoting the set of edges of the graph.

A function $\lambda\colon E \cup V \mapsto \mathbb{N}$ is called a *graph labeling*. A *labeled graph* is a graph $G = (V, E)$ with an additional graph labeling λ_G. In this thesis all graphs are considered to have no graph labeling, but as long as not stated otherwise all results can be applied on labeled graphs, too. If in any figure a graph contains labels its for the readers convenience – a help to better identify different parts of the graph.

There is a *path* between two vertices $u, v \in V$ (denoted by $u \rightsquigarrow v$) if there is a set of vertices $w_0, \ldots, w_k \in V$ such that w_{i-1} is incident to w_i for all $i = 1, \ldots, k$ and $w_0 = u$ and $w_k = v$. A graph is *connected* if for all vertices $u, v \in V$ there is a path $u \rightsquigarrow v$. An induced subgraph is called *component* if it is connected. If a graph is still connected after the removal of any $k - 1$ vertices, then the graph is *k-connected*.

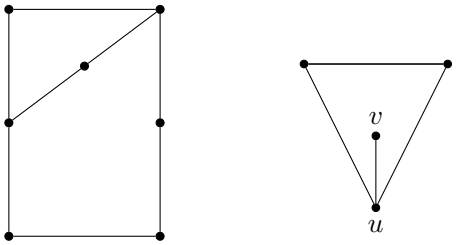


Figure 2.1: A 2-connected (a) and 1-connected undirected graph (b).

An induced subgraph is called *k-connected component*, if it is k-connected. Let $W \subseteq V(G)$ and $G[W] := (W, \{(u,v) \in E(G) \colon u,v \in W\})$ then $G[W]$ denotes the *induced* subgraph by W in G. If a graph G is connected and for any $v \in V(G)$ the graph $G \setminus v := G[V(G) \setminus \{v\}]$ is not connected, then v is called *cutvertex*. Also let $S \subseteq V(G)$, then $G \setminus S := G[V(G) \setminus S]$. If $G \setminus S$ is not connected the set S is called *separator*. Also let $\mathcal{C}(G \setminus S)$ denote the set of connected components caused by the removal of S. The graph in Figure 2.1(a) is 2-connected since no matter which vertex is removed the graph is still connected. The graph in Figure 2.1(b) on the other hand is only connected since the vertex u is a cutvertex because if u is removed, there is no path from any other vertex to v.

A graph G contains a *cycle* if there is a vertex $v \in V(G)$ and a path $v \rightsquigarrow v$ which contains at least one more vertex $u \in V$ such that $u \neq v$. If on the other hand the path contains only the edge (v,v), then the cycle is called *loop*. A graph without any cycles is called *forest*. If the graph is connected, then the forest is called *tree*. Let G be a tree. The vertices $v \in V(G)$ with $\deg(v) = 1$ are called *leaves*.

A graph $G = (V,E)$ is *bipartite* if there are two partitions called *bipartition* $V_1, V_2 \subseteq V$ such that $V_1 \cap V_2 = \emptyset, V_1 \cup V_2 = V$ and there is no $e = (u,v) \in E$ such that $u,v \in V_1$ or $u,v \in V_2$.

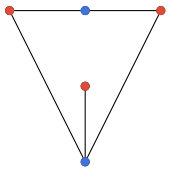


Figure 2.2: A bipartite graph. All vertices in a bipartition are either blue or red.

Let $G = (V,E)$ be a graph. A *matching* is a subset $M \subseteq E$ of edges such that for all $v \in V$ there is at most one edge $e \in M$ with e is incident to v. Hence each vertex is incident to at most one edge in the matching. Let $G = (V,E)$ be a graph and $w \colon E \to \mathbb{N} \cup \{-\infty\}$ be a weighting function. The *maximum weighted matching problem* denotes the problem of finding a matching M such that for each other matching M' $\sum_{e \in M} w(e) \geq \sum_{e \in M'} w(e')$. The maximum weighted bipartite matching problem denotes the maximum matching prob-

lem on bipartite graphs. In this thesis a function MwbMatching(M, M', w) is used. This function returns the size of a maximum weighted matching of the bipartite graph $(M \cup M', M \times M')$ and a given weighting function.

A *complete graph* is a graph in which each disjunct pair of vertices is connected by an edge. K_n denotes a complete graph with n vertices. A complete bipartite graph is a graph in which each two vertices with both vertices being in a different bipartition are adjacent. $K_{n,m}$ denotes the complete bipartite graph in which one bipartition contains n vertices and the other contains m vertices.

A graph is *simple* if it contains no loops and at most one edge between any two vertices. In this thesis only simple graphs are considered. A graph is called *planar* if it can be drawn into the Euclidean plane so that no edge crosses another edge[1]. A graph is *outerplanar* if it has a crossing-free embedding in the plane such that all vertices are on the outer face, i.e. it is possible to draw a line starting at an arbitrary vertex with infinite length never crossing an edge of the graph.

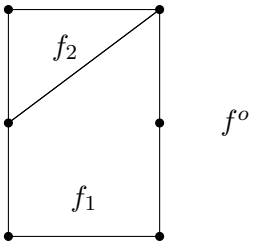
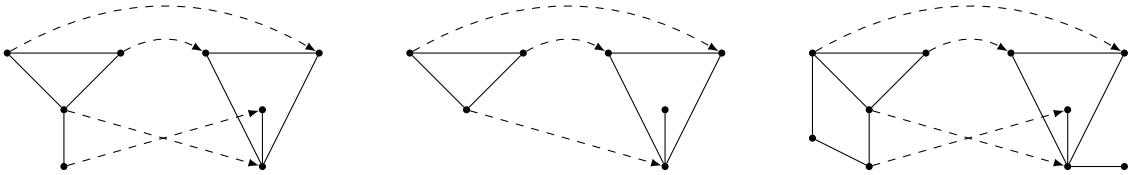


Figure 2.3: An outerplanar graph with inner faces f_1, f_2 and the outer face f^o.

Let G and H be graphs. If $V(H) \subseteq V(G)$ and $E(H)$ is a subset of the edges $E(G)$ which consists only of vertices out of $V(H)$, then H is called *subgraph* of G. G and H are *isomorphic*, denoted by $G \cong H$, if there is a bijection $\phi\colon V(G) \to V(H)$, such that $(u, v) \in E(G) \iff (\phi(u), \phi(v)) \in E(H)\ \forall u, v \in V(G)$. H is *subgraph isomorphic* to G (denoted by $H \preccurlyeq G$), if H is isomorph to a subgraph of G. There is a *common subgraph isomorphism* between G and H, if there is a set $R \subseteq V(G)$ and a set $S \subseteq V(H)$ such that the induced subgraphs $G[R]$ and $H[S]$ are isomorph. Let ϕ be the common subgraph isomorphism, then ϕ is a *maximum common subgraph isomorphism* if there is no common subgraph isomorphism ϕ' with $|\mathrm{dom}(\phi')| > |\mathrm{dom}(\phi)|$.

The *edge induced common subgraph* which is considered in [25, 26] is another type of common subgraph. The measure for the size of an edge induced common subgraph are the edges of the subgraph. The main difference is, that for an edge induced common subgraph of two subgraphs G and H of graphs G' resp. H' there must not be a bijection $\phi\colon V(G) \to V(H)$, such that $(u, v) \in E(G') \iff (\phi(u), \phi(v)) \in E(H')\ \forall u, v \in V(G)$. Instead only the subgraphs must be isomorphic.

[1] A much more detailed definition can be found in [23].



(a) (b) (c)

Figure 2.4: A graph isomorphism (a), subgraph isomorphism (b) and a common subgraph isomorphism (c).

Therefore an *edge induced maximum common subgraph*, which is defined analogous to the maximum common subgraph but with respect to the number of edges, is not necessarily equivalent to a maximum common subgraph. In [19] it is shown that under some restrictions[2] an edge induced maximum common subgraph induces an maximum common subgraph as defined above. In Figure 2.5 the highlighted red edges contradict the restriction of a maximum common subgraph. The vertices which would be contained in a maximum common subgraph are highlighted in green.

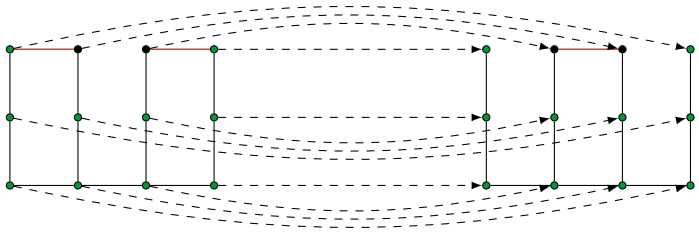


Figure 2.5: Example of an edge induced maximum common subgraph.

2.2 Graph decompositions

In this section different graph decompositions are presented. The main idea behind graph decompositions is to gain additional information about the graph by adding further structures to the graph in a way that different characteristics of the graph are highlighted. First in Section 2.2.1 and 2.2.2 two tree decompositions for graphs are presented which emphasize how a graph differs from a tree and also are used to define a measure of the difference. Then, in Section 2.2.3 a decomposition which focuses on the 2-connected components of a graph is presented. This graph decomposition is then extended by the previously presented decomposition. Last in Section 2.3 a tree decomposition which is only applicable

[2]The approach is based on line graphs, which represent the edges of a graph. In [29] it is shown that if two line graphs are isomorphic, then the graphs are isomorphic, too, with respect to the following exception. This does not work if any of the two graphs contains a K_3 or $K_{1,3}$ since these graphs have isomorphic line graphs but are not isomorphic.

to 2-connected graphs and mainly represents the 3-connected components of these graphs is presented. This decomposition is then used to extend the decomposition presented in Section 2.2.3 to be applicable to all partial 2-trees.

2.2.1 Tree decompositions

The *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathrm{TD}(G) = (T, \mathcal{X})$ where T is a tree and $\mathcal{X} = (X_z)_{z \in V(T)}$ is a family of subsets of $V(G)$, called *bags*, which satisfies the following conditions:

TD1 $\bigcup_{z \in V(T)} X_z = V(G)$,

TD2 $\forall (u, v) \in E(G)\ \exists z \in V(T)$ such that $u, v \in X_z$,

TD3 $X_{z_1} \cap X_{z_3} \subseteq X_{z_2}\ \forall z_1, z_2, z_3 \in V(T)$ if z_2 is contained in a path $z_1 \rightsquigarrow z_3$ in T.

The *tree width* of tree decomposition $\mathrm{TD}(G) = (T, \mathcal{X})$ is $\max_{z \in V(T)} \{|X_z| - 1\}$ and the tree width of a graph G (denoted by $\mathrm{tw}(G)$) is equal to the smallest tree width of all tree decompositions of G. It must be noted that the tree decomposition of a graph is not unique, as a tree decomposition with only one bag which contains all vertices of the graph is always a feasible tree decomposition. In the further course, if there is a tree decomposition of a graph G with $\mathrm{tw}(G) = k$, it is assumed, that all bags in a tree decomposition of G contain at most $k + 1$ vertices.

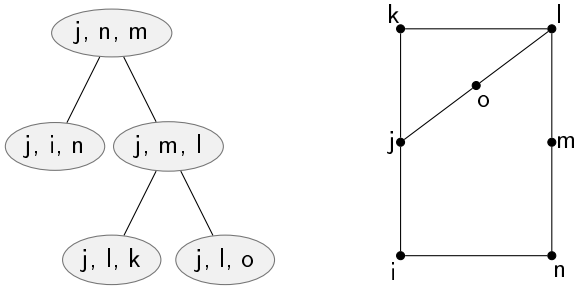


Figure 2.6: A tree decomposition (a) of the graph shown in (b).

The main idea of the tree width is to figure out, how tree-alike a graph is, as many hard problems can easily be solved in trees. The importance of the tree decomposition and the related tree width is shown by the fact, that many **NP**-complete problems can be solved in polynomial time in graphs with a bounded tree width. Popular examples are the *maximum independent set problem*[3] and the *Hamiltonian cycle problem*[4] [6]. It must be noted that the problem of finding the tree width of a graph is **NP**-complete [5], whereas

[3] Find the greatest set of vertices of which no two vertices are adjacent to each other [14, GT20].

[4] Does the graph contain a path which contains all vertices exactly once and also is a cycle [14, GT37]?

it can be checked in linear time whether a graph has tree width less or equal k for a fixed parameter k as proved in [7]. In the same paper it is shown, that the tree decomposition with tree width less or equal k can also be computed in linear time.

2.2.2 Normalized tree decompositions

The normalized tree decomposition presented in this section is based on [15]. Unlike the tree decomposition, which seems to have a standard definition in the literature, there are more definitions of a normalized tree decomposition which are not equal[5].

Let G be a connected graph. As mentioned before, a set $S \subseteq V(G)$ is called separator of G, if and only if $G[V(G) \setminus S]$ consists of two or more connected components. These separators are used in [15], where a stricter version of the tree decomposition, the *normalized tree decomposition*, denoted by $\mathrm{NTD}(G)$, is introduced. A normalized tree decomposition is a restricted tree decomposition denoted by $\mathrm{NTD}(G) = (T, \mathcal{X})$. The vertices of the tree T are divided into two types: The *clique nodes* denoted by C and the *separator nodes* denoted by S. For each separator node $s \in S$, the associated bag X_s must be a separator of G. Let G be a graph. A tree decomposition $\mathrm{TD}(G)$ is called normalized tree decomposition if the following conditions are satisfied in addition to the conditions (TD1) – (TD3):

NTD1 T is a bipartite graph regarding S and C and all leaves of T are clique nodes,

NTD2 $|X_s| \leq k \ \forall s \in S$ and $|X_c| \leq k+1 \ \forall c \in C$, where $k = \mathrm{tw}(G)$,

NTD3 for each three adjacent nodes (i, j, k) in $T : X_j = X_i \cup X_k$ if $j \in C$ and $X_j = X_i \cap X_k$ if $j \in S$.

Like the tree decomposition described in subsection 2.2.1 a normalized tree decomposition is not unique for a graph. For each graph with tree width k there is a normalized tree decomposition with tree width k. Therefore it is **NP**-complete to find a normalized tree decomposition with tree width equal to the three width of the graph but a normalized tree decomposition with size k can be found for any $k \in \mathbb{N}$ if there is any. A normalized tree decomposition can be constructed from a tree decomposition in polynomial time.

2.2.3 BC-trees

Since the restriction of only being applicable to 2-connected graphs is rather strict, there is a need for a decomposition which can represent even connected graphs. As unlike arbitrary graphs a partial 2-tree is at most 2-connected, the following definitions apply only for 2-connected graphs and may not apply for connected graphs. Let G be a graph. Each k-connected component with $k \geq 2$ of maximal size is called a *block*. There are two

[5]The normalized tree decomposition presented in [18] is a rooted tree decomposition with restrictions regarding the children based on the parent-child relation and without restriction NTD2.

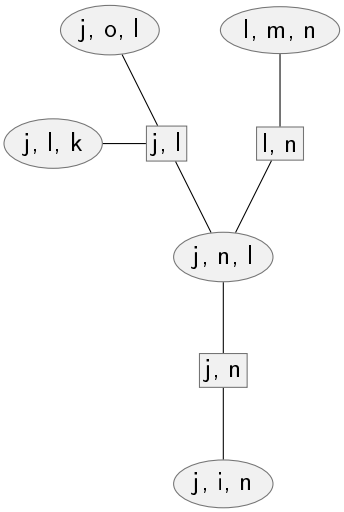


Figure 2.7: A normalized tree decomposition of the graph in Figure 2.6(b).

different types of blocks: A maximal 2-connected subgraph[6] and a *bridge* which denotes to a graph consisting of two vertices which are both incident to the same vertex. Due to the definition, any two blocks of G may have only one vertex in common, which must be a cutvertex. Otherwise these two blocks would be in the same 2-connected component. Blocks which are not bridges are called *non-bridge blocks*.

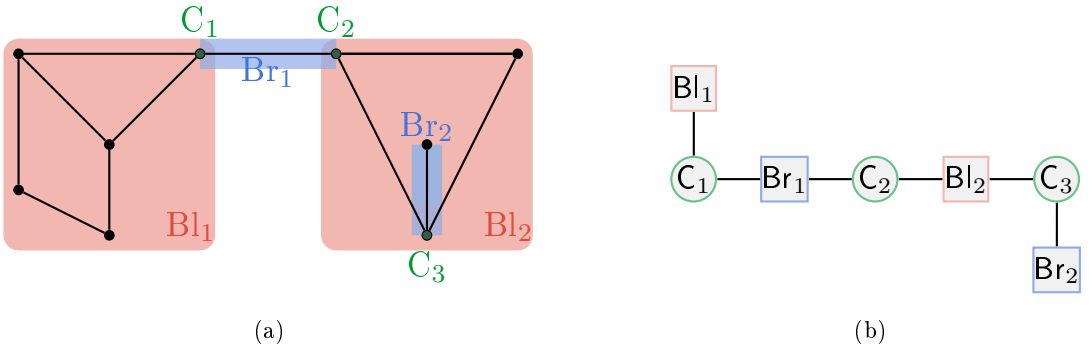
The *block graph* of a graph G is the graph which has all blocks and cutvertices of G as nodes[7] and an edge between two nodes, if one is associated with a cutvertex v while the other is associated with a block and the intersection of the vertices corresponding to the nodes contains exactly v. Therefore a block graph is bipartite with respect to the nodes associated with a cutvertex and the nodes associated with a block. All edges and vertices of G are contained in the blocks of G. The set of cutvertices of G is equal to the union of all pairwise disjoint intersections of blocks of G. Let B denote the set of blocks of G and C the set of cutvertices of G. The graph with vertices $B \cup C$, edges between each block $b \in B$ and cutvertex $c \in C$ if c is contained in b is called block graph of G and denoted by BC(G). Like the nodes in a SP-tree, the nodes in a BC-tree have a skeleton graph. In the case of a cutvertex the skeleton graph consists of a single vertex while in case of a block the whole subgraph is contained in the skeleton graph. Unlike the skeleton graphs in SP-trees there are no virtual edges.

A block graph contains two kinds of nodes: The *B-nodes* which represent blocks and the *C-nodes* which represent cutvertices. Due to its definition this graph is a forest and bipartite. If G is connected, the block graph is a tree as proved in Lemma 2.2.1 and referred

[6] For an arbitrary graph the subgraph may be more than 2-connected.

[7] In the course of this thesis the vertices of graph decompositions are called nodes while the vertices of the original graph are still called vertices. Therefore, it should be easier to distinguish the vertices of a decomposition and the vertices of the underlying graph.

to as *BC-tree*. It must be noticed, that in Figure 2.8 the B-nodes are also divided into two sets. The set Br contains all bridges while Bl contains all non-bridge blocks. Therefore $\mathrm{B} = \mathrm{Bl} \dot{\cup} \mathrm{Br}$. This extension of the BC-tree is needed in Section 2.3.3 where the non-bridge blocks are extended.



(a) (b)

Figure 2.8: A partial 2-tree with cutvertices highlighted in green, bridges in blue and non-bridge blocks in red (a) and the block graph of this graph (b).

Lemma 2.2.1. *The BC-tree of any connected graph G is a tree.*

Proof. Let G be a connected graph and $T = \mathrm{BC}(G)$ the BC-tree of G. There are two cases in which T would not be a tree: First T is not connected, second T contains a cycle. Let $B_1, \ldots, B_n$ be the B-nodes of T for $n \geq 2$. If there is just one B-node, T must be connected. If T is not connected, then there must be at least two B-node B_1 and B_2 such that there is no path $B_1 \rightsquigarrow B_2$ in T. Thus for any vertex v_1 associated with B_1 and v_2 associated with B_2 there are no cutvertices on a path $v_1 \rightsquigarrow v_2$ in G, but since v_1 and v_2 are in different blocks, G cannot be connected which is contradicting to the assumption.

Let $B_1, \ldots, B_n$ with $n \geq 2$ be the B-nodes which are contained in the cycle in T. The induced subgraph of G by the vertices in $B_1, \ldots, B_n$ would contain a cycle, as each B-node contains adjacent vertices and at least two B-nodes have one vertex in common. These B-nodes are 2-connected and may not be split into different B-nodes, therefore the cycle in T is reduced to a single B-node, so that T is cycle-free and it can be noted that a BC-tree cannot contain any cycles. □

2.3 Partial 2-trees

Each graph with tree width of at most two is a *partial 2-tree*[8]. Also partial 2-trees can be defined by one forbidden minor, the K_4 [28]. A more visual definition can be given with the help of *series-parallel graphs*. Let $K_2^{s,t}$ be a K_2 where one vertex is denoted by s and the other is denoted by t. A graph is a series-parallel graph if it can be computed from a finite set of $K_2^{s,t}$s with the following two operations merging two connected components by:

S-operation: merging the vertex denoted with s in one component with the vertex denoted by t in the other component or vice versa.

P-operation: merging both vertices denoted by s and both vertices denoted by t.

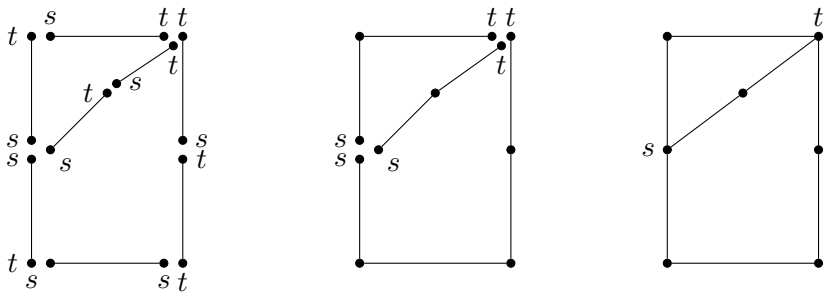


Figure 2.9: Construction of a series-parallel graph from a set of $K_2^{s,t}$ (a) using S-operations (b) and P-operations (c).

After each operation each connected component has exactly one vertex denoted by s and one denoted by t. In [12] it is shown that series-parallel graphs also have the K_4 as forbidden minor. This leads to the third equivalent definition of partial 2-trees. A graph G is a partial 2-tree if and only if each 2-connected component of G is a series parallel graph. Thus, each 2-connected partial 2-tree is a series-parallel graph and can be constructed as described above. It must be noted that a series-parallel graph constructed with the operations defined above is not necessarily 2-connected, as a sequence of S-operations results in a simple path. But the definition only requires 2-connected components to be series-parallel and therefore it is not required that each series-parallel graph is 2-connected.

2.3.1 SPQR-trees

SPQR-trees have been introduced in [11] for the first time and are a graph decomposition for 2-connected graphs. The SPQR-tree of a 2-connected graph G is denoted by SPQR(G). The idea of SPQR-trees is to represent all planar embeddings of the graph and are therefore unique for each graph. A SPQR-tree can be computed and updated in linear time [17]. Let G be a 2-connected planar graph and $T = \text{SPQR}(G)$. The SPRQ-tree of G is a tree in which each node represents a different part of the graph. A SPQR-tree has four different types of nodes:

S-node: Represents a cycle in G, concerning partial 2-trees and series-parallel graphs in particular this can be seen as the result of a sequence of S-operations.

P-node: Represents different paths between two vertices. Regarding series-parallel graphs this is the result of a sequence of P-operations.

Q-node: A single edge in G is represented by a Q-node. Often these nodes are integrated in the other node types and therefore omitted.

R-node: The rigid case, each subgraph of G which cannot be depicted as one of the other types is represented by a R-node. This also implies that the subgraph has a unique planar embedding[8].

In addition to its type each node λ in the SPQR-tree T is associated with a *skeleton graph* (denoted by skel(λ)). The skeleton graph of a node consists of all vertices and edges called *real edges* in G which are represented by the node λ_i. In addition there may be *virtual edges* in the skeleton graph. For each edge between two nodes λ_i and λ_j in the SPQR-tree, there is a virtual edge in the skeleton graphs skel(λ_i) and skel(λ_j) associated with the edge (λ_i, λ_j). The virtual edge is incident to the vertices which are contained in both skeleton graphs, which must be exactly two due to the construction of the SPQR-tree. The virtual edge in λ_i is called *pertinent* to λ_j and the virtual edge in λ_j pertinent to λ_i accordingly. The two vertices are a separator of the graph, which must be a set of two vertices, as the graph must be 2-connected.

Thus, the skeleton graph of a S-node is a cycle consisting of at least three vertices and real edges while the skeleton graph of a Q-node consists of exactly two vertices. Q-nodes can be omitted by simply adding read edges to the skeleton graphs. In the original definition each non-Q-node only consists of virtual edges pertinent to a Q-node. The P-node consists of exactly two vertices and virtual edges between them. There might also be a real edge between the two vertices. In this case the P-node is called *closed* and *open* otherwise.

(a) (b) (c) (d)

Figure 2.10: Example of the skeleton graph of a S-node (a), a P-node (b), a Q-node (c) and a R-node (c).

2.3.2 SP-trees

Partial 2-trees are the class of graphs which are primarily considered in this thesis. Due to the structure of partial 2-trees, each SPQR-tree of a 2-connected partial 2-tree only consist of S- and P-nodes, as proven in Lemma 2.3.1. Therefore, the SPQR-tree of a partial 2-tree G is called *SP-tree* and denoted by SP(G).

[8] An embedding is unique when the direction is ignored as shown in Theorem 4.0.3

Lemma 2.3.1. *Let G be a 2-connected partial 2-tree and $T = SPQR(G)$, then each node in T is either a S-node or a P-node.*

Proof. As described above, Q-nodes are ignored as they can be represented by the other types of nodes. Therefore, it is sufficient to show that there are no R-nodes in a SPQR-tree of a 2-connected partial 2-tree. As a 2-connected partial 2-tree is a series-parallel graph which can only be constructed by S- and P-operations in particular, it can be constructed by a finite sequence $C = \{c_1, c_2, \ldots, c_k\}$ of S- and P-operations. Without loss of generality it is assumed that the operations are ordered in a way that there is a family of sets O such that each set in the family contains a disjunct subsequence $C_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_n}\}$ with $C = \bigcup_{i=1}^{l} C_i$ consists of only S- or P-operations. As each S-operation leads to a path and each P-operation to a cycle, it can be shown by induction regarding the length of the sequence that T only consists of S- and P-nodes. □

SP-trees and normalized tree decompositions are strongly related representations of a graph as both can be obtained from each other. But unlike a normalized tree decomposition a SP-tree of a graph is unique, as it is a SPQR-tree without any R-nodes. A SP-tree can be obtained from a normalized tree decomposition by merging each separator node $s \in S$ which is adjacent to exactly two clique nodes and also is associated with a bag X_s of vertices that are non-adjacent in G with its neighbors. After merging all these nodes, the separator nodes can be transformed to P-nodes and the clique nodes can by transformed to S-nodes. The skeleton graphs of the P-nodes contain the vertices which are in the bags associated with the separator nodes whereas the skeleton graphs of the S-nodes contain all vertices which are in the bags associated with the clique nodes and all edges between these vertices which are in the graph itself. In addition to these edges both types of nodes contain virtual edges, which are generated as described above.

A normalized tree decomposition can be obtained from a SP-tree by reversing the merging process. To do so, each S-node with a skeleton graph containing more than three real vertices has to be split into two S-nodes with and P-node pertinent to both. Then, if there is no node with a skeleton graph that contains more than three edges, each S-node represents a clique node whereas each P-node represents a separator node. The bags associated with these nodes contain the vertices which are contained in the skeleton graphs of the S- and P-nodes. A clique node and a separator are adjacent if and only if the corresponding S- and P-node are adjacent in the SP-tree.

The algorithm presented in chapter 4 is initially presented in [20] where a notation for SP-trees is used which is introduced in [10]. The same notation is used in this thesis.

2.3.2 Definition (SP-tree). Let G be a 2-connected partial 2-tree with at least three vertices. Then the SP-tree of G denoted by $\mathrm{SP}(G) = T$ is the smallest tree such that the following conditions are satisfied:

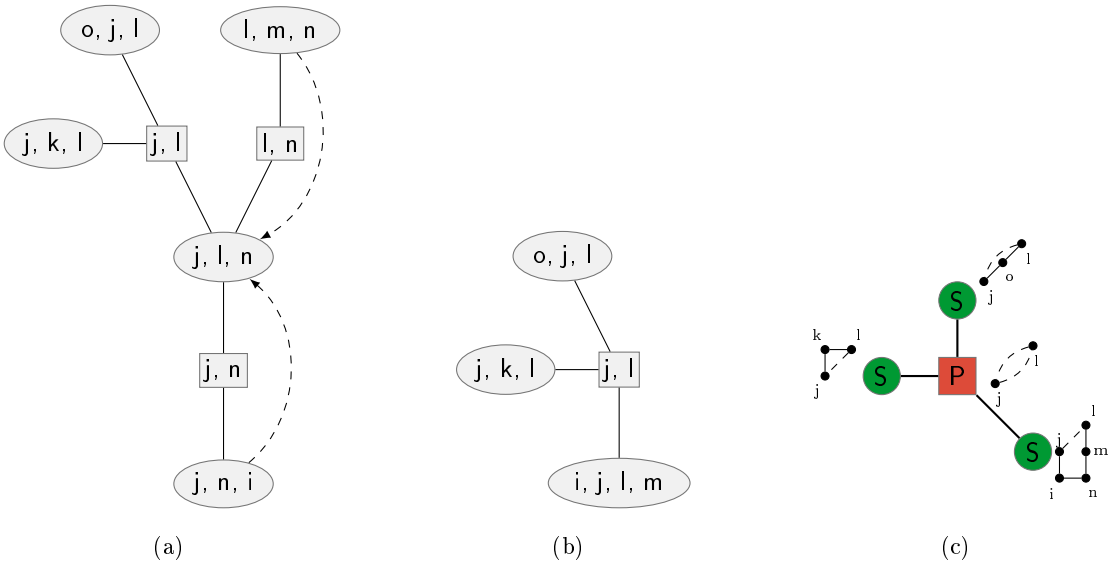


Figure 2.11: Normalized tree decomposition shown in Figure 2.7 with arrows pointing to the edges being merged (a). The resulting tree (b) and the corresponding SP-tree (c).

SP1 each node λ of T is associated with a skeleton graph $S_\lambda = (V_\lambda, E_\lambda)$. Each edge $e = (u, v) \in E_\lambda$ is either a real or a virtual edge. If e is a virtual edge, then $S = \{u, v\}$ is a separator of G.

SP2 T has two different types of nodes. S-nodes where the skeleton graph is a simple cycle and P-nodes which have a skeleton graph consisting of multiple parallel edges between exactly two vertices. If one of the edges is real, the P-node is called open, closed otherwise.

SP3 for two adjacent nodes λ and η in T, the skeleton graph S_λ contains a virtual edge e_η representing S_η and vice versa. The edge e_η is pertinent to the node η and analogous for λ.

SP4 The graph resulting by merging all skeleton graphs in a way that each virtual edge is replaced by its pertinent node in T is exactly G.

The set of S-nodes and P-nodes in T are denoted by $V_S(T)$ or $V_P(T)$ and T is bipartite regrading these two sets of nodes. Let T be the SP-tree of G and $r \in E(G)$. The *rooted SP-tree* is the T rooted at the S-node λ with $r \in V(S(\lambda))$. A rooted tree induces a parent-child relation where a node λ is parent an adjacent η node if the path from the root node to λ is shorter than the path from the root node to η. If a node λ is parent of a note η and $e_\lambda \in E(S(\eta))$ is the virtual edge pertinent to λ in η, then e_λ is called *reference* of λ and denoted by $\mathrm{ref}(\lambda)$.

2.3.3 Extended BC-trees

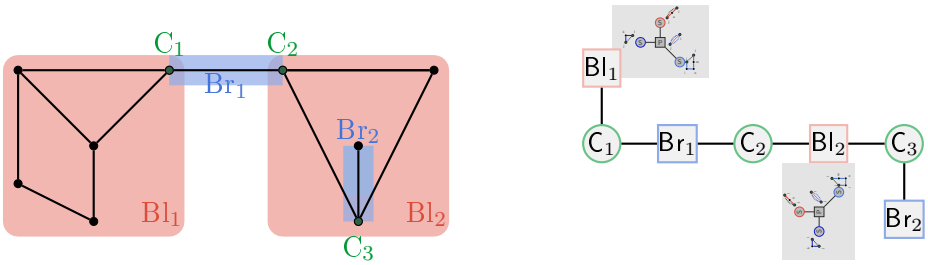
SP-trees can only decompose 2-connected partial 2-trees. As arbitrary partial 2-trees are considered later in this thesis, there is need for a graph decomposition which is capable of decomposing those graphs, too. As even arbitrary partial 2-trees are at most 2-connected, the BC-tree of a partial 2-tree consists of blocks which are either 2-connected partial 2-tree or bridges and cutvertices. In this subsection the BC-tree presented in subsection 2.2.3 is extended.

For the blocks, there are two different cases regarding the subgraph which is associated with the block. If the block contains a maximal 2-connected subgraph, then the node will be associated with a SP-tree of this 2-connected subgraph. If the block is a bridge, then it will be associated with a skeleton graph containing exactly the bridge. During the further thesis whenever a BC-tree is mentioned, the extended BC-tree is meant. Let G be a partial 2-tree and $T = BC(G)$. The nodes of G have one of three types:

Bl contains a non-bridge block which is associated with the SP-tree of the 2-connected partial 2-tree represented by this block. The set of all BL-nodes is denoted by $\mathrm{Bl}(T)$.

Br contains a bridge. Each bridge contains a skeleton graph which contains exactly the bridge. The set of all Br-nodes is denoted by $\mathrm{Br}(T)$.

C contains a cutvertex. A cutvertex represents exactly one vertex. The set of all C-nodes is denoted by $\mathrm{C}(T)$.



(a) (b)

Figure 2.12: A partial 2-tree (a) and its extended BC-tree without the skeleton graphs of Br-nodes and the SP-trees associated with the Bl-nodes highlighted with a gray background (b).

BC-trees are bipartite regarding blocks and cutvertices but not regarding Bl-nodes and Br-nodes. A BC-tree of a partial 2-tree contains all information about the graph itself as each 2-connected component of G is represented as SP-tree. Therefore, each BC-tree is unique since each 2-connected component of G is represented by its SP-tree, which is unique and these components are connected by simple paths.

2.4 Complexity

One part of complexity theory is to classify problems by their severity. To do so, different measures of complexity have been introduced. In this section a short introduction of complexity theory is given mainly focusing on the measure of time. The definitions and notations are based on [14].

A *problem* is a general question for which a solution must be found. Usually several *parameters* are left unspecified. The problem is described by first giving a general description of all its parameters and second a statement of what properties the *solution* is required to satisfy. An *instance* of a problem is obtained by specifying particular values for all the parameter, e.g. the maximum independent set problem, which is given for arbitrary graphs and the properties the is solution required to satisfy are: Any two vertices in the solution are adjacent if and only if the corresponding vertices in the input graphs are adjacent. An instance of this problem would be two graphs. Given a problem, an *algorithm* for the problem is a step-by-step procedure for solving the problem, thus given an instance for the problem returning a solution satisfying all requirements given by the problem.

Let I be all instances of a problem $\mathcal{P}$ with an algorithm $\mathcal{A}$ for $\mathcal{P}$. For each $i \in I$, $|i|$ denotes the size of the instance given in a reasonable encoding, e.g. integers can be represented in binary, thus all $n \in \mathbb{N}$ require size $\log_2(n)$ to be represented. Usually, a unary representation in which all $n \in \mathbb{N}$ require size n to be represented is not reasonable. An algorithm is called *deterministic* if each next step is well defined and *nondeterministic* if there are two options in each step but no rule which option is taken next.

The complexity class **P** contains all problems $\mathcal{P}$ for which a deterministic algorithm $\mathcal{A}$ exists such that there is a polynomial p and the running time of $\mathcal{A}$ is bounded by $p(|i|)$ for all $i \in I$. All problems $\mathcal{P}$ for which a nondeterministic algorithm $\mathcal{A}$ exists such that there is a polynomial p and the running time of $\mathcal{A}$ is bounded by $p(|i|)$ for all $i \in I$ are in the complexity class **NP**. Therefore all problems in **P** are also in **NP** and thus $\mathbf{P} \subseteq \mathbf{NP}$ whereas it is an open problem whether $\mathbf{P} \subsetneq \mathbf{NP}$ or not. If a problem is in **P**, it is considered to be efficiently solvable, but if it is in **NP**, it is not considered to be solvable efficiently. A more visual explanation of these two classes is, that **P** contains all problems which are solvable in polynomial time whereas **NP** contains all problems for whom it is possible in polynomial time to verify whether a solution is feasible or not.

A *polynomial-time reduction* from a problem $\mathcal{P}$ to a problem $\mathcal{P}'$ is an algorithm $\mathcal{A}$ that solves $\mathcal{P}$ by having access to an algorithm $\mathcal{A}'$ for solving $\mathcal{P}'$ such that if $\mathcal{A}'$ solves $\mathcal{P}'$ in polynomial time, then $\mathcal{A}$ solves $\mathcal{P}$ in polynomial time If such an algorithm $\mathcal{A}$ exists, then $\mathcal{P}$ is polynomial time reducible from $\mathcal{P}'$. A problem $\mathcal{A}$ is called ***NP-hard*** if every problem in **NP** is polynomial time reducible to $\mathcal{A}$. If $\mathcal{A}$ is also in **NP**, then $\mathcal{A}$ is called ***NP-complete***. All **NP**-complete problems are in the complexity class **NPC**. As each problem in **NP** can be reduced to a **NP**-complete problem, it is sufficient to show that a **NP**-complete problem

can be reduced to a problem $\mathcal{P}$ to proof that $\mathcal{P}$ is **NP**-hard. If there is a deterministic polynomial time algorithm for any problem in **NPC**, then **P** = **NP**.

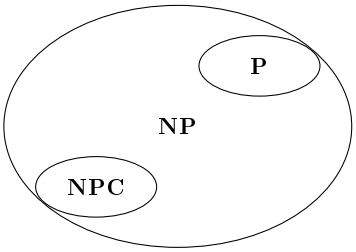


Figure 2.13: Relation of **P** and **NP**

As written above, the input must be presented in a reasonable encoding therefore, problems containing numbers cannot easily be reduced to problems on other data structures as graphs. For instance the *Knapsack* problem, [14, MP9]: Given a finite set U of elements, a size $s\colon U \to \mathbb{Z}^+$ and a value $v\colon U \to \mathbb{Z}^+$ for each element in U and two positive integers B and K. Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$? If there is an algorithm with running time polynomial in a values of $s(u), v(u), B$ or K for any $u \in U$ it is not a polynomial time algorithm for the Knapsack problem. It is only polynomial in the values of the instance but not in its length as a reasonable encoding of integers only requires logarithmic length. Hence, such an algorithm would be a *pseudo-polynomial time* algorithm.

The concept of *strong* **NP**-completeness is introduced in [13]. A problem is called **NP**-complete in the strong sense if it remains **NP**-complete or -hard respectively if the numbers are bounded by a polynomial. An equivalent and more vivid definition is that the problem remains **NP**-complete or -hard respectively even if the numbers in an instance of the problem are encoded unary. So if a problem is **NP**-complete in the strong sense, the values of the numbers can be used in the reduction without loosing the polynomial part of the reduction.

A parameterized problem is a problem for which one or more parameters are fixed so each instance is extended to a set of tuples (k, i) with $k \in \mathbb{N}$ and $i \in I$. One example for a parameterized problem is the tree width problem. The question whether a graph G has tree width k would be represented by the tuple (k, G). A problem is *fixed parameter tractable* if there is an algorithm which solves the problem with input (k, i) and running time $f(k)p(i)$ where f is a recursive function and p a polynomial. The set of all fixed parameter tractable problems is called **FPT**. As stated earlier, the tree width problem is not solvable in polynomial time but as the parametrized version of the problem is solvable in polynomial time regarding the parameter the problem is in **FPT**. Problems in **FPT** are also considered to be efficient solvable since there exists an algorithm with polynomial running time for the parametrized version of the problem.

Chapter 3

The Maximum Common Subgraph Problem for Partial 2-trees

The maximum common subgraph problem is an optimization problem. For an optimization problem, there is a set I of instances, a function F which maps each instance $i \in I$ to a set $F(i)$ of feasible solutions, a *weighting function* w which maps an instance $i \in I$ and a feasible solution $f \in F(i)$ of the instance to a value and a *goal function* which is either *minimize* or *maximize*. The solution for the optimization problem is the minimum or maximum feasible solution of the instance regarding the goal and weighting function.

3.0.1 Definition (Maximum common subgraph problem).

Input: Two graphs G and H.

Output: The size of the maximum common subgraph isomorphism between G and H.

Regarding the maximum common subgraph problem the instances are pairs of graphs, the feasible solutions are graph isomorphisms between the vertices of the graphs. The weighting function maps each of the subgraph isomorphisms to an integer equal to the number of vertices in the codomain of the isomorphism. Last, the goal function of the problem is maximize because the size of the biggest subgraph isomorphism needs to be determined.

There is also a decision version of the maximum common subgraph problem which has an an additional integer $k \in \mathbb{N}$ as input and the output is the answer to the question whether there is a maximum common subgraph of size greater or equal to k.

The maximum common subgraph problem is **NP**-hard for arbitrary graphs while the decision version of the problem is **NP**-complete [14, GT48]. Thus, the only known polynomial time algorithm are solving restricted versions of the problem, where either the graph class of the input graphs G and H or properties of the maximum common subgraphs are

restricted. In some cases both the input graphs and the maximum common subgraphs are restricted.

Let $\mathcal{C}$ be a graph class. Then the *maximum common subgraph problem for* $\mathcal{C}$ refers to the maximum common subgraph problem where both input graphs $G, H \in \mathcal{C}$. In addition to a special graph class, the graphs in this class can be restricted by a property, e.g. the maximum degree of the vertices or the number of occurrences of a specific structure. Sometimes even the restriction of the graph class of the input graphs is not enough. Let $\mathcal{C}$ and $\mathcal{D}$ be graph classes, then the *maximum common* $\mathcal{D}$ *subgraph problem for* $\mathcal{C}$ refers to the maximum common subgraph problem where for both input graphs $G, H \in \mathcal{C}$ and in addition a common subgraph $G^\star$ is a feasible solution if and only if $G^\star \in \mathcal{D}$.

In this chapter the maximum common subgraph problem is classified regarding its time complexity. It is shown that it is **NP**-hard and **NP**-complete, respectively, if the decision version is considered, even if both input graphs are restricted. It is known that the problem is **NP**-hard for partial k-trees even if all but k vertices have bounded degree by $k+2$ [16]. It is proven, that this result can be enhanced for partial 2-trees. The problem is **NP**-hard and **NP**-complete for the decision version even if all but one vertex have bounded degree. The final result is presented in Theorem 3.3.2.

The proof of Theorem 3.3.2 is split into three parts. First in Section 3.1 it is shown that the decision version of the maximum common subgraph problem is in **NP** even if the input graphs are restricted to partial 2-trees and the *numerical matching with target sums* problem is introduced.

Then in Section 3.2 two graphs are constructed with polynomial size regarding an instance of the numerical matching with target sums problem. These graphs are 2-connected and have degree bounded by three for all but one vertex. Also special characteristics of these graphs regarding their maximum common subgraphs are proved, which then are used in the third part of the proof.

Last, it is shown in Section 3.3 that these two graphs can be used for a polynomial-time reduction from the numerical matching with target sums problem to the maximum common subgraph problem and its decision version. All mentions of a polynomial-time reduction in this chapter are referring to this polynomial-time reduction from the numerical matching with target sums problem.

3.1 The Complexity of the Decision Version of the Maximum Common Subgraph Problem and the Numerical Matching with Target Sums Problem

It is easy to see, that the decision version of the maximum common subgraph problem for partial 2-trees is in **NP**, since a common subgraph of size k can simply be guessed and then

be verified in polynomial time. It is not possible to verify that there is no greater common subgraph for any guessed subgraph, therefore the optimization version of the problem is not known to be in **NP**.

Lemma 3.1.1. *The decision version of the maximum common subgraph problem for partial 2-trees is in **NP**.*

Proof. There is a nondeterministic algorithm computing the maximum common subgraph of two partial 2-trees, as the additional information for the algorithm can be a mapping of vertices from one input graph to the other. The size of this mapping is bounded by the input graphs. Therefore it is possible to first test in polynomial time whether the mapping is a subgraph isomorphism and second verify whether the mapping maps at least k vertices. Thus, it is possible to verify in polynomial time whether the additional information represents a common subgraph of size at least k. Therefore the maximum common subgraph problem for partial 2-trees is in **NP**. □

To proof later in Section 3.3 that the decision version of the maximum common subgraph problem for partial 2-trees is **NP**-complete and the optimization version is **NP**-hard, respectively it is be shown that they both can be reduced from the following problem:

3.1.2 Definition (Numerical matching with target sums problem). Given two disjoint sets X and Y with $|X| = |Y| = n$, size $s\colon X \cup Y \to \mathbb{Z}^+$ and a vector $\vec{b} = \langle b_1, b_2, \ldots, b_n \rangle$ with $b_i \in \mathbb{Z}^+$ for all $i = 1, 2, \ldots, n$. Can $X \cup Y$ be partitioned into disjoint sets $A_1, A_2, \ldots, A_n$ each containing one element from each of X and Y, such that $\sum_{a \in A_i} s(a) = b_i$ for all $i = 1, 2, \ldots, n$, [14, SP17].

The numerical matching with target sums problem is **NP**-complete in the strong sense[1], therefore, the numerical values can be used in the polynomial-time reduction without causing a pseudo-polynomial time algorithm. While it is possible to reduce numerical matching with target sums problem to the maximum common subgraph problem, it is not clear whether the maximum common subgraph problem for partial 2-trees is in **NP**, therefore, the polynomial-time reduction only proofs that it is **NP**-hard.

3.2 The Graphs for a Polynomial-Time Reduction

In this section, the two graphs which later will be used in the polynomial-time reduction from the numerical matching with target sums problem to the maximum common subgraph problem are constructed. Therefore, it is important that they can be computed in polynomial time for each instance. Let $(X, Y, s, \vec{b})$ be an instance of the numerical matching with target sums problem and $n = |\vec{b}|$. For a better overview the following notation

[1] Reduction from the 3-dimensional numeric matching problem [14].

is used: $\Sigma_s := \sum_{i=1}^{n} (s(x_i) + s(y_i))$, $\Sigma_{\vec{b}} := \sum_{i=1}^{n} b_i$, $V_G = V(G_s^{X,Y})$, $E_G = E(G_s^{X,Y})$, $V_H = V(H_{s,r}^{X,Y}), E_H = E(H_{s,r}^{X,Y})$ and $[a] := \{1, 2, \ldots, a\}$ $\forall a \in \mathbb{Z}^+$.

3.2.1 The Graph $G_s^{X,Y}$

Graph $G_s^{X,Y}$ represents the values of the elements in sets X and Y. The graph consist of a base gadget, i.e. Figure 3.1. A similar base gadget can be found in graph $H_{s,\vec{b}}^{X,Y}$[2].

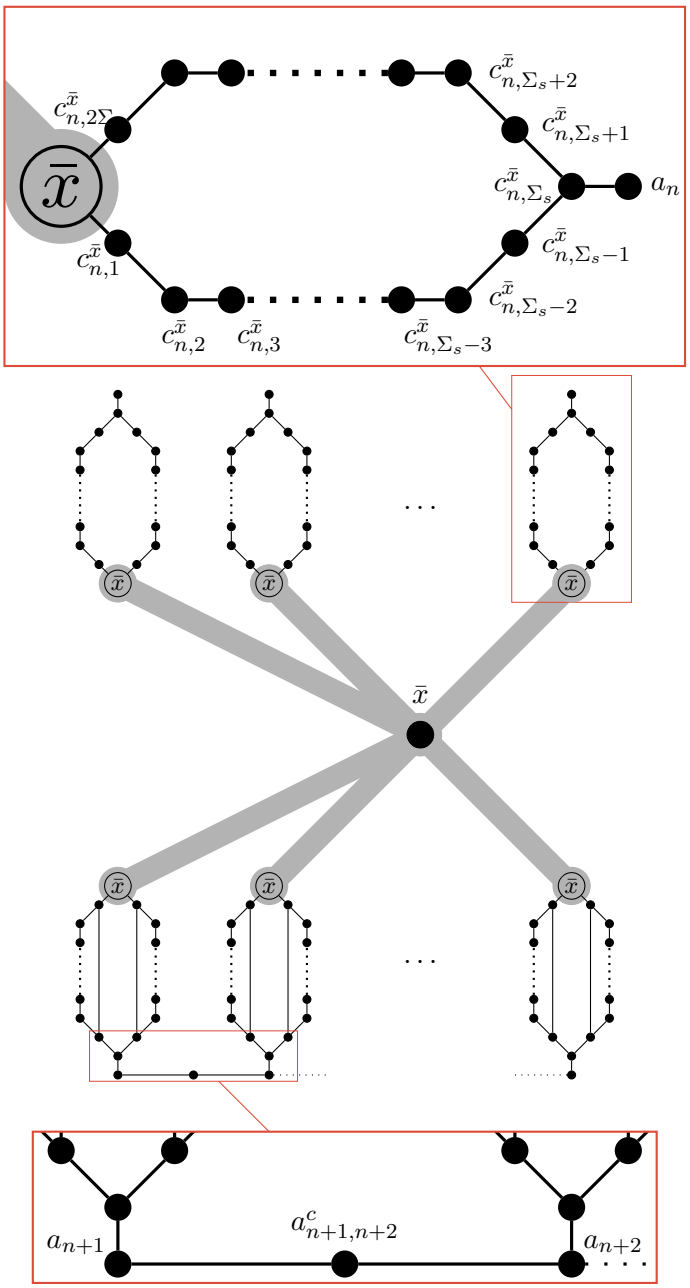


Figure 3.1: Base gadget of $G_s^{X,Y}$, all vertices labeled $\bar{x}$ are representing the same vertex.

[2]See Figure 3.4.

These gadgets are used to guarantee that each maximum common subgraph of the graphs is a feasible solution of the numerical matching with target sums problem later in the polynomial-time reduction. The base gadget of $G_s^{X,Y}$ consists of one vertex $\bar{x}$ which is the vertex with unbounded degree in $G_s^{X,Y}$. There are also $2n$ cycles which are vertex disjoint except for the vertex $\bar{x}$. Each of these cycles consists of $2\Sigma_s + 1$ vertices. Let $C_i := \left\{c_{i,j}^{\bar{x}} : j \in [2 \cdot \Sigma_s^{X,Y}]\right\} \cup \{\bar{x}\} \;\; \forall i \in [2n]$ denote the set containing all vertices in the i-th cycle.

These cycles are used to distinguish between elements in set X and set Y of the numerical matching with target sums instance. To do so, n of these cycles are chordless while n cycles have exactly 2 chords. Without loss of generality it is assumed that in the base gadget of $G_s^{X,Y}$ the first n cycles are chordless. In addition to the vertex $\bar{x}$ and the cycles, the baste gadget of $G_s^{X,Y}$ consists of $2n$ *anchor* vertices a_i for all $i \in [2n]$.

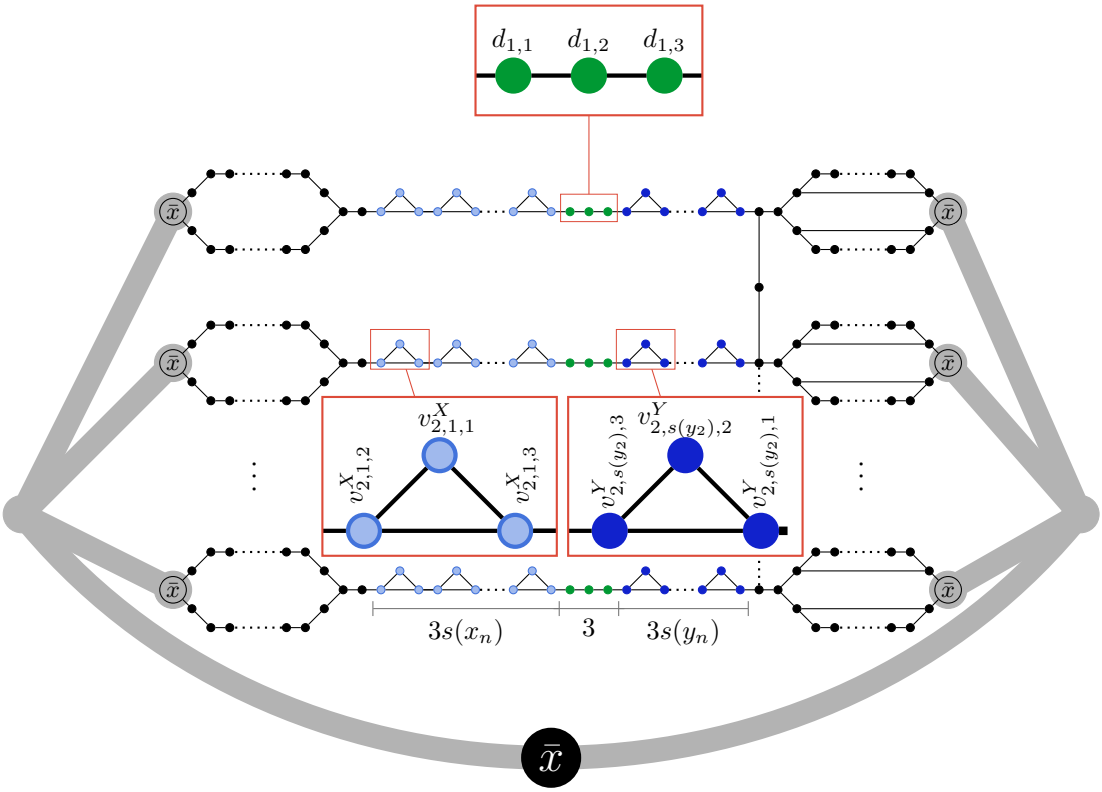


Figure 3.2: Graph $G_s^{X,Y}$ with its vertices contained in the base in black, vertices in the separating paths in green and the vertices representing the values of the elements in X and Y in light and dark blue, respectively.

The gadgets encoding the values of the elements in X and Y are later connected to these vertices. In addition to the anchor vertices, there are $n-1$ vertices $a_{i,i+1}^c$ which are contained in a path $(a_{n+1}, a_{n+1,n+2}^c, a_{n+2}, \ldots, a_{2n-1,2n}^c, a_{2n})$. This path is required as otherwise $G_s^{X,Y}$ would not be 2-connected. Since the 2-connectivity of the graph is an

important characteristic, these vertices and the path are necessary. Later it is shown that no vertex $a^c_{n+i,n+i+1}$ is contained in any maximum common subgraph of $G_s^{X,Y}$ and $H^{X,Y}_{s,\vec{b}}$ for all $i \in [n]$. Therefore, there is no 2-connected maximum common subgraph.

For each element $z \in X \cup Y$ the value of the element is represented by $s(z)$ connected K_3's. Thus, for each of the elements in X and Y there is a set of vertices $S_i^X := \left\{v^X_{i,j,k} \colon j \in [s(x_i)], k \in [3]\right\}$ and $S_i^Y := \left\{v^Y_{i,j,k} \colon j \in [s(y_i)], k \in [3]\right\}$, respectively. These vertices are connected by edges defined in Line 3.5, 3.6 and 3.8. The vertices in S_i^X, S_i^Y and the incident edges are a subgraphs of $G_s^{X,Y}$ representing the value of the associated x_i and y_i for all $i \in [n]$, respectively. Each of these subgraphs is connected with one of the cycles of the base with respect to the set they are associated with. Therefore there is an edge $(c^{\bar{x}}_{i,\Sigma_s^{X,Y}}, v^X_{j,1,1})$ and $(c^{\bar{x}}_{n+i,\Sigma_s^{X,Y}}, v^Y_{j,1,1})$ for each of these subgraphs, Line 3.9. To obtain a 2-connected graph, two subgraphs induced by S_i^X and S_i^Y representing values from different sets are connected by a *separating path* for each $i \in [n]$. These paths contain three vertices d_i^1, d_i^2 and d_i^3 for each $i \in [n]$. In addition of ensuring a 2-connected graph, these paths are later used in the reduction as they indicate whether there is a numerical matching with target sums, or not. In total $G_s^{X,Y}$ is defined as followed:

$$V_G = \bigcup_{i=1}^{n} \left(\left\{d_i^1, d_i^2, d_i^3\right\} \cup \left\{a^c_{i+n,i+n+1}\right\}\right) \cup \bigcup_{i=1}^{2n} \left(\bigcup_{j=1}^{2\cdot\Sigma_s} \left\{c^{\bar{x}}_{i,j}\right\} \cup \{a_i\} \right) \cup \{\bar{x}\} \tag{3.1}$$

$$\cup \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{s(x_i)} \bigcup_{k=1}^{3} \left\{v^X_{i,j,k}\right\} \cup \bigcup_{j=1}^{s(y_j)} \bigcup_{k=1}^{3} \left\{v^Y_{i,j,k}\right\} \right) \tag{3.2}$$

$$E_G = \bigcup_{i=1}^{2n} \left\{(\bar{x}, c^{\bar{x}}_{i,1}), (c^{\bar{x}}_{i,2\cdot\Sigma_s}, \bar{x}), (x^{\bar{x}}_{i,\Sigma_s}, a_i)\right\} \cup \bigcup_{i=n+1}^{2n-1} \left\{(a_i, a^c_{i,i+1}), (a^c_{i,i+1}, a_{i+1})\right\} \tag{3.3}$$

$$\cup \bigcup_{i=1}^{2n} \bigcup_{j=1}^{2\cdot\Sigma_s-1} \left\{(c^{\bar{x}}_{i,j}, c^{\bar{x}}_{i,j+1})\right\} \cup \bigcup_{i=n+1}^{2n} \left\{(c^{\bar{x}}_{i,1}, c^{\bar{x}}_{i,\Sigma_s-1}), (c^{\bar{x}}_{i,\Sigma_s+1}, c^{\bar{x}}_{i,2\Sigma_s-1})\right\} \tag{3.4}$$

$$\cup \bigcup_{i=1}^{n} \bigcup_{j=1}^{s(x_i)} \left\{(v^X_{i,j,1}, v^X_{i,j,3}), (v^X_{i,j,1}, v^X_{i,j,2})(v^X_{i,j,2}, v^X_{i,j,3})\right\} \tag{3.5}$$

$$\cup \bigcup_{i=1}^{n} \bigcup_{j=1}^{s(y_i)} \left\{(v^Y_{i,j,1}, v^Y_{i,j,3}), (v^Y_{i,j,1}, v^Y_{i,j,2})(v^Y_{i,j,2}, v^Y_{i,j,3})\right\} \tag{3.6}$$

$$\cup \bigcup_{i=1}^{n} \left\{(v^X_{i,s(x_i),3}, d_i^1), (d_i^1, d_i^2), (d_i^2, d_i^3), (d_i^3, v^Y_{i,s(y_i),3})\right\} \tag{3.7}$$

$$\cup \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{s(x_i)-1} \left\{(v^X_{i,j,3}, v^X_{i,j+1,1})\right\} \cup \bigcup_{j=1}^{s(y_i)-1} \left\{(v^Y_{i,j,3}, v^Y_{i,j+1,1})\right\} \right) \tag{3.8}$$

$$\cup \bigcup_{i=1}^{n} \left(\left\{(a_i, v^X_{i,1,1}), (v^Y_{i,1,1}, a_{n+i})\right\}\right) \tag{3.9}$$

Lemma 3.2.1. *$G_s^{X,Y}$ can be computed in linear time for each instance of the numerical matching with target sums problem and is a partial 2-tree.*

Proof. First it is shown that $G_s^{X,Y}$ has a linear size with respect to an instance of the numerical matching with target sums problem and can therefore be computed in linear time.

Line in definition	Short description	Number of vertices or edges
3.1	Vertices used in the base gadget of $G_s^{X,Y}$ and the separating paths.	$4n + 8n\Sigma_s + 1$
3.2	Vertices used to represent the values of the elements in X and Y.	$3\Sigma_s$
3.3	Edges incident to $\bar{x}$ and the anchor vertices.	$10n$
3.4	Edges contained in the cycles of the base gadget.	$4n\Sigma_s + 2n$
3.5	Edges in the K_3 representing the values in X.	$3\sum_{i=1}^{n} s(x_i)$
3.6	Edges in the K_3 representing the values in Y.	$3\sum_{i=1}^{n} s(y_i)$
3.7	Edges connecting the K_3's and the separating paths.	$4n$
3.8	Edges connecting the K_3's.	$\Sigma_s - 2n$
3.9	Edges connecting the K_3's and the anchor vertices.	$2n$

Table 3.1: Number of vertices and edges in $G_s^{X,Y}$ with respect to an instance of the numerical matching with target sums problem.

As $|V_G| = 4n+8n\Sigma_s+3\Sigma_s+1$ and $|E_G| = 16n+4n\Sigma_s+4\Sigma_s$ and the values of instances of the numerical matching with target sums problem can be used, because the problem is **NP**-complete in the strong sense, $G_s^{X,Y}$ can be computed in linear time regarding a polynomial-time reduction. $G_s^{X,Y}$ is also a partial 2-tree, and as a graph is a partial 2-tree if each 2-connected component of the graph is a series-parallel graph and as $G_s^{X,Y}$ is 2-connected, it is sufficient to show that $G_s^{X,Y}$ is a series-parallel graph.

Since it is enough to show that $G_s^{X,Y}$ is a series-parallel graph, it is sufficient to show that it can be constructed with S- and P-operations to proof that $G_s^{X,Y}$ is a partial 2-tree. It is easy to see that a K_3 is a series-parallel graph and can therefore be used in this construction of $G_s^{X,Y}$.

By joining K_3's and K_2's with the S-operation, the subgraphs induced by S_i^X and S_i^Y can be created in a way that the vertices v_1^X and v_1^Y are denoted with s and t for all $i \in [n]$. The same can be done for the cycles of the base gadget with the corresponding anchor vertices such that the cycles with chords have vertex $c_{i,\Sigma_s}^{\bar{x}}$ denoted by s and $a_{i,i+1}^c$ denoted by t for all $i \in [2n-1]$ while the $2n$-th cycle is not connected to any vertex $a_{2n,2n+1}^c$. Joining these components with a S-operation and a P-operation results in a graph where the vertex, which is later denoted by $\bar{x}$ is denoted by s and for the i-th of these graphs, the vertex a^{i+1} is denoted with t for all $i \in [2n-1]$. These components can then simply be merged with P-operations, resulting in the graph $G_s^{X,Y}$.

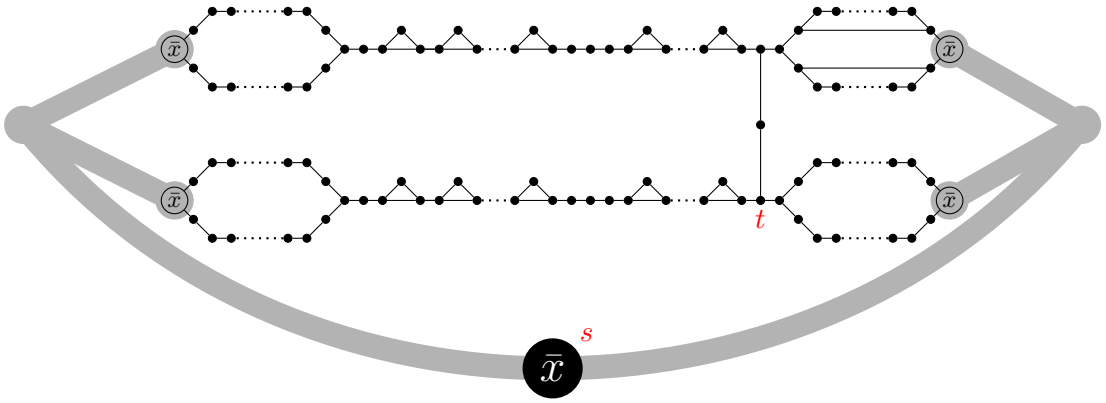
Therefore $G_s^{X,Y}$ can be generated with S- and P-operations and is a series-parallel graph. Since it is 2-connected it also is a partial 2-tree. □

The construction of $G_s^{X,Y}$ is exemplified in Figure 3.3. To create greater graphs, the result of the P-operation in Figure 3.3(a) must be joined with a path of length 2 on the vertex labeled with t. This results in a graph which can be joined with the graph in 3.3(c), resulting in a graph for which can represent the values of the elements in X and Y of an instance of the numeric matching with target sums problem containing 3 elements each. Repeating this process then results in a graph which can represent arbitrary instances.

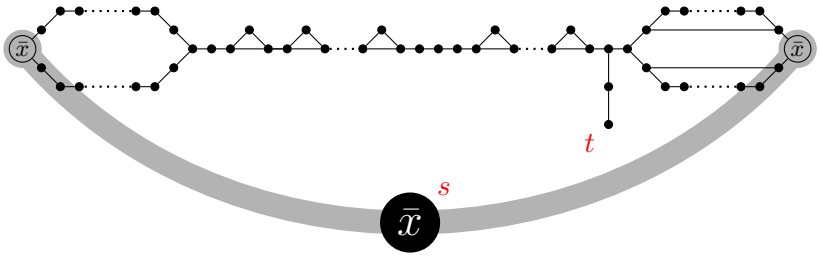
3.2.2 The Graph $H_{s,\vec{b}}^{X,Y}$

Graph $H_{s,\vec{b}}^{X,Y}$ on the other hand represents the values in vector $\vec{b}$. To guarantee that all vertices are mapped accordingly, $H_{s,\vec{b}}^{X,Y}$ contains a base gadget similar to the base of $G_s^{X,Y}$, c.f. Figure 3.4. The only difference between the base gadget of $G_s^{X,Y}$ and the base gadget of $H_{s,\vec{b}}^{X,Y}$ is, that the latter has a path between the anchor vertices of chordless cycles while the path in the base gadget of $G_s^{X,Y}$ contains the anchor vertices of the cycles containing chords.

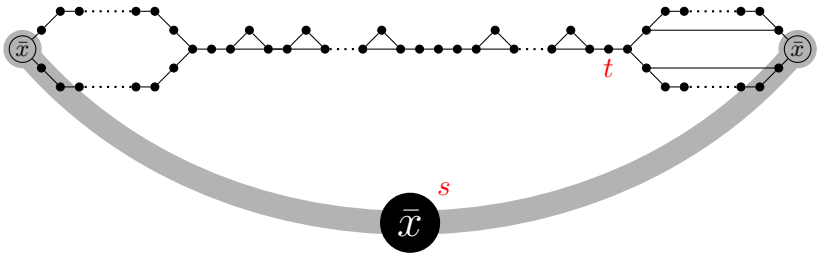
Like the values of the elements in X and Y in $G_s^{X,Y}$, the values of the vector $\vec{b}$ are represented by K_3's. For each b_i there is a set of vertices $B_i := \{v_{i,j,k} \colon j \in [b_i+1], k \in [3]\}$ for all $i \in [n]$. These subgraphs are also connected to the two cycles induced by C_i^X and C_i^Y of the base of $H_{s,\vec{b}}^{X,Y}$. The difference between $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ is, that there are only n values which need to be represented in $H_{s,\vec{b}}^{X,Y}$ in contrast to the $2n$ values in $G_s^{X,Y}$. Thus, there is no need for separating paths. To represent the value of b_i, $H_{s,\vec{b}}^{X,Y}$ contains a set of vertices B_i which contains exactly $3b_i + 3$ vertices for all $i \in [n]$. As the values are represented by K_3's, $3b_i$ vertices are needed to exactly represent the value. The additional three vertices are used for an additional K_3 in each subgraph induced by B_i. These K_3's are later used in the reduction and are corresponding to the separating paths in $G_s^{X,Y}$. These vertices are connected by edges analogously to the vertices in S_i^X resp. S_i^Y, see Line 3.14. Thus in total $H_{s,\vec{b}}^{X,Y}$ is defined as followed, also see Figure 3.5:



(a)



(b)



(c)

Figure 3.3: The graph in (a) can be constructed by joining the graphs in (b) and (c) with a P-operation.

$$V_H = \bigcup_{i=1}^{n} \left(\{a^c_{i+n,i+n+1}\} \cup \bigcup_{j=1}^{b_i+1} \bigcup_{k=1}^{3} \left\{ v^{\vec{b}}_{i,j,k} \right\} \right) \cup \bigcup_{i=1}^{2n} \left(\bigcup_{j=1}^{2\cdot\Sigma_s} \left\{ c^{\bar{y}}_{i,j} \right\} \cup \{a_i\} \right) \cup \{\bar{y}\} \tag{3.10}$$

$$E_H = \bigcup_{i=1}^{2n} \left\{ (\bar{y}, c^{\bar{y}}_{i,1}), (c^{\bar{y}}_{i,2\cdot\Sigma_s^{X,Y}}, \bar{y}), (c^{\bar{y}}_{i,\Sigma_s}, a_i) \right\} \cup \bigcup_{i=n+1}^{2n-1} \left\{ (a_i, a^c_{i,i+1}), (a^c_{i,i+1}, a_{i+1}) \right\} \tag{3.11}$$

$$\cup \bigcup_{i=1}^{2n} \bigcup_{j=1}^{2\cdot\Sigma_s^{X,Y}-1} \left\{ (c^{\bar{y}}_{i,j}, c^{\bar{y}}_{i,j+1}) \right\} \cup \bigcup_{i=1}^{n} \left\{ (c^{\bar{y}}_{i,1}, c^{\bar{y}}_{i,\Sigma_s-1}), (c^{\bar{y}}_{i,\Sigma_s+1}, c^{\bar{y}}_{i,2\Sigma_s-1}) \right\} \tag{3.12}$$

$$\cup \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{b_i+1} \left\{ (v^{\vec{b}}_{i,j,1}, v^{\vec{b}}_{i,j,3}), (v^{\vec{b}}_{i,j,1}, v^{\vec{b}}_{i,j,2})(v^{\vec{b}}_{i,j,2}, v^{\vec{b}}_{i,j,3}) \right\} \cup \bigcup_{j=1}^{b_i} \left\{ (v^{\vec{b}}_{i,j,3}, v^{\vec{b}}_{i,j+1,1}) \right\} \right) \tag{3.13}$$

$$\cup \bigcup_{i=1}^{n} \left(\left\{ (a_i, v^{\vec{b}}_{i,1,1}), (v^{\vec{b}}_{i,b_i+1,3}, a_{n+i}) \right\} \right) \tag{3.14}$$

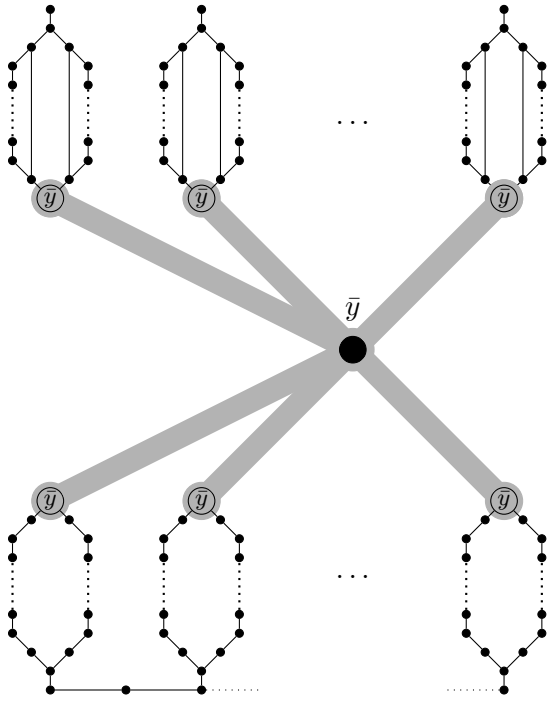


Figure 3.4: Base gadget of $H_{s,\vec{b}}^{X,Y}$, all vertices labeled $\bar{y}$ are representing the same vertex.

Lemma 3.2.2. *$H_{s,\vec{b}}^{X,Y}$ can be computed in linear time for each instance of the numerical matching with target sums problem and is a partial 2-tree.*

Proof. Similar to $G_s^{X,Y}$, the size of $H_{s,\vec{b}}^{X,Y}$ is linear to the size of the instance of the numerical matching problem with target sums. With the same argument used for $G_s^{X,Y}$ about the numerical matching with target sums problem being **NP**-complete in the strong sense, $H_{s,\vec{b}}^{X,Y}$ can be computed in linear time, as $|V_H| = n + \Sigma_{\vec{b}} + 8n\Sigma_s + 1$ and $|E_H| = 16n + 4n\Sigma_s + 4\Sigma_{\vec{b}}$, see Table 3.2 for a detailed explanation of the size.

Also, with the same argumentation used for $G_s^{X,Y}$ it is easy to see, that $H_{s,\vec{b}}^{X,Y}$ is a series-parallel graph and therefore a partial 2-tree because it is 2-connected. The construction is nearly the same except for the fact that there are no separating paths and the anchor vertices connected by the path $(a_1, a_{1,2}^c, \ldots, a_{i-1,i}^c, a_i)$ contains the vertices a_i for $i \in [n]$ and therefore the anchors of the chordless cycles. □

The similarity of the structure given by the base gadgets and the K_3's representing the values of the numbers is later used in the reduction, allowing to determine the size of a maximum common subgraph in the case, that the instance of the numerical matching with target sums problem has such a matching. The use of the additional K_3 representing the values in $\vec{b}$ is explained in Lemma 3.2.3. It allows to exclude some subgraphs of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ as common subgraph.

Line in definition	Short description	Number of vertices or edges
3.10	All vertices in the graph $H_{s,\vec{b}}^{X,Y}$.	$n + \Sigma_{\vec{b}} + 8n\Sigma_s + 1$
3.11	Edges incident to $\bar{x}$ and the anchor vertices.	$10n$
3.12	Edges contained in the cycles of the base gadget.	$4n\Sigma_s + 2n$
3.13	Edges in the K_3 representing the values of $\vec{b}$ and connecting these K_3's.	$4\Sigma_{\vec{b}} + 2n$
3.14	Edges connecting the K_3's and the anchor vertices.	$2n$

Table 3.2: Number of vertices and edges in $H_{s,\vec{b}}^{X,Y}$ with respect to an instance of the numerical matching with target sums problem.

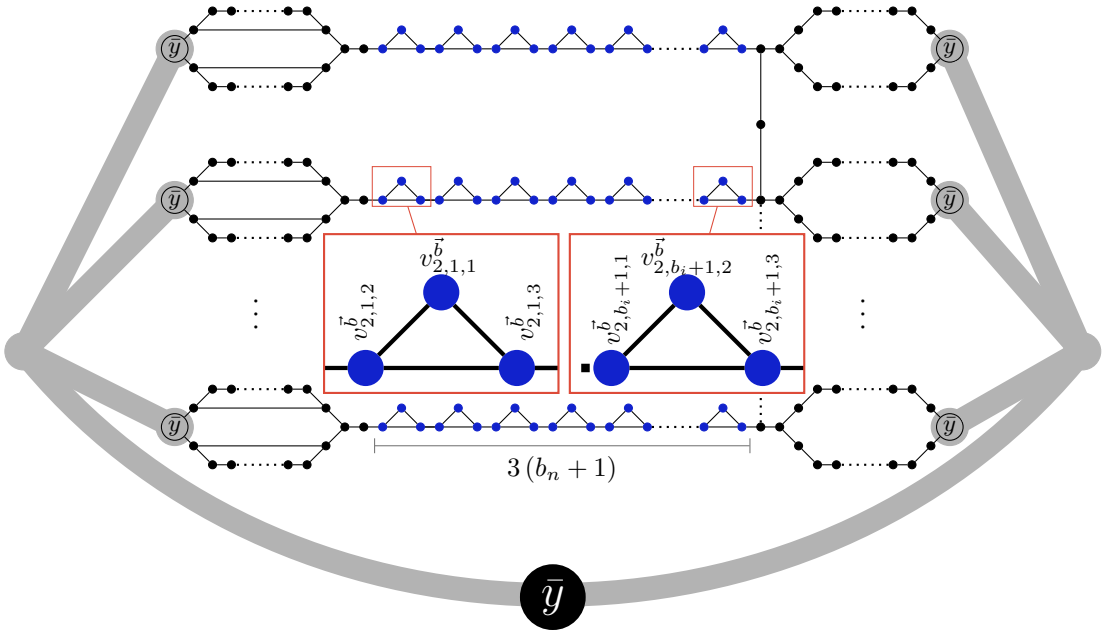


Figure 3.5: Graph $H_{s,\vec{b}}^{X,Y}$ with its vertices contained in the base in black and the vertices representing the values of the elements in $\vec{b}$ in dark blue.

3.2.3 Characteristic of a Maximum Common Subgraph of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$

Before showing that $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ can be used in a polynomial-time reduction from the numerical matching problem with target sums to the maximum common subgraph problem, either the decision or the optimization version, one characteristic of these graphs and especially their maximum common subgraphs is shown. These results require the

graphs to have a minimum size, since the chords in the base gadgets require the cycles o contain at least eight vertices. Therefore it is assumed without loss of generality that the size is sufficient. For each instance of the numerical matching with target sums problem where the requirements are not met, there is a $k \in \mathbb{N}$ such that the requirements are met if each numeric value in the instance is multiplied with k. Still, there is a matching for an instance after the multiplication if and only if there is an instance before.

Lemma 3.2.3. *Let $(X, Y, s, \vec{b})$ be an instance of the numerical matching with target sums problem and $G_s^{X,Y}$ the graph constructed for this instance. Also let S_i^X, S_i^Y be the sets containing the vertices in $G_s^{X,Y}$ used to represent $s(x_i)$ and $s(y_i)$ as defined in Section 3.2.1. The subgraph induced by S_i^X, S_i^Y and the vertices in the separating paths $D_i := \{d_i^1, d_i^2, d_i^3\}$ cannot be all in the maximum common subgraph for all $i \in [n]$.*

Proof. This is because if the vertices in S_i^X and S_i^Y are mapped to vertices in B_j then at most two vertices in D_i can be mapped to B_j due to the construction of the graphs as the three vertices in the separating paths cannot be mapped accordingly. Even if S_i^X and S_i^Y are mapped to vertices in B_j and B_k with $i, j, k \in [n]$, at least one vertex in S_i^X, S_i^Y or D_i cannot be in the common subgraph as then there would be two adjacent vertices in these sets mapped to non-adjacent vertices in B_j and B_k, see Figure 3.6.

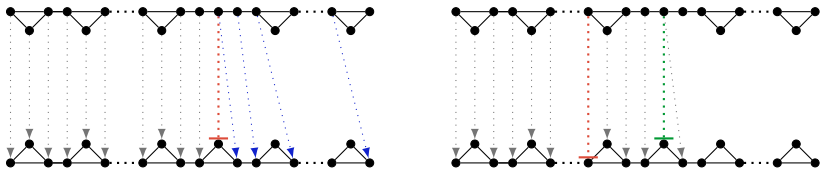
The red arrow in Figure 3.6(a) shows the vertex which cannot be mapped if both sets contain the same number of vertices and that it is therefore not possible to map all vertices of the subgraph induced by S_i^X, S_i^Y and the corresponding separating path. Because if the vertices connected by the red arrow would be mapped accordingly, the vertices connected by the blue dotted arrows would be the only feasible maximal mapping with respect to the mapping of the vertices connected by the red arrow. Since this mapping is not maximal, the common subgraph cannot be a maximum common subgraph.

In Figure 3.6(b) the red and green arrows show the problem if the sets S_i^X, S_i^Y and D_i contain less respectively more vertices than B_j. Then either at some point there is no vertex which can be mapped to a vertex in B_j or the separating path needs to be completely mapped which is not possible as described above. □

Lemma 3.2.4. *Let $M^{cs} = \{(cs_1^G, cs_1^H, \varphi_1), (cs_2^G, cs_2^H, \varphi_2), \dots (cs_k^G, cs_k^H, \varphi_k)\}$ denote the set of all common subgraphs of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ where cs_i^G is a subgraph if $G_s^{X,Y}$ isomorphic to cs_i^H which is a subgraph of $H_{s,\vec{b}}^{X,Y}$ and φ_i denotes the subgraph isomorphism for all $i \in [k]$, then*

$$|cs_i^G, | \geq |cs_j^G| \ \forall j \in [k] \Rightarrow \varphi(\bar{x}) = \bar{y} \text{ and } \varphi(c_{i,\cdot}^{\bar{x}}) = c_{j,\cdot}^{\bar{y}} \Rightarrow i \in [n], j \notin [n] \text{ or } i \notin [n], j \in [n].$$

Proof. First it is shown, that each maximum common subgraph of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ must contain $\bar{x}$ and $\bar{y}$, respectively and in addition $\bar{x}$ must be mapped to $\bar{y}$ by the subgraph isomorphism. Then it is shown, that the vertices $c_{i,1}^{\bar{x}}$ and $c_{j,1}^{\bar{y}}$ are mapped as defined above.



(a) $|S_i^X \cup D_i \cup S_i^Y| = |B_j|$ (b) $|S_i^X \cup D_i \cup S_i^Y| \neq |B_j|$

Figure 3.6: Mapping vertices in a subgraph induced by vertices in S_i^X, D_i and S_i^Y to a subgraph induced by vertices in B_j such that the vertex $v_{i,1}^{\bar{x}}$ is mapped to $v_{j,1}^{\vec{b}}$ and $v_{i,1}^{\bar{y}}$ is mapped to $v_{j,b_j+1}^{\vec{b}}$ for any $i, j \in [n]$.

Assume that there are two maximum common subgraphs cs^G and cs^H and the subgraph isomorphism φ such that $\varphi(\bar{x}) \neq \bar{y}$. Without loss of generality it is assumed that $\bar{x} \in V(cs^G)$ while $\bar{y}$ may not even be contained in the maximum common subgraph. Due to the construction of both $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$, $\bar{x}$ and $\bar{y}$ are the only vertices with an unbound degree. The degree of all other vertices is bounded by three. If $\varphi(\bar{x}) \neq \bar{y}$, then $\varphi(\bar{x})$ has at most three adjacent vertices and if there is a $v \in V(cs^G)$ such that $\varphi(v) = \bar{y}$, then v has at most three adjacent vertices. Therefore, there can at most be two cycles of the base gadgets be contained in the common subgraph. Since the degree of the vertices is three, there could also be two path contained in the maximum common subgraph each starting at either $\bar{x}$ or v in $G_s^{X,Y}$. As the longest path in both $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ is smaller than the size of a cycle in the base gadget, the common subgraph cannot be a maximum common subgraph.

Now assume that the vertices $c_{i,\cdot}^{\bar{x}}$ are not mapped accordingly. Since each base gadget has chordless cycles and cycles containing chords, a mapping of vertices in a chordless cycle to vertices contained in a cycle with chords would result in a common subgraph which cannot contain the anchor vertex of the corresponding cycles. Therefore each chordless cycle mapped to a cycle containing chords results in at least one two vertices which cannot be contained in the common subgraph, the anchor vertex and the vertex connecting the anchor vertex with another anchor vertex. In Lemma 3.2.3 it is shown that not all vertices not contained in the base gadget can be contained in a common subgraph. Therefore it is not possible that the anchor vertex is contained in it. If on the other hand all chordless cycles in $G_s^{X,Y}$ are mapped to chordless cycles in $H_{s,\vec{b}}^{X,Y}$ and the same is true for the cycles containing chords, all anchor vertices can be contained in a maximum common subgraph.

Therefore, in each maximum common subgraph the vertices of the base gadgets must be mapped accordingly and especially $\varphi(\bar{x}) = \bar{y}$. □

Thus, in a maximum common subgraph of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$, the vertices $\bar{x}$ and $\bar{y}$ must be mapped. Also all vertices $c_{i,\cdot}^{\bar{x}}$ and $c_{j,\cdot}^{\bar{y}}$ have a characteristic regarding their mapping in a maximum common subgraph because half of the cycles in each base gadget contains chords

while the other does not. These characteristics can now be used in the polynomial-time reduction as they result in a common structure of each maximum common subgraph.

3.3 A Polynomial-Time Reduction from the Numerical Matching with Target Sums Problem to the Maximum Common Subgraph Problem

Last it is shown that $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ can be used in a polynomial-time reduction from the numerical matching with target sums problem to the maximum common subgraph problem. As $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ can be computed in polynomial time regarding the values of any instance of the numerical matching with target sums problem, they can be used in the reduction. Finally the following lemma leads to Theorem 3.3.2.

Lemma 3.3.1. *Any instance $(X, Y, s, \vec{b})$ has a numerical matching satisfying all target sums if and only if each maximum common subgraph of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ has size $|V(H_{s,\vec{b}}^{X,Y})|-n$ and all vertices $\bigcup_{i=1}^{n}\left(S_i^X \cup S_i^Y\right)$ are contained in the maximum common subgraph.*

Proof. Let $M^{mcs} := \left\{(mcs_1^G, mcs_1^H, \varphi_1), (mcs_2^G, mcs_2^H, \varphi_2), \ldots (mcs_k^G, mcs_k^H, \varphi_k)\right\}$ denote the set of all maximum common subgraphs of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ and for each $m \in M^{mcs}$ let $|m|$ denote the size of the common subgraphs, also without loss of generality it is is assumed that $\varphi_i(\bar{x}) = \bar{y}$, as proven in Lemma 3.2.4.

First, it is shown that if there is a numerical matching for an instance $(X, Y, s, \vec{b})$ satisfying all target sums, then the size of a maximum commons subgraph of the corresponding graphs $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ has size $|V(H_{s,\vec{b}}^{X,Y})| - n$ and all vertices S_i^X and S_i^Y are contained in the maximum common subgraph for all $i \in [n]$.

Assume that an instance $(X, Y, s, \vec{b})$ has a numerical matching satisfying all target sums and that each maximum common subgraph $m \in M^{mcs}$ of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ has size $|m| \neq |V(H_{s,\vec{b}}^{X,Y})| - n$. As the bases of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ are mapped accordingly, it is important to consider the subgraphs induced by S_i^X, S_i^Y resp. B_i for all $i \in [n]$. Let S_i^G denote the subgraph induced by S_i^X, S_i^Y and $\left\{d_i^1, d_i^2, d_i^3\right\}$ and B_i^H the subgraph induced by B_i for all $i \in [n]$. Then it must be noted that it is not possible to fully map all vertices in S_i^G to B_i^H for all $i, j \in [n]$ as shown in Lemma 3.2.3. This is due to the separating path. Therefore $|m| < |V(H_{s,\vec{b}}^{X,Y})| - n$ and thus there are two cases: Either there is at least one D_i such that two of the vertices are not contained in the maximum common subgraph or not all vertices S_i^X and S_i^Y are contained in the maximum common subgraph for all $i \in [n]$.

If there is at least one D_i such that two vertices in D_i are not contained in the maximum common subgraph for any $i \in [n]$, then there is at least one $j \in [n]$ such that all vertices in B_j are either mapped to vertices in S_k^X and S_l^Y for any $k, l \in [n]$ or not contained in the maximum common subgraph. As B_i contains $3b_i+3$ vertices and S_k^X and S_l^Y contain $3s(x_k)$

resp. $3s(y_l)$ vertices there cannot be a numerical matching as not all vertices associated with x_k or y_l are in the maximum common subgraph.

The case that not all vertices S_i^X and S_i^Y are contained in the maximum common subgraph for all $i \in [n]$ even thought there is a numerical matching satisfying all target sums is proved the same way. Without loss of generality there are some vertices in S_1^X which are not contained in the maximum common subgraph. As a maximum common subgraph containing more vertices of S_1^X is always greater than a common subgraph containing less vertices of S_1^X but all vertices in the separating path the maximum common subgraph does not contain all vertices of D_i. Thus there cannot be a numerical matching as otherwise all vertices of S_1^X would be contained in the maximum common subgraph.

Second, it is shown that for each instance $(X, Y, s, \vec{b})$, if there is a maximum common subgraph of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ with size $|V(H_{s,\vec{b}}^{X,Y})| - n$ and all vertices S_i^X and S_i^Y are contained in the maximum common subgraph for all $i \in [n]$, then there is a numerical matching for the instance satisfying all target sums.

Assume that there is no such matching, even if the maximum common subgraph of the corresponding graphs $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ has the required size and all vertices S_i^X and S_i^Y are contained in the maximum common subgraph for all $i \in [n]$. Let $m \in M^{mcs}$ be a maximum common subgraph of those graphs, therefore the bases of $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ are mapped accordingly. Therefore each subgraph induced by vertices in S_i^X and S_i^Y is contained in m as otherwise $|m| < |V(H_{s,\vec{b}}^{X,Y})| - n$. If each subgraph induced by the vertices in S_i^X and S_i^Y is contained in m, then $|m| = |V(H_{s,\vec{b}}^{X,Y})| - 3n < |V(H_{s,\vec{b}}^{X,Y})| - n$. Thus there must be exactly $2n$ vertices contained in the separating paths also be contained in m. As it is not possible for all three vertices of a separating path to be contained in m there are only one cases: $\exists i \in [n]$ exactly two of $\{d_{i,1}, d_{i,2}, d_{i,3}\}$ is in m. In this case, there must be at least one $j \in [n]$ such that that no of the vertices in $D_i := \{d_{i,1}, d_{i,2}, d_{i,3}\}$ is contained in m. Due to the construction of the graph this is not possible because this would require at least for one $i \in [n]$ all vertices in B_i are contained in m which is also not possible as described above. But if that cannot be the case, then at least one of each vertex in B_i must be contained in m. □

Thus $G_s^{X,Y}$ and $H_{s,\vec{b}}^{X,Y}$ can be used in a reduction from the numeric matching with target sums problem to the maximum common subgraph problem and its decision version. Therefore the first is **NP**-hard while the latter is **NP**-complete. As the reduction only uses 2-connected partial 2-trees with degree restricted by three for all but two vertices the restrictions have been proven, too.

Due to these restrictions, the following result concludes form Lemma 3.3.1.

Theorem 3.3.2. *The maximum common subgraph problem in partial 2-trees is **NP**-hard and the decision version **NP**-complete even if both input graphs are 2-connected and have degree bounded by three for all but one vertex.*

In [16] it has been proven, that the subgraph isomorphism problem for partial k-trees is **NP**-complete if either of the graphs is not k-connected or has more than k vertices of unbounded degree while the rest is bounded by $k+2$. The subgraph isomorphism problem can be reduced to the maximum common subgraph problem, as a graph G is subgraph isomorph to a graph H if and only if a maximum common subgraph has size of $|V(G)|$. Therefore this result is applicable for the maximum common subgraph problem. However, Theorem 3.3.2 restricts the graph classes, for which the maximum common subgraph problem is **NP**-hard, even more.

This type of reduction cannot be applied to the restricted version of the maximum common subgraph problem for partial 2-trees, where the degree of all vertices of the input graphs is bounded. As a restriction to the degree either restricts the input length of the problem which is reduced to the maximum common subgraph problem or specifies an order on the elements in which way they are compared as demonstrated in Figure 3.7. Both restrictions seem to be too tight for any **NP**-complete problem to be reduced to.

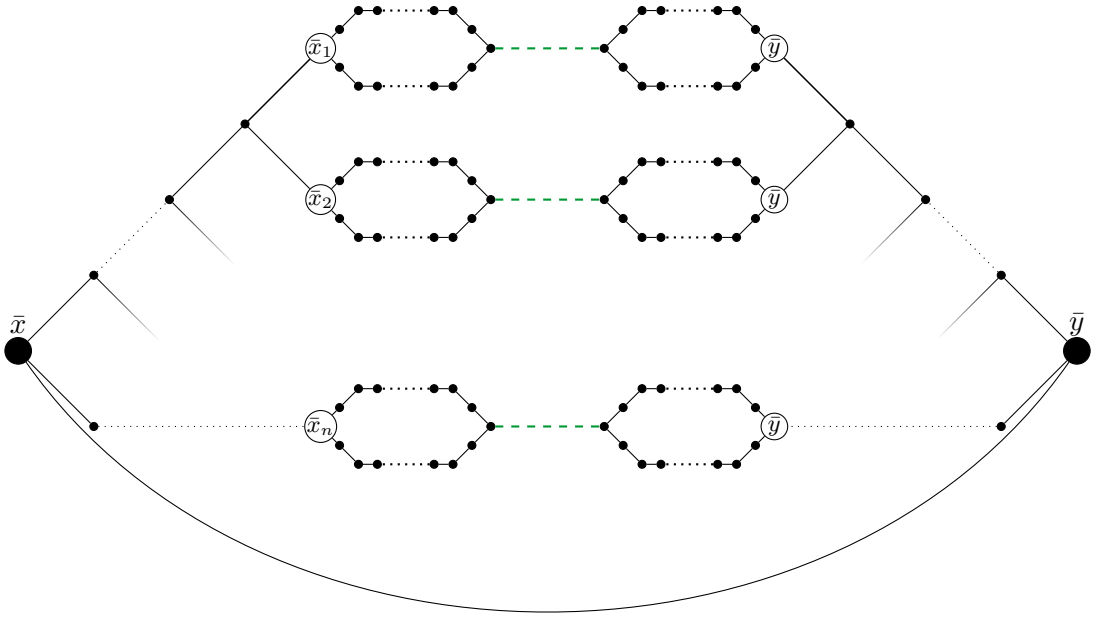


Figure 3.7: Base gadget if degree is bounded for all vertices. The green dashed edges can be replaced such that the graph is a $G_s^{X,Y}$ or a $H_{s,\bar{b}}^{X,Y}$.

If the degree is bounded for all vertices, the base has to have a tree-like structure and therefore it is determined which elements are compared with an entry of the vector. Thus it does matter which subgraphs induced by S_i^X and S_j^Y are connected with a separating path, for all $i, j \in [n]$. Thus a proof as shown above is not possible.

Chapter 4

The 2-connected Maximum Common Subgraph Problem in 2-connected Partial 2-Trees

In [4] is shown, that the maximum common subgraph problem in outerplanar graphs of bounded degree can be solved in polynomial time. Considering the hierarchy of the graph classes, partial 2-trees are a direct superclass of outerplanar graphs. Therefore each result applicable for partial 2-trees is also valid for outarplanar graphs. In this section, a restricted variant of the maximum common subgraph problem for partial 2-trees is considered: The *2-connected maximum common subgraph problem in 2-connected partial 2-trees*. This problem refers to the maximum common subgraph problem in partial 2-trees in which both a feasible maximum common subgraph and the two input graphs G and H must be 2-connected.

The algorithm presented in [4] which solves the maximum common subgraph problem for outerplanar graphs with bounded degree in polynomial time makes massive usage of Theorem 4.0.3 which is proven in [22]. But unlike outerplanar graphs, partial 2-trees do not have a unique planar embedding[1], therefore, Theorem 4.0.3 is not adaptable for all partial 2-trees and thus the presented algorithm is not applicable for partial 2-trees, even if they are 2-connected.

Theorem 4.0.3. *Let G and H be 2-connected outerplanar graphs. Let $(u_1, u_2 \ldots, u_m)$ and $(v_1, v_2, \ldots, v_n)$ be the vertices of G resp. H arranged in the clockwise order in some (outer)planar embedding[2] of G and H. If there is an isomorphic mapping $u_1 \mapsto v_{i_1}, u_2 \mapsto v_{i_2}, \ldots, u_m \mapsto v_{i_m}$ from G to a subgraph of H, $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$ appear in H in either clockwise or counterclockwise order.*

[1] Ignoring the direction.

[2] In [4] the theorem only refers to a planar embedding, but the outerplanar embedding must be considered to order all vertices clock- resp. counterclockwise.

In [20] an algorithm for the 2-connected maximum common subgraph problem in 2-connected partial 2-trees is presented. In this chapter the algorithm is explained in detail as the algorithm presented later in Chapter 5 is based on it. It should be noted, that this chapter is based on [20] since the idea is introduced there. The proofs for all lemmas and theorems can also be found there. Before presenting the algorithm some notations and theoretical background is given. Especially the idea of separators is recessed and it is shown that they can be used to define subgraphs which contain each feasible 2-connected subgraph with a special characteristic.

4.1 Ideas of the Algorithm

The algorithm for the 2-connected maximum common subgraph problem in 2-connected partial 2-trees is mainly based on two ideas. First each vertex in a 2-connected common subgraph must be contained in at least one cycle and second the problem is dividable in smaller problems, the 2-connected maximum common subgraph problem for subgraphs of the original instance, in particular. The first idea is used to identify feasible solutions while the second idea is later used to proof that the algorithm has a polynomial running time.

4.1.1 Separators

Separators are used in Section 4.1.2 to define subgraphs based on rooted SP-trees. Later they also are used to analyze the running time as the number of separators which are used to define the subgraphs is bounded polynomial by the size of the graph.

Let S be a separator. S is called *k-separator* if $|S| = k$, also S is called a *(u, v)-separator* if $G \setminus S$ does not contain any path $u \rightsquigarrow v$. A separator P is called *potential* separator of G if there is a 2-connected induced subgraph of $G' \subseteq G$ such that S is a separator of G'. Therefore, for any two partial 2-trees G and H, each separator $\{u, v\}$ in a 2-connected common subgraph $G' \subseteq G$ of these graphs, such that ϕ is the subgraph isomorphism, is a potential separator in G and $\{\phi(u), \phi(v)\}$ is a potential separator in H.

Let S be an separator of a 2-connected partial 2-tree G, then S is called *compulsive* if every normalized tree decomposition of G contains a separator node i associated with a bag X_i such that $X_i = S$. If a separator S is compulsive for a graph G but not for an induced 2-connected subgraph of G, then S is called *critical*.

Lemma 4.1.1. *Let G be a 2-connected partial 2-tree and $S = \{u, v\}$ such that $u, v \subseteq V(G)$, then the following statements are equivalent:*

(a) S is a critical separator of G,

(b) S is a compulsive separator for G and $(u, v) \notin E(G)$,

(c) there is a P-node λ in the SP-tree of G such that $V(S_\lambda) = S$ and $(u, v) \notin E(G)$,

(d) the graph $G \setminus S$ has at least three connected components and $(u, v) \notin E(G)$.

A separator S *crosses* a set of two vertices $\{u, v\}$ if S is a (u, v)-separator. Two non-crossing separators are called *parallel*. The critical and potential separators can be used to define a subgraph which contains every 2-connected subgraph separated by a potential separator. In Lemma 4.1.2 a formal definition is given, concluding in the graph G_P^S having the mentioned characteristic.

Lemma 4.1.2. *Let G be a 2-connected partial 2-tree and S a critical separator crossing a potential separator $P = \{u, v\}$. Let $C_u := \{C \in G \setminus S \colon u \in V(C)\}$ and $C_v := \{C \in G \setminus S \colon u \in V(C)\}$ be the components of $G \setminus S$ which contain u and v, respectively. Then every 2-connected induced subgraph $G' \subseteq G$ separated by P is a subgraph of the 2-connected graph $G_P^S := G[V(C_u) \cup V(C_v) \cup S]$.*

Proof. Let $S = \{s, t\}$ be a critical separator crossing a potential separator $P = (u, v)$. Also, let G' be a 2-connected induced subgraph of a partial 2-tree G such that P separates G'. In the set of components $\mathcal{C}(G' \setminus P)$ there are two components C_s and C_t containing vertex s and t, respectively. This is because the separator S crosses P in G. Therefore, all paths $s \rightsquigarrow t$ in G' must contain either u or v as otherwise S would not be crossing P. It must be noted that since $|\mathcal{C}(G \setminus S)| \geq 3$, see Lemma 4.1.1, in each of the components there must be a path $s \rightsquigarrow t$ because G is 2-connected. Also let C_u and C_v be the components containing u and v, respectively.

Assume that there are vertices $z_1, \ldots, z_n \in V(G')$ with $z_1, \ldots, z_n \notin V(C_u) \cup V(C_v)$. If there is a path $u \rightsquigarrow v$ in G containing $z_1, \ldots, z_n$, P cannot be a separator of G' since it does not contain any vertex on the path. Now assume, that there is no path $u \rightsquigarrow v$ containing vertices in either C_u or C_v. In this case G' cannot be 2-connected because there is no path between $u \rightsquigarrow v$. Since there are exactly two components containing either u or v, the vertices in these components which are also on a path $u \rightsquigarrow v$ must be preserved in any 2-connected induced subgraph of G which is separated by P.

Lemma 4.1.2 leads to the following corollary defining the maximal 2-connected induced subgraph G' of G which is separated by a potential separator of G with respect to all critical separators of G crossing the potential separator.

Corollary 4.1.3. *Let G be a 2-connected partial 2-tree and P a potential separator of G. Also let $\mathcal{S} = \{S_1, \ldots, S_l\}$ be the set of all critical separators crossing P. The graph $G_P^\star := G\left[\bigcap_{S \in \mathcal{S}} V(G_P^S)\right]$ is the maximal 2-connected subgraph of G with separator P.*

Proof. Let $G, P, \mathcal{S}$ and $G_P^\star$ be defined as above. Also let G' be a 2-connected induced subgraph of G with $|V(G)| > |V(G_P^\star)|$. Assume that G' is separated by P. Therefore, there must be a connected component in $G' \setminus P$ which is not in $G_P^\star \setminus P$. Let $\mathcal{C}_{G'}$ be the

connected components $\mathcal{C}(G' \setminus P)$ and $\mathcal{C}_{G^\star_P}$ the connected components of $G^\star_P \setminus P$. Also let $C \in \mathcal{C}_{G'}$ be a connected component such that $C \notin \mathcal{C}_{G^\star_P}$. Since the connected components in both graphs are maximal, there is no component in $\mathcal{C}_{G^\star_P}$ which shares any vertex with C.

Since $C \notin \mathcal{C}_{G^\star_P}$, there is at least one critical separator crossing P which does not result in the connected component C, therefore, if C is contained in the subgraph, P cannot be a separator of this graph. □

Therefore $G^\star_P$ is the subgraph of a 2-connected partial 2-tree G obtained by removing all components $C_i \in G \setminus S_i$ which do not contain either u or v for all critical separators S_i crossing $P = (u, v)$. This behavior is then used in the algorithm to compute well-defined subgraphs by decomposing the 2-connected partial 2-trees at their potential separators.

4.1.2 Split Graphs

To compute the 2-connected subgraphs in the algorithms for graph G, the SP-tree decomposition of G is rooted at a distinguished edge $r \in E(G)$ called the *root*. For a potential separator $P = (u, v)$ and a root $r \neq (u, v)$ the separator P *splits* G into two subgraphs G^r_{uv} and $\overline{G^r_{uv}}$ called the *split graphs*. If P is not a separator of G, which implies that $G^\star_P \neq G$, then the operation is called *shear split*. Let $\{C_1, \ldots, C_l\} = \mathcal{C}(G^\star_P \setminus P)$ and assume without loss of generality that $r \in E(C_1)$. Then $\overline{G^r_{uv}} := G[V(C_1) \cup \{u, v\}]$ and $G^r_{uv} := G\left[\bigcup_{i=2}^{l} C_l \cup \{u, v\}\right]$. The vertices u and v are referred to as *base vertices*.

Let G be a partial 2-tree and $T^{\mathrm{SP}}_G = \mathrm{SP}(G)$ the SP-tree decomposition of G. For each vertex $v \in V(G)$, $\lambda(v)$ denotes the representative of v in the skeleton graph S_λ. This is necessary as a vertex in v may be associated with more than one node in T^{SP}_G. The set of all nodes of T^{SP}_G whose skeleton graphs contain a vertex associated with v, called the set of *allocation nodes*, is defined as followed: $\Upsilon(v) := \left\{\lambda \in V(T^{\mathrm{SP}}_G) \colon u \in V(S_\lambda), \lambda(u) = v\right\}$. A *shear path* $P(u, v)$ is the shortest path $\lambda \rightsquigarrow \lambda'$ with $\lambda \in \Upsilon(u)$ and $\lambda' \in \Upsilon(v)$ in T^{SP}_G.

The following lemma explains how the critical separators can be found with a SP-tree of the graph and therefore be used in the algorithm which makes excessive use of the SP-tree decompositions.

Lemma 4.1.4. *Let $P(u, v) = (\lambda_1, \mu_1, \ldots, \mu_{l-1}, \lambda_l)$ be a shear path, then $S = \{u, v\}$ is*

- *a potential separator of G if and only if there is no P-node μ_i such that S_{μ_i} contains a real edge for all $i \in \{1, \ldots, l-1\}$,*
- *a separator of G if and only if $l = 1$.*

In the first case, T is crossed by the critical separators $V(S_{\mu_i}), i \in \{1, \ldots, l-1\}$

4.1.3 The Methods MwbMatching and Next

The algorithm uses two methods MwbMatching and Next, respectively. These methods are used to solve the *maximum weighted bipartite matching problem* and to find the next vertex which is required to be in a feasible 2-connected common subgraph.

MwbMatching solves the maximum weighted bipartite matching problem. The parameter of this method are two sets and a set defining the bipartite graph. It is assumed that there is an edge between between each two elements contained in different sets. Therefore the graph must be bipartite. The third parameter is a function assigning a value to each edge. The value is in $\mathbb{Z}^+ \cup \{-\infty\}$. The problem can be solved in $\mathcal{O}(n^3)$ by the Hungarian method [21].

The algorithm uses SP-trees as uniform data structure. Since all vertices and edges are contained in S-nodes, these nodes are mainly considered. The skeleton graph of each S-node contains a cycle. There are real and virtual edges adjacent to the vertices in the S-node. Given a vertex u and a S-node λ in the SP-tree, Next returns a vertex, which is adjacent to u, in $V(S_\lambda)$ and has yet not been included in the common subgraph, i.e. the next vertex in the cycle with respect to the direction given by the first vertex mapped in the S-node. It must be noted that the method can also return vertices which have already been considered by the algorithm.

4.2 An Algorithm for the 2-connected Maximum Common Subgraph Problem in 2-connected Partial 2-Trees

The algorithm consists of three methods. Algorithm 4.1 is the main method and only called once. For both input graphs the SP-tree decompositions are computed. Next, all S-nodes in the SP-tree T_G^{SP} of G are compared with all S-nodes in the SP-tree T_H^{SP} of H. Since all edges of G are contained in the skeleton graphs of the S-nodes, it is not necessary to compare the P-nodes of the SP-trees. If a P-node is closed and therefore contains a real edge, all adjacent S-nodes contain a virtual edge. This case is considered in Lines 5 and 7 where also virtual edges are considered if there is an edge in the graph G or H incident to the same vertices the virtual edge is incident to. Then these trees are rooted at a real edge contained in the skeleton graph of the currently considered S-nodes. The vertices incident to the roots are then mapped in the possible subgraph isomorphism. It is sufficient to root the SP-tree T_G^{SP} at an arbitrary edge, while the SP-tree T_H^{SP} is rooted at each real edge contained in the skeleton graph of the S-node since all real edges in the skeleton graph of an S-node are either contained in the 2-connected common subgraph or not. Otherwise the common subgraph can not be 2-connected.

The SP-trees T_G^{SP} and T_H^{SP} are rooted at the edges $(u, v) = r \in E(G)$ and $(u', v') = r' \in E(H)$, respectively. Then the size of a maximum common subgraph of the subgraphs G_{uv}^r

Algorithm 4.1 2-MCS(G, H)

Input: Two 2-connected partial 2-trees G and H.

Output: Size of 2-connected-maximum common subgraph problem for partial 2-trees for the partial 2-trees G and H.

1: $T_G^{\mathrm{SP}} \leftarrow \mathrm{SP}(G)$
2: $T_H^{\mathrm{SP}} \leftarrow \mathrm{SP}(H)$
3: $mcs \leftarrow 0$
4: **for all** $(\lambda_1, \lambda_2) \in S(T_G) \times S(T_H)$ **do**
5: $\quad r \leftarrow$ arbitrary edge $(u, v) \in E(\lambda_1) \cap E(G)$
6: $\quad$ root T_G^{SP} at r
7: $\quad$ **for all** edges $r' = (u', v') \in E(\lambda_2) \cap E(H)$ **do**
8: $\quad\quad$ root T_G^{SP} at r'
9: $\quad\quad p_1 \leftarrow$ sMCS-Series$(u, v, \lambda_1, u', v', \lambda_2)$
10: $\quad\quad p_2 \leftarrow$ sMCS-Series$(u, v, \lambda_1, v', u'\lambda_2)$
11: $\quad\quad mcs \leftarrow \max\{mcs, p_1, p_2\}$
12: **return** $mcs + 2$

and $H^{r'}_{u'v'}$ is computed with respect to the fact that u is mapped to u' and v is mapped to v', respectively. In line 9 and 10 both possible mappings are considered ($\phi(u) = u', \phi(v) = v'$ and $\phi(u) = v', \phi(v) = u'$). This is necessary as the direction of the mapping is an important factor in Procedure 4.2 which is highlighted in Figure 4.1.

Figure 4.1: Example of the importance to consider both possible mappings of base vertices.

Algorithm 4.1 uses the following procedures to calculate the size of a 2-connected maximum common subgraph of the 2-connected partial 2-trees G and H:

2-MCS-Series$(u, v, \lambda, u', v'\lambda')$**:** This procedure is called for each edge, either real or virtual. The edges are defined by the vertices u, v and u', v' and the S-nodes λ and λ', respectively. The procedure returns the size of a 2-connected maximum common subgraph of the graphs $G^r_{u,r}$ and $H^{r'}_{u',v'}$ under the assumption that u is mapped to u' and v is mapped to v' in the subgraph isomorphism.

2-MCS-Edge$(e, \lambda, e', \lambda')$**:** This procedure is used in 2-MCS-Series$(u, v, \lambda, u', v'\lambda')$ to compare edges. Due to the characteristics of the SP-trees it is not possible to simply

map vertices if only one vertex is incident to a real edge while the other is incident to a virtual edge or vice versa.

With these procedures the algorithm computes the 2-connected maximum common subgraph of two 2-connected partial 2-trees. All combinations of S-nodes are compared, also each possible mapping regarding the vertices contained in the skeleton graphs of the S-nodes are considered. In Line 12 the algorithm returns the size of such a subgraph. The size is increased by 2, as due to the construction of the algorithm, the vertices which are incident to the root are not included in the result up to this point. If there is no 2-connected maximum common subgraph, the algorithm returns $-\infty$, if there is such a subgraph, the value returned is always at least three[3].

4.2.1 Computation of 2-MCS-Series

Procedure 2-MCS-Series computes a 2-connected maximum common subgraph of two 2-connected subgraph of G and H, respectively. The method is initially called with the parameters u, v, λ, u', v' and λ' where u and v are adjacent vertices in G and both in $V(S_\lambda)$ and u' is adjacent to v' and also both vertices are in $V(S_{\lambda'})$. For the common subgraph to be 2-connected, there must be a path $(u, v, v_1, \ldots, v_n, u)$ in G and a path $(u', v', v'_1, \ldots, v'_n, u')$ in H which is also contained in the subgraph. The vertices in these paths are mapped based on the direction given by the initial parameters. The main idea of the procedure is to find feasible cycles and map the vertices contained in these cycles accordingly.

There are four cases which need to be considered when the vertices are mapped. All these cases are based on the rooted SP-tree.

(1) At least one edge in the S-nodes incident to both vertices u, v and u', v' is virtual and this edge is a reference to a parent P-node,

(2) the vertices returned by the method Next are both the vertices the procedure was called with initially in Algorithm 4.1,

(3) only one vertex is a vertex the procedure was called with initially in Algorithm 4.1,

(4) the vertices which are returned by the method Next have not been considered and the edges are no references to a parent P-node.

The first case is considered in Lines 4 and 5. In this case the procedure 2-MCS-Series is called with the same arguments except for the S-node which is replaced with the parent S-node. It must be noted that this child-parent relation is given by the rooting of the SP-trees and therefore the parent is unique. In Line 6 both calls of Next have returned

[3] In this thesis all graphs are considered to be simple.

the vertex the procedure was originally called with, therefore the cycle is complete in both subgraphs. But since there is no guarantee that in both S-nodes the vertices are incident to a real edge, procedure 2-MCS-MatchEdges must be called. This is necessary because if at least one edge in the S-nodes is virtual, there may not be a cycle of the same length in both graphs containing the vertices the procedure was initially called with. If there are no such cycles, the third case will occur. In Line 7 it is checked whether the common subgraph would be a cycle in one graph and a path in the other. In this case there is no feasible 2-connected common subgraph with respect to a subgraph isomorphism mapping u to u' and v to v'.

Procedure 4.2 2-MCS-Series$(u, v, \lambda, u', v', \lambda')$

Input: Base vertices u, v of G and u', v' of H and S-nodes $\lambda \in S(T_G^{\mathrm{SP}})$ and $\lambda' \in S(T_H^{\mathrm{SP}})$, respectively.

Output: Size of a 2-connected maximum common subgraph of the graphs $G^r_{u,v}$ and $H^{r'}_{u',v'}$ under the condition that u and v are mapped to u' and v'.

1: $e = (v, w) \leftarrow$ Next(v, λ)
2: $e' = (v', w') \leftarrow$ Next(v', λ')
3: $mcs \leftarrow 0$
4: **if** $e = \mathrm{ref}(\lambda)$ **then return** MSC-Series$(u, v, \mathrm{pS}(\lambda), u', v', \lambda')$
5: **if** $e' = \mathrm{ref}(\lambda')$ **then return** MSC-Series$(u, v, \lambda, u', v', \mathrm{pS}(\lambda'))$
6: **if** $w = u$ and $w' = u'$ **then return** MatchEdge$(e, \lambda, e', \lambda')$
7: **if** $w = u$ xor $w' = u'$ **then return** $-\infty$
8: $mcs \leftarrow$ MatchEdge$(e, \lambda, e', \lambda')$+ MCS-Series$(u, w, \lambda, u', w', \lambda')$
9: **if** $e \notin E(G)$ or $e' \notin E(H)$ **then**
10: **if** $e \in E(G)$ **then** $M \leftarrow \{\lambda\}$
11: **else** $M \leftarrow \mathrm{sC}(e)$
12: **if** $e' \in E(H)$ **then** $M' \leftarrow \{\lambda'\}$
13: **else** $M' \leftarrow \mathrm{sC}(e')$
14: **for all** $(\eta, \eta') \in M \times M'$ **do**
15: $p \leftarrow$ MCS-Series$(u, w, \eta, u', w', \eta')$
16: $mcs \leftarrow \max\{mcs, p\}$
17: **return** mcs

If none of these three cases applies, the size of a 2-connected maximum common subgraph is computed recursively. In addition to the call of 2-MCS-Series the procedure 2-MCS-MatchEdges is called, see Line 8. This is necessary since at least one of the edges may be virtual in the skeleton graph of the S-nodes. The recursive call, where the

parameters u and u' are the same as in the initially call, is used to complete the cycle starting and ending at the vertices u and u', respectively.

Starting in Line 9 and ending in Line 16, a deformed case is considered. In this case, at least one of the new vertices is incident to a virtual edge. Since the cycle can contain vertices which are in the skeleton graph of different S-nodes, the vertices contained in the skeleton graphs of the child S-nodes are considered.

4.2.2 Computation of 2-MCS-MatchEdges

Procedure 2-MCS-MatchEdges is called whenever two vertices are considered to be mapped in a common subgraph. There are different cases with respect to the edge in the skeleton graph incident to both, the vertex v, v' and the vertices w, w' returned by the method Next.

If one of the edges is real while the other is virtual, the procedure checks whether the virtual edge is pertinent to a closed P-node. If so, 0 is returned and the algorithm continues, otherwise it returns $-\infty$ as this would result in a common subgraph which can not be 2-connected. If at least one of the edges is real, the procedure returns 0 since this case can only apply if both edges are real, as otherwise the first case would apply. When both edges are real, the addition of the mapping with $w \mapsto w'$ is still a feasible common subgraph and therefore procedure 2-MCS-Series can continue to compute a feasible 2-connected common subgraph.

If none of these cases occur, both edges must be virtual. In this case, all child S-nodes are compared. This is done with the maximum weighted bipartite matching problem, since only S-nodes in the SP-tree of G need to be compared to S-nodes in the SP-tree of H. If the matching has size 0, there is no feasible common subgraph of any two child S-nodes. Therefore in the common subgraph computed up to this point, the vertices v, w and v', w', respectively, are not connected and the common subgraph can not be 2-connected.

4.3 Analysis of the Algorithm

In this section it is shown that the algorithm has a polynomial running time. To do so, it is shown that the number of split graphs is polynomial in the size of a graph and that therefore the algorithm can be transformed into a dynamic programming algorithm where the table has a polynomial size. Later it is shown that due to special characteristics of the SP-tree of an outherplanar graph the algorithm has a better running time on an outerplanar graph than on a partial 2-tree.

Lemma 4.3.1. *Let G be a 2-connected partial 2-tree and $n = |V(G)|$, then the number of split graphs of G is $\mathcal{O}(n^2)$.*

Procedure 4.3 2-MCS-EDGES$(e, \lambda, e', \lambda')$

Input: Edges $e = (u, v) \in E(S_\lambda)$ and $e' = (u', v') \in E(S'_\lambda)$

Output: Size of the 2-connected maximum common subgraph of the graphs $G^r_{u,v}$ and $H^{r'}_{u',v'}$ under the condition that u and v are mapped to u' and v'.

1: **if** $e \in E(G)$ xor $e' \in E(H)$ **then return** $-\infty$
2: **if** e or e' is a *real* edge in S_λ resp. $S_{\lambda'}$ **then return** 0
3: $M \leftarrow \mathrm{cS}(e)$
4: $M' \leftarrow \mathrm{cS}(e')$
5: **for all** $f = (\eta, \eta') \in M \times M$ **do**
6: $\quad w(f) \leftarrow$ MCS-SERIES$(u, v, \eta, u', v'\eta')$
7: $p \leftarrow$ MWBMATCHING(M, M', w)
8: **if** $p = 0$ and $e \notin E(G), e' \notin E(H)$ **then return** $-\infty$
9: **return** p

Proof. The split graphs $G^r_{u,v}$ of a graph G are defined by a potential separator $P = \{u, v\}$ of G and an edge $r \in E(G)$. The edge r is used to root the SP-tree of G and thus in combination with P is used to determine which components are deleted from G to compute the split graph. As a potential separator consists of two vertices, there are $\mathcal{O}(n^2)$ potential separators and the number of split graphs for a potential separator P is $c(P) := |\mathcal{C}(G^\star_P \backslash P)|$.

If $c(P)$ is greater than 2, the separator must be a separator of G. In Lemma 4.1.1 it is stated, that the connected components resulting from the removal of the separator are the S-nodes pertinent to the P-node whose skeleton graph contains the separator. Therefore $c(P) < n$ and in this case the number of these separators must be in $\mathcal{O}(n^2)$. Then, if $c(P)$ is equal 2, there are exactly two connected components if P is removed. Thus $\mathcal{O}(n^2)$ split graphs can be computed with such a separator. Since the number of connected components gained by the removal of a potential separator is always bounded like the number of connected components gained by the removal of a separator, the number of split graphs is bounded by $\mathcal{O}(n^2)$. □

Theorem 4.3.2. *The 2-connected maximum common subgraph problem for two partial 2-trees G and H can be solved in $\mathcal{O}(n^6)$, where $n = \max\{|V(G)|, |V(H)|\}$.*

Proof. To proof the running time of the algorithm, the procedures 2-MCS-SERIES and 2-MCS-MATCHEDGES are transformed into dynamic programming algorithms. The split graphs, the 2-connected maximum common subgraphs, are defined by the parameters given to these procedures. Therefore the cells of the table are associated with a pair of split graphs, one split graph of G and one split graph of H. As the number of split graphs of G and H, respectively is bounded by $\mathcal{O}(n^2)$, see Lemma 4.3.1. The size of the table storing the size of the 2-connected maximum common subgraphs of two split graphs is

bounded by $\mathcal{O}(n^4)$. It must be noted that the number of edges and vertices in both graphs is obviously bounded linear by n, but this is also true for the virtual edges in the SP-tree since a virtual edge represents a path in the graph.

Without loss of generality it can be assumed that, whenever procedure 2-MCS-Series is called, the size of all smaller split graphs, than the one defined by the parameters the procedure is called with, are already computed. As the smallest split graph consists of a single edge and two vertices this can easily be accomplished. Since these results can be used in time $\mathcal{O}(1)$, the dominating part of the procedure can be found in Line 14. This loop is only called if at least one considered edge, given by the parameters, is a virtual edge. As there are only $\mathcal{O}(n)$ S-nodes pertinent to any P-node in a SP-tree, this loop requires time $\mathcal{O}(n^2)$. Therefore the total running time needed to compute all calls of 2-MCS-Series is $\mathcal{O}(n^6)$.

In Procedure 2-MCS-Edges is a loop in Line 5 which requires running time $\mathcal{O}(n^2)$ due to the same arguments stated above. Also the method MwbMatching is used. The maximum weighted bipartite matching problem can be solved in time $\mathcal{O}(n^3)$ as described before. The matching problem only needs to be solved if both edges are virtual edges, as otherwise the procedure returns in Line 1 if there is one real edge and one virtual edge, and returns in Line 2 if both edges are real. As the number of real and virtual edges is linear in n, there are $\mathcal{O}(n^2)$ calls of MbwMatching and therefore the total running time needed to compute all calls of 2-MCS-MatchEdegs is $\mathcal{O}(n^5)$.

So all in all the running time is dominated by the time needed to compute 2-MCS-Series, so the 2-connected maximum common subgraph problem for partial 2-trees can be solved in time $\mathcal{O}(n^6)$. □

Since each outerplanar graph is also a partial 2-tree, the algorithm can also be used to solve the 2-connected maximum common subgraph problem for outerplanar graphs. SP-trees of outerplanar graphs have a characteristic which can be used in the analysis of the running time of the algorithm. This characteristic can be found in Lemma 4.3.3.

Lemma 4.3.3. *Let G be a 2-connected partial 2-tree and T_G^{SP} the SP-tree decomposition of G, then G is outerplanar if and only if all P-nodes in T_G^{SP} have degree two.*

Proof. The class of outerplanar graphs can be defined by the forbidden minors K_4 or $K_{2,3}$. Assume that there is a P-node λ in $V_P(T_G^{\mathrm{SP}})$ with degree of at least three. Then there are at least three paths $u \rightsquigarrow v$ where $u, v \in V(S_\lambda)$ and therefore G has a $K_{2,3}$ as minor since there are two vertices connected by three paths containing at least one vertex not in $V(S_\lambda)$. □

As each P-node in a 2-connected outerplanar graph has degree two, the following theorem is implied by the proof of Theorem 4.3.2.

Theorem 4.3.4. *The 2-connected maximum common subgraph problem for two outerplanar graphs G and H can be solved in $\mathcal{O}(n^4)$, where $n = \max\{|V(G)|, |V(H)|\}$.*

Proof. In both procedure 2-MCS-Series and 2-MCS-MatchEdges, there are loops[4] over the Cartesian product of two sets M and M'. These loops are only required if there is at least one virtual edge considered. These sets contain all S-nodes pertinent to the considered virtual edges which have not already been considered or the current S-node, if the edge is not virtual. Since there is exactly one S-node for each graph, the sets contain exactly one pair of S-nodes, therefore the loop in Procedure 2-MCS-Series has running time $\mathcal{O}(1)$ and also the matching can be solved in time $\mathcal{O}(1)$ since there is only one possibility. Thus all in all, the running time for the procedure 2-MCS-Series is $\mathcal{O}(n^4)$ and for procedure 2-MCS-Edges $\mathcal{O}(n^3)$.

Therefore the running time of the algorithm is still dominated by the Procedure 2-MCS-Series, but if both graphs are outerplanar, the algorithm computes the 2-connected maximum common subgraph in $\mathcal{O}(n^4)$. □

4.4 Summary

In this section an algorithm for the 2-connected maximum common subgraph problem for partial 2-trees is presented. First it is explained why the approach used presented in [4] cannot be easily extended to work with partial 2-trees. Then, separators are introduced and a decomposition into subgraphs, called split graphs, based on separators and a rooted SP-tree are presented. Using these split graphs the 2-connected maximum common subgraph of two 2-connected partial 2-trees can be computed. The separators are used in the analysis of the running time as it can be shown that the number of split graphs defined with these separators is polynomial in the size of a graph and that each 2-connected subgraph of the input graph is a 2-connected subgraph of such a split graph.

[4] See Line 14 and 5, respectively.

Chapter 5

The Block-and-Bridge Preserving Maximum Common Subgraph Problem in Partial 2-Trees

The restriction that both input graphs must be 2-connected partial 2-trees is quite hard as one of the applications of the maximum common subgraph problem is finding similarities of molecules [25] which are mostly not 2-connected when represented as graph. In this chapter another restricted version of the maximum common subgraph problem for partial 2-trees is considered. The restrictions regarding the maximum common subgraph problem for partial 2-trees are introduced in [26] and also used in [4].

Let G and H be arbitrary partial 2-trees, ϕ a common subgraph isomorphism of $G' \subseteq G$ and $H' \subseteq H$, then the common subgraph is called *block-and-bridge preserving* if the following conditions are satisfied:

(BBP1) For any two vertices $u, v \in V(G')$, u and v are in the same block in G' if and only if they are in the same block in G. The same must be true for each pair of vertices in H'.

(BBP2) Let $u, v \in V(G')$ then u and v are contained in a bridge node if and only if u and v are contained in a bridge node in G. As above the same must be true for each pair of vertices in H'.

The problem of finding a maximum common subgraph satisfying restrictions (BBP1) and (BBP2) is called *block-and-bridge preserving maximum common subgraph problem*. In this chapter it is shown that the block-and-bridge preserving maximum common subgraph problem in partial 2-trees can be solved in polynomial time. To do so, the algorithm presented in Chapter 4 is extended.

5.1 Characteristics of a Block-and-Bridge Preserving Maximum Common Subgraph

Before presenting the extension of the algorithm to solve the block-and-bridge preserving maximum common subgraph problem in partial 2-trees, there is need for some background information about the restrictions. It is easy to see that a 2-connected maximum common subgraph of two 2-connected partial 2-trees is also a block-and-bridge preserving maximum common subgraph of these graphs. Hence each feasible solution of the block-and-bridge preserving maximum common subgraph problem of 2-connected partial 2-trees is a feasible solution of the 2-connected maximum common subgraph problem in 2-connected partial 2-trees. Examples of feasible and infeasible block-and-bridge preserving maximum common subgraphs can be found in Figure 5.1.

Lemma 5.1.1. *Let G and H be two graphs, ϕ a common subgraph isomorphism of $G' \subseteq G$ and $H' \subseteq H$, then any two vertices in G or H which are in different 2-connected components cannot be in the same 2-connected component of G' resp. H'.*

Proof. Let G, H, G', H' and ϕ be defined as above, also let $u, v \in V(G)$ be two vertices in different 2-connected components in G. There is at least one vertex $w \in V(G)$ which is contained in all paths $u \rightsquigarrow v$ as there is at least one cutvertex between different 2-connected components. Otherwise, if there is a path $u \rightsquigarrow v$ which does not contain w, u and v would be in the same 2-connected component. If u, v and w are mapped to u', v' and w' in a common subgraph and u', v' are in the same 2-connected component, then there is a path $u' \rightsquigarrow v'$ in the common subgraph which does not contain w'. As there is no mapping for such a path in G, the common subgraph cannot be an induced common subgraph. □

Lemma 5.1.2. *Vertices associated with one bridge in G or H cannot be contained in one 2-connected component in a common subgraph of G and H.*

Proof. Let u and v be vertices contained in one bridge of G. If u, v are mapped to $u', v' \in V(H)$ where u', v' are contained in a 2-connected component C. Let $C = \{u', v', w_1, \ldots, w_k\}$ and $w_1, \ldots, w_k \in V(H)$. Therefore, $u', v', w_1, \ldots, w_k$ are the vertices contained in the 2-connected component, then there is no feasible mapping for $w_1, \ldots, w_k$ in G because, if there were such a mapping, u and v could not be a bridge. □

Lemma 5.1.1 and Lemma 5.1.2 are valid for arbitrary maximum common subgraphs regardless restrictions (BBP1) and (BBP2) required for block-and-bridge preserving maximum common subgraphs. Concerning the restrictions it is not possible to map vertices in a 2-connected component of G to vertices in different 2-connected component in H, as this would be contradicting condition (BBP1). While it is possible that a vertex is in different 2-connected components, that can only be the case if and only if the vertex is a cutvertex.

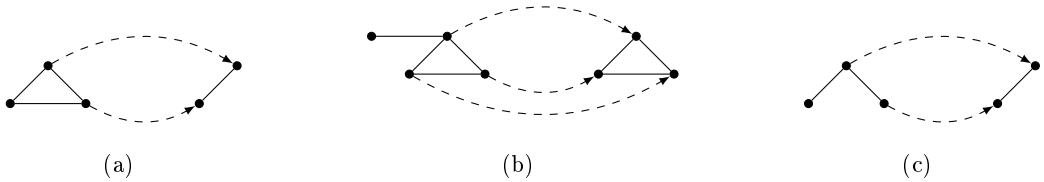


Figure 5.1: Maximum common subgraph which is not block-and-bridge preserving because vertices in a non-bridge block are mapped to vertices in a bride block (a) and maximum common subgraphs which are block-and-bridge preserving (b) and (c).

Two vertices are considered to be in different 2-connected components if and only if there is no 2-connected component containing both vertices. Also due to restriction (BBP2) each two vertices contained in one bridge in a common subgraph of G and H must be contained in one bridge in G and H. Therefore it is not necessary to compare non-bridge and bridge blocks of the BC-tree decompositions of the graphs, as vertices contained in the first can never be mapped to vertices in the latter and vice versa if they are not for cutvertices.

As vertices in one non-bridge block cannot be mapped to vertices in a bridge block or to vertices in more than one non-bridge block if not both vertices are cutvertices, Algorithm 4.1 presented in Chapter 4.2 is required to behave differently to compute a block-and-bridge preserving maximum common subgraph instead of a 2-connected maximum common subgraph if two cutvertices are mapped, otherwise one of the restrictions (BBP1) or (BBP2) would be disregarded.

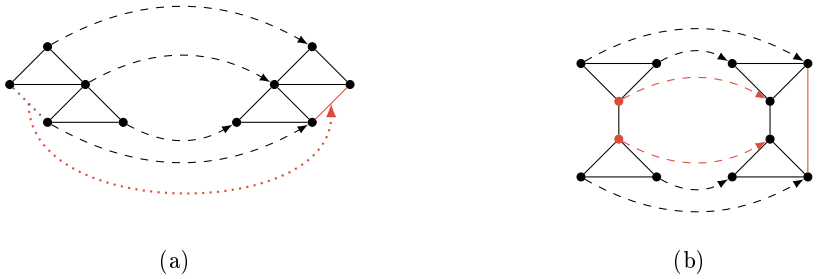
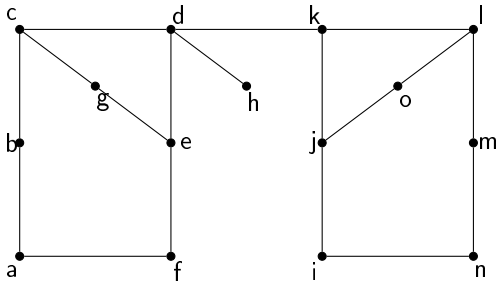


Figure 5.2: Example of Lemma 5.1.1 in (a) and Lemma 5.1.2 in (b) where the edges and vertices highlighted in red problematic for a block-and-bridge preserving common subgraph.

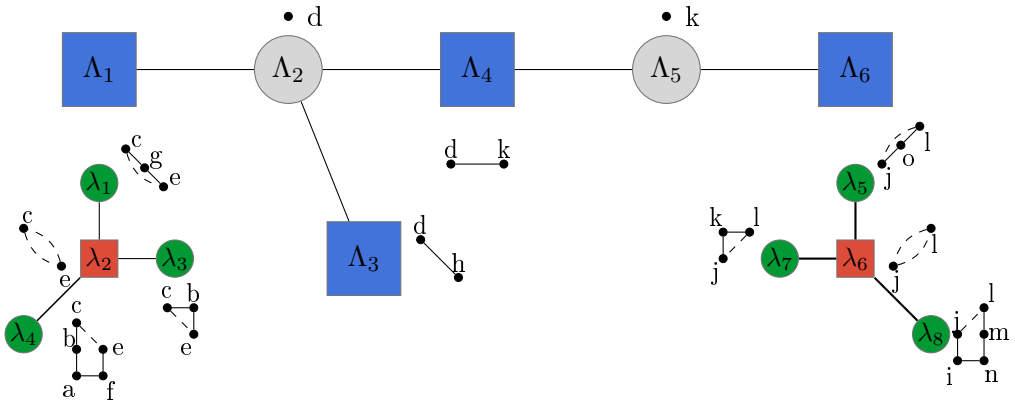
Figure 5.2 highlights the problematic edges and vertices mentioned in Lemma 5.1.1 and Lemma 5.1.2. In Figure 5.2(a) the shown mapping is not feasible, because the edge highlighted in red has no corresponding edge in the other graph. Whenever vertices in two connected components are mapped to vertices in one 2-connected component, the a common subgraph can never be 2-connected. In Figure 5.2(b) the shown mapping is not feasible, again due to the edge highlighted in red has no corresponding edge in the other graph. Hence, a common subgraph of a bridge and a 2-connected component cannot be 2-connected.

Partial 2-trees are not necessarily 2-connected. Therefore it is not possible to represent all partial 2-trees by their unique SP-tree decomposition. To get a uniform and unique representation the BC-tree decomposition described in Section 2.3.3 is used, because a unique BC-tree decomposition exists for arbitrary partial 2-trees. The main idea of the extension of the algorithm presented in Chapter 4 is to behave differently if and only if two cutvertices are mapped. If the algorithm would behave differently at any other condition it would not compute a block-and-bridge preserving maximum common subgraph as stated before. Hereby it can be guaranteed, that the algorithm behaves exactly the same if both input graphs are 2-connected partial 2-trees.

For the reader's convenience Greek uppercase letters are used to describe B- and C-nodes of the BC-trees, whereas Greek lowercase letters are used to describe the S- and P-nodes of the SP-trees contained in the B-Nodes of the BC-trees. Also the elements of a BC- and SP-tree decomposition are called nodes and edges whereas the elements of the graphs G and H are called vertices and edges.



(a)



(b)

Figure 5.3: A graph G (a) and its BC-tree decomposition T_G^{BC} with an example of the naming of the nodes of the BC-tree and the associated SP-trees (b). B-nodes are blue, C-nodes gray, S-nodes green and P-nodes red. The associated skeleton graph is drawn next to the nodes.

Due to the construction of the algorithm, just as in Chapter 4 a child-parent relation is needed. As the nodes in BC-trees are associated with skeleton graphs which are either

associated with a bridge or a SP-tree, the SP-trees needs to have a child-parent relation, too. This relation must be implicitly given by the child-parent relation of the BC-tree. In addition to comparing the nodes, the BC-trees are rooted at edges contained in the skeleton graphs of these nodes. With rooting the BC-tree at a B-node all SP-trees in the non-bridge nodes are rooted implicitly. This is done by rooting the SP-trees contained in the non-bridge nodes after the following rules. Let T^{BC} be a BC-tree and $\Lambda \in V_{\mathrm{B}}(T^{\mathrm{BC}})$ the B-node the BC-tree is rooted at, then there are two cases:

$\Lambda \in V_{\mathrm{Br}}(T^{\mathrm{BC}})$ Thus S_Λ contains exactly one edge, this edge is the root of the tree.

$\Lambda \in V_{\mathrm{Bl}}(T^{\mathrm{BC}})$ Hence S_Λ contains at least three edges, one edge needs to be distinguished to be the root. Then $T^{\mathrm{SP}}_\Lambda = \mathrm{SP}(S_\Lambda)$ is rooted as described in the previous chapters.

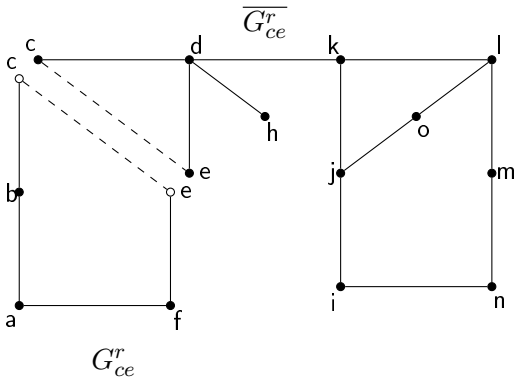
All SP-trees in child non-bridge nodes Ξ of B-node Λ are rooted in the following way. Let $T^{\mathrm{SP}}_\Xi = \mathrm{SP}(S_\Xi)$ be the SP-tree of the skeleton graph contained in the Bl-node. Then there are two cases, either Ξ is the root or there is a vertex $u \in V(S_\Xi)$ the vertex which is pertinent to the C-node which is the parent of Ξ such that $u \in V(S_\Lambda)$. In the first case, each vertex is chosen as root once during the algorithm. In the second case, the root is given implicit by the following rules.

(RBL1) If $\exists \xi \in V_P(T^{\mathrm{SP}}_\Xi)$ such that $u \in V(S_\xi)$, then T^{SP}_Ξ is rooted at ξ.

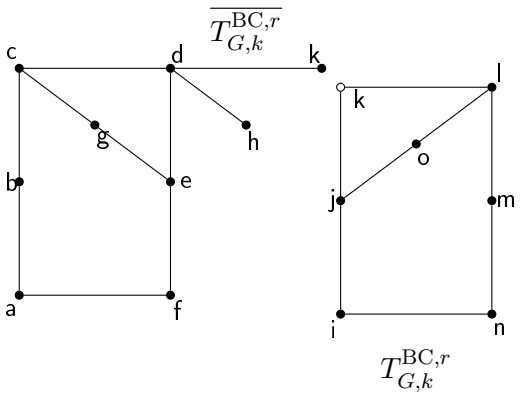
(RBL2) If $\nexists \xi \in V_P(T^{\mathrm{SP}}_\Xi)$ such that $u \in V(S_\xi)$, then there is exactly one $\pi \in V_S(S_\Xi)$ such that $u \in V(S_\pi)$. In this case T^{SP}_Ξ is rooted at π.

As T^{SP}_Ξ can be rooted at a P-node, there is no edge at which the SP-tree must be rooted. In the case that no such edge exists the SP-tree is simply rooted at a single vertex. During Algorithm 5.1 this is only the case when two cutvertices are mapped. For the initial Bl-nodes, there is always an edge at which they are rooted. These rooted BC- resp. SP-trees are especially used in Procedure 5.3. Let G be a graph and $u \in V(G)$ such that u is a cutvertex, here $cB(u)$ (as seen in lines 3 and 4) denotes all B-nodes which are children of the C-node Λ with $u = V(S_\Lambda)$. Also let T^{SP} be a SP-tree and $\lambda \in V_P(T^{\mathrm{SP}})$, then $\mathrm{cS_P}(\lambda)$ (as seen in line 13) denotes all S-nodes which are children of λ. This case is related to (RBL2), as all S-nodes which contain virtual edges pertinent to λ are children of λ.

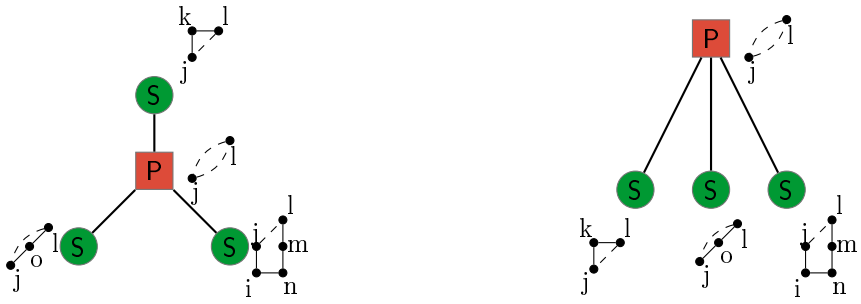
Let T^{BC}_G be the BC-tree decomposition of a partial 2-tree G. Also let $r \in E(G)$ be an edge of G, then $T^{\mathrm{BC},r}_G$ denotes the BC-tree rooted at r. Let $u \in V(G)$ be a cutvertex. Also let $C = \{C_1, C_2, \ldots, C_n\}$ be the connected components of $G - u$. Without loss of generality let C_1 be the connected component such that $r \in E(C_1)$. Then $\overline{T^{\mathrm{BC},r}_{G,u}}$ denotes the induced subgraph $G[V(C_1) \cup \{u\}]$ and $T^{\mathrm{BC},r}_{G,u}$ denotes the induced subgraph $G[\bigcup_{i=2}^{n} C_i \cup \{u\}]$, called the *block split graphs* of G.



(a)



(b)



(c) (d)

Figure 5.4: Examples of the different split graphs of the graph G in Figure 5.3(a) rooted at $r = (d, h)$. In (a) the split graphs G^r_{ce} and $\overline{G^r_{ce}}$ are shown, while in (b) the block split graphs $T^{\mathrm{BC},r}_{G,k}$ and $\overline{T^{\mathrm{BC},r}_{G,k}}$. The rooting of a SP-tree given by the block split graph in (c) in Figure (d) and a rooting of a SP-tree if the cutvertex the block split graph would be split at j or l.

5.2 An Algorithm for the Block-and-Bridge Preserving Maximum Common Subgraph Problem in Partial 2-Trees

The Algorithm used to solve the block-and-bridge preserving maximum common subgraph problem in partial 2-trees is an extension of Algorithm 4.1. Therefore the structure and functioning is similar. The Algorithm itself uses the following procedures:

BBP-MCS(G, H) computes the size of a block-and-bridge preserving maximum common subgraph of two partial 2-trees G and H. In this method, the BC-tree decompositions are computed and rooted accordingly while all feasible combinations of B-nodes of both BC-trees are compared.

BBP-MCS-Series$(u, v, \lambda, u', v'\lambda')$ computes the maximum common subgraph of the SP-trees in which the S-nodes λ and λ' are contained. The computation is done with respect to the root chosen in BBP-MCS.

BBP-MCS-Cut(u, u') computes the maximum common subgraph of two bridges with respect to the chosen root in the case that u and u' are mapped. This method is one of the extensions to Algorithm 4.1.

BBP-MCS-Edges(e, e') is used in BBP-MCS-Series to compute the 2-connected maximum common subgraph of two non-bridge nodes by verifying that two vertices can be mapped, taking their incident edges in the skeleton graph into account.

Algorithm 5.1 is the main method of the algorithm. As Algorithm 4.1 compares all S-nodes of the SP-trees, all feasible combinations of B-nodes in the BC-tree decomposition are compared. Due to restrictions (BBP1) and (BBP2) only nodes of the same type need to be compared as it is not feasible to map an edge contained in a bridge to an edge contained in a non-bridge node. Also there is no need to compare multiple non-bridge nodes and compare them to a single non-bridge node as this contradicts condition (BBP1). Hence all suitable nodes are compared with each other in Line 4 and 14, respectively. In these loops, the BC-trees are rooted accordingly. It must be noted, that both, the BC-tree decomposition and the SP-tree decomposition of non-bridge blocks can be computed in linear time [27] and [17], respectively.

In Line 4 the non-bridge blocks are mapped. This is done in exactly the same way as in Algorithm 4.1. The argumentation why it is sufficient to compare one edge to all other in each S-node is exactly the same. One of the additions to the algorithm can be found in Lines 11 and 12, where in addition to sMCS-Series the Procedure sMCS-Cut is called also. Since there can be cutvertices, all adjacent B-nodes in the BC-tree decompositions need to be taken into account, too.

In Line 14 bridge nodes are compared. Due to the structure of bridges only two combinations of mappings are possible for each bride. If the bridge is not contained in a

Procedure 5.1 BPP-MCS(G, H)

Input: Two partial 2-trees G and H.

Output: Size of a block-and-bridge preserving preserving maximum common subgraph for the partial 2-trees G and H.

1: $T_G^{\mathrm{BC}} \leftarrow \mathrm{BC}(G)$
2: $T_H^{\mathrm{BC}} \leftarrow \mathrm{BC}(H)$
3: $mcs \leftarrow 0$
4: **for all** $(\Lambda, \Lambda') \in V_{\mathrm{Bl}}(T_G^{\mathrm{BC}}) \times V_{\mathrm{Bl}}(T_H^{\mathrm{BC}})$ **do**
5: $\quad T_\Lambda^{\mathrm{SP}} \leftarrow \mathrm{SP}(S_\Lambda^{\mathrm{BC}})$
6: $\quad T_{\Lambda'}^{\mathrm{SP}} \leftarrow \mathrm{SP}(S_{\Lambda'}^{\mathrm{BC}})$
7: $\quad$ **for all** $(\lambda, \lambda') \in V_S(T_\Lambda^{\mathrm{SP}}) \times V_S(T_{\Lambda'}^{\mathrm{SP}})$ **do**
8: $\quad\quad r \leftarrow$ arbitrary real edge $(u, v) \in E(S_\lambda)$; root T_G^{BC} at r
9: $\quad\quad$ **for all** *real edges* $r' = (u', v') \in E(S_{\lambda'})$ **do**
10: $\quad\quad\quad$ root T_H^{BC} at r'
11: $\quad\quad\quad p_1 \leftarrow$ sMCS-Series$(u, v, \lambda, u', v', \lambda') +$ sMCS-Cut(u, u')
12: $\quad\quad\quad p_2 \leftarrow$ sMCS-Series$(u, v, \lambda, v', u'\lambda') +$ sMCS-Cut(u, u')
13: $\quad\quad\quad mcs \leftarrow \max\{mcs, p_1, p_2\}$
14: **for all** $(\Lambda, \Lambda') \in V_{\mathrm{Br}}(T_G^{\mathrm{BC}}) \times V_{\mathrm{Br}}(T_H^{\mathrm{BC}})$ **do**
15: $\quad r = (u, v) \leftarrow E(S_\Lambda^{\mathrm{BC}})$; root T_G^{BC} at r
16: $\quad r' = (u', v') \leftarrow E(S_{\Lambda'}^{\mathrm{BC}})$; root T_H^{BC} at r'
17: $\quad p_1 \leftarrow$ sMCS-Cut$(u, u') +$ sMCS-Cut(v, v')
18: $\quad p_2 \leftarrow$ sMCS-Cut$(u, v') +$ sMCS-Cut(v, u')
19: $\quad mcs \leftarrow \max\{mcs, p_1, p_2\}$
20: **return** mcs +2

maximum common subgraph, but one of the vertices in the bridge is, this vertex is also contained in the adjacent nodes regarding the BC-tree. In Line 20 before the result is returned, it is increased by two. This is because the two nodes incident to the chosen root are never counted in any method called by Algorithm 5.1. Thus to return the correct result, this increment is necessary.

5.2.1 Computation of BBP-MCS-Series

Procedure 5.2 is the extended version of Procedure 4.2. The crucial changes can be found in Lines 9 and 9 where also the result of Procedure BBP-MCS-Cut is added to the size of the computed subgraph. Since the graph may contain cutvertices, all B-nodes in the BC-trees which are adjacent to the C-node associated with the cutvertex need to be considered for the common subgraph. The rest of the procedure is the same. There are no more changes necessary for the reasons discussed above. Due to this change and the

fact that it only affects graphs which contain cutvertices the algorithm behaves exactly the same as described in Chapter 4 if there are no cutvertices in at least one of the input graphs.

Procedure 5.2 BBP-MCS-SERIES$(u, v, \lambda, u', v'\lambda')$

Input: Base vertices u, v of G and u', v' of H and S-nodes $\lambda \in S(T_G)$ resp. $\lambda' \in S(T_H)$.

Output: Size of a block-and-bridge preserving maximum common 2-connected subgraphs of the split graphs G^r_{uv} and $H^{r'}_{u'v'}$ under the condition that u and v are mapped to u' and v'.

1: $e = (v, w) \leftarrow$ NEXT(v, λ)
2: $e' = (v', w') \leftarrow$ NEXT(v', λ')
3: $mcs \leftarrow 0$
4: **if** $e = \text{ref}(\lambda)$ **then return** MSC-SERIES$(u, v, \text{pS}(\lambda), u', v', \lambda')$
5: **if** $e' = \text{ref}(\lambda')$ **then return** MSC-SERIES$(u, v, \lambda, u', v', \text{pS}(\lambda'))$
6: **if** $w = u$ and $w' = u'$ **then return** MATCHEDGE$(e, \lambda, e', \lambda')$
7: **if** $w = u$ xor $w' = u'$ **then**
8: **return** $-\infty$
9: $mcs \leftarrow$ MATCHEDGE(e, e') + sMCS-SERIES$(u, w, \lambda, u', w', \lambda')$ + sMCS-CUT$(w, w') + 1$
10: **if** $e \notin E(G)$ or $e' \notin E(H)$ **then**
11: **if** $e \in E(G)$ **then** $M \leftarrow \{\lambda\}$
12: **else** $M \leftarrow \text{sC}(e)$
13: **if** $e' \in E(H)$ **then** $M' \leftarrow \{\lambda'\}$
14: **else** $M' \leftarrow \text{sC}(e')$
15: **for all** $(\eta, \eta') \in M \times M'$ **do**
16: $p \leftarrow$ MCS-SERIES$(u, w, \eta, u', w', \eta')$ + sMCS-CUT(w, w')
17: $mcs \leftarrow \max\{mcs, p\}$
18: **return** mcs

5.2.2 Computation of PPB-MCS-CUT

Since there are no cutvertices in 2-connected graphs, they are not considered by the algorithm presented in Chapter 4. Therefore, the addition of Procedure 5.3 is the biggest change of this algorithm. Whenever two vertices are mapped in Procedure 5.2 Line 9, it is checked whether both vertices are cutvertices. If so, all blocks pertinent to the cutvertices are compared. To do so, all children of the cutvertices are considered in Lines 3 and 4. Like in Algorithm 5.1 only the same types of nodes in the BC-trees need to be compared since the common subgraph still needs to be 2-connected.

Cutvertices are also mapped whenever two vertices in a bridge are mapped, as all vertices in a bridge node are cutvertices per definition. So there is no need for an additional procedure for bridges, as this case is covered by Procedure 5.3.

The main idea of the procedure is to compare all child B-nodes and find the greatest block-and-bridge preserving common subgraph of the block split graphs $T^{\mathrm{BC},r}_{G,u}$ and $T^{\mathrm{BC},r}_{H,u'}$. Since the result must be block-and-bridge preserving, non-bridge nodes in $T^{\mathrm{BC},r}_{G,u}$ are only compared with non-bridge nodes in $T^{\mathrm{BC},r}_{H,u'}$, see Line 6 and the same goes for bride nodes, see Line 21.

When comparing two non-bridge nodes the result cannot be the size of an arbitrary block-and-bridge preserving maximum common subgraph. The crucial point is that the cutvertices must be mapped accordingly. This is ensured in Line 17 where the arbitrary real edge must be incident to the cutvertex, and also in Line 18 where all edges which are considered must be incident to the cutvertex. It is sufficient to consider only an arbitrary edge of one of the S-nodes since it must be contained in the block-and-bridge preserving common subgraph if all real edges and vertices in the skeleton graph of that S-node are contained in the common subgraph.

The comparison of bridge nodes is simpler since the only vertex in a bridge not already considered is the other vertex in the bridge. The result of the mapping of these cutvertices is then taken into account. The computation of the best matching can be done by solving a maximum bipartite matching problem.

Let $Br^G_1, Br^G_2, \ldots, Br^G_h, Bl^G_1, Bl^G_2, \ldots, Bl^G_i$ and $Br^H_1, Br^H_2, \ldots, Br^H_j, Bl^H_1, Bl^H_2, \ldots, Bl^H_k$ be all child nodes of the mapped C-nodes in G resp. H. Also let u and u' be the vertices in G resp. H associated with the mapped C-nodes. To compare all blocks, an edge-weighted bipartite graph $F = (X \cup Y, E, w)$ is constructed:

$$X = \{Br^G_1, Br^G_2, \ldots, Br^G_h, Bl^G_1, Bl^G_2, \ldots, Bl^G_i\} \tag{5.1}$$

$$Y = \{Br^H_1, Br^H_2, \ldots, Br^H_j, Bl^H_1, Bl^H_2, \ldots, Bl^H_k\} \tag{5.2}$$

$$E = \{(x, y) \colon x \in X, y \in Y\} \tag{5.3}$$

The weight $w \colon E \to \mathbb{N} \cup \{-\infty\}$ is the size of a block-and-bridge preserving maximum common subgraph if the vertices in the blocks incident to the edge are mapped. For two non-bridge blocks, there may be multiple possible combinations whereas for two bridge blocks, there is only one possible combination as the skeleton graph of a bridge contains exactly two vertices and one edge incident to those vertices. All edges incident to nodes associated with a bridge and a non-bridge block have weight $-\infty$ as a mapping of vertices contained in the skeleton graphs of these nodes contradicts to restriction (BBP1). Therefore

Procedure 5.3 BBP-MCS-CUT(u, u')

Input: Two cutvertices $u \in V(G)$, $u' \in V(H)$.

Output: Size of a block-and-bridge preserving maximum common subgraph of the block split graphs $T^{\mathrm{BC},r}_{G,u}$ and $T^{\mathrm{BC},r}_{H,u'}$ under the condition that u is mapped to u'.

1: **if** u or v is not pertinent to a node C-node **then**
2: **return** 0
3: $M \leftarrow \mathrm{cB}(u)$
4: $M' \leftarrow \mathrm{cB}(u')$
5: $mcs \leftarrow 0$
6: **for all** $f = (\Lambda, \Lambda') \in V_{\mathrm{Bl}}(M) \times V_{\mathrm{Bl}}(M)$ **do**
7: $T^{\mathrm{SP}}_{\Lambda} \leftarrow \mathrm{SP}(S^{\mathrm{BC}}_{\Lambda})$
8: $T^{\mathrm{SP}}_{\Lambda'} \leftarrow \mathrm{SP}(S^{\mathrm{BC}}_{\Lambda'})$
9: **if** $\exists \lambda \in V_P(T^{\mathrm{SP}}_{\Lambda})$ such that $u \in V(S_\lambda)$ **then**
10: $N \leftarrow \mathrm{cS_P}(\lambda)$
11: **else** $N \leftarrow \{\lambda\}$
12: **if** $\exists \lambda' \in V_P(T^{\mathrm{SP}}_{\Lambda'})$ such that $u' \in V(S_{\lambda'})$ **then**
13: $N' \leftarrow \mathrm{cS_P}(\lambda')$
14: **else** $N' \leftarrow \{\lambda'\}$
15: $mcs_{\mathrm{tmp}} \leftarrow 0$
16: **for all** $(\lambda, \lambda') \in N \times N'$ **do**
17: $(\mathbf{u}, v) \leftarrow$ arbitrary edge in $E(S_\lambda) \cap E(G)$
18: **for all** edges $(\mathbf{u'}, v') \in E(S_{\lambda'}) \cap E(H)$ **do**
19: $mcs_{\mathrm{tmp}} \leftarrow \max\{mcs_{\mathrm{tmp}}, \text{sMCS-SERIES}(u, v, \lambda, u', v', \lambda')\}$
20: $w(f) \leftarrow mcs_{\mathrm{tmp}}$
21: **for all** $f = (\eta, \eta') \in V_{\mathrm{Br}}(M) \times V_{\mathrm{Br}}(M)$ **do**
22: $(u, v) \leftarrow \text{NEXT}(u, \lambda)$
23: $(u', v') \leftarrow \text{NEXT}(u', \lambda')$
24: $w(f) \leftarrow \text{sMCS-CUT}(v, v') + 1$
25: $mcs \leftarrow \text{MWBMATCHING}(M, M', w)$
26: **return** mcs

$$w(\Lambda, \Lambda') = \begin{cases} 1 + \max_{v,v',\lambda,\lambda'} \{\text{BPP-MCS-SERIES}(u, v, \lambda, u', v', \lambda'))\}, & (1) \\ 1 + \text{BBP-MCS-CUT}(v, v'), & (2) \\ -\infty, & \text{otherwise.} \end{cases} \tag{5.4}$$

Case (1) is selected if $\Lambda \in V_{\mathrm{Bl}}(T^{\mathrm{BC}}_G)$ and $\Lambda' \in V_{\mathrm{Bl}}(T^{\mathrm{BC}}_H)$ and therefore both nodes are non-bridge nodes and the maximum is calculated over $\lambda \in V_S(\mathrm{skel}(\Lambda)), \lambda' \in V_S(\mathrm{skel}(\Lambda'), v \in$

$E_{\mathrm{r}}(\mathrm{skel}(\lambda))$ and $v' \in E_{\mathrm{r}}(\mathrm{skel}(\lambda'))$. Case (2) is selected if $\Lambda \in V_{\mathrm{Br}}(T_G^{\mathrm{BC}})$ and $\Lambda' \in V_{\mathrm{Br}}(T_H^{\mathrm{BC}})$ and therefore both nodes are bridge nodes and v resp. v' are unique as they are the other vertex contained in the skeleton graph of the bridge node.

5.2.3 Computation of BBP-MCS-Edges

Since the result of the algorithm still needs to be 2-connected, Procedure 5.4 is not changed. This is due to the fact that the algorithm needs to behave exactly the same when it comes to non-bridge blocks as presented in Chapter 4. The procedure is listed below for an complete overview of the algorithm.

Procedure 5.4 BBP-MCS-Edges$(e, \lambda, e', \lambda')$

Input: Edges $e = (u, v) \in E(S_\lambda)$ and $e' = (u', v') \in E(S'_\lambda)$

Output: Size of a block-and-bridge preserving maximum common subgraph of the block split graphs $T_{G,u}^{\mathrm{BC},r}$ and $T_{H,u'}^{\mathrm{BC},r}$ under the condition that u is mapped to u'.

1: **if** $e \in E(G)$ xor $e' \in E(H)$ **then return** $-\infty$
2: **if** e or e' is a *real* edge in S_λ resp. $S_{\lambda'}$ **then return** 0
3: $M \leftarrow \mathrm{cS}(e)$
4: $M' \leftarrow \mathrm{cS}(e')$
5: **for all** $f = (\eta, \eta') \in M \times M$ **do**
6: $\quad w(f) \leftarrow$ MCS-Series$(u, v, \eta, u', v'\eta')$
7: $p \leftarrow$ MwbMatching(M, M', w)
8: **if** $p = 0$, $e \notin E(G)$ or $e' \notin E(H)$ **then return** $-\infty$
9: **return** p

5.3 Correctness and Running Time

Now it has to be proven, that Algorithm 5.1 computes a block-and-bridge preserving maximum common subgraph of two partial 2-trees in polynomial time. First it is shown that the result of the algorithm is a block-and-bridge preserving common subgraph. Then it is argued that the common subgraph is also a maximum common subgraph. Last it is proven that this block-and-bridge preserving maximum common subgraph can be computed in polynomial time hence the block-and-bridge preserving maximum common subgraph problem in partial 2-trees is solvable in polynomial time. Later it is shown that these results, if applied to outerplanar graphs, have a better running time than any other algorithms known right now.

Lemma 5.3.1. *The common subgraph of two partial 2-trees computed by Algorithm 5.1 is a block-and-bridge preserving common subgraph.*

Proof. The only change in Procedure 5.2 is the addition in Line 9 where Procedure 5.3 is also added to the result. This is only relevant if both vertices which are mapped in this step are cutvertices as otherwise zero is returned in Line 2. If both vertices are cutvertices, all child nodes in the BC-trees are compared. At no point vertices contained in two blocks in the common subgraph can be contained in only one block in the corresponding input graph, because each block of G is only compared with exactly one block of H at each time. Of course, during the algorithm the blocks are compared to multiple blocks but only the maximum of those comparisons is taken into account. Therefore, restriction (BBP1) is always met.

Second, it is easy to see that only blocks of the same type are compared during the whole computation of the algorithm, see Lines 4 and 14 in Algorithm 5.1 and Lines 6 and 21 in Procedure 5.3. Thus, during the whole algorithm it is not possible that vertices which are not contained in a bridge node in one of the input graphs are contained in a bridge node in the common subgraph. Also as Procedure 5.2 computes a 2-connected common subgraph for two non-bridge nodes, it is not possible that a vertex which is not contained in a bridge node is contained in a bridge node in the common subgraph. Therefore restriction (BBP2) cannot be violated due to the design of the algorithm.

As both restrictions are satisfied at each point during the computation, the result of the algorithm is a block-and-bridge preserving common subgraph of both input graphs. □

It must be noted that initially both types of blocks are considered. Therefore not only non-bridge blocks are compared, but also bridges are compared and chosen as root for the BC-trees. This is necessary since a simple path is also a partial 2-tree, and the BC-tree of a simple path only consists of cutvertices and bridge nodes. Now it must be shown that the common subgraph is also a maximum common subgraph of the input graphs. This proof is straight forward, as simply all feasible combinations of nodes in the BC-trees are compared.

Lemma 5.3.2. *The block-and-bridge preserving common subgraph of two partial 2-trees computed by Algorithm 5.1 is a maximum common subgraph.*

Proof. In Chapter 4 it is proven, that Algorithm 4.1 computes the 2-connected maximum common subgraph of two 2-connected partial 2-trees. Therefore the algorithm presented in this chapter computes the block-and-bridge preserving maximum common subgraph of two 2-connected partial 2-trees per definition. As the graphs are not necessarily 2-connected[1], these cases must be considered, too.

If only one graph is not 2-connected, the common subgraph computed by the algorithm is a maximum common subgraph, as sMCS-Cut (Procedure 5.3) always returns 0 and thus

[1] A connected partial 2-tree is either simply connected or 2-connected as a partial 2-tree cannot be 3-connected per definition.

the algorithm behaves exactly the same as in Chapter 4 and the bridges of the input graph which is not 2-connected cannot be contained in the common subgraph.

If both graphs are not 2-connected, the algorithm computes a 2-connected maximum common subgraph for each combination of non-bridge blocks. It has been proven before that these are computed correct. Also during the combination, all 2-connected common subgraphs are considered. These results can be stored and then be reused for Procedure sMCS-Cut to compute the block-and-bridge preserving maximum common subgraph of the subgraph given by the child nodes of the C-node under the restriction that the vertices associated with the C-nodes are mapped accordingly. The size of a maximum common subgraph of two bridges is trivial, as it is always two. As all feasible combinations of these results are combined, the result must be a maximum common subgraph. It must be noted that not only the maximum common subgraphs of two non-bridge blocks are compared but all feasible common subgraphs. This is necessary as the size of a common subgraph may be increased if two cutvertices are mapped. □

Last it has to be shown that Algorithm 5.1 computes the block-and-bridge preserving maximum common subgraph of two partial 2-trees in polynomial time. This result is not trivial as all block-and-bridge preserving common subgraphs of non-bridge blocks of both input graphs need to be considered in the solution. Let $n = \max\{V(G), V(H)\}$ be the maximum number of vertices in a graph of the instance and $m = \max\{E(G), E(H)\}$ be the maximum number of edges in a graph of the instance.

Theorem 5.3.3. *The block-and-bridge preserving maximum common subgraph problem in partial 2-trees can be solved in time* $\mathcal{O}\left(n^6\right)$.

Proof. In Theorem 4.3.2[2] it is proven, that the size of a 2-connected maximum common subgraph of two 2-connected partial 2-trees can be computed in time $\mathcal{O}\left(n^6\right)$. The algorithm is transformed into a dynamic programming algorithm for the analysis. In the proof of the theorem is has been shown, that tables of size $\mathcal{O}(n^4)$ are sufficient to store the information about the split graphs, furthermore they store the size of the maximum common subgraph of two split graphs.

Once again it is assumed without loss of generality that for each smaller block split graph the block-and-bridge preserving maximum common subgraph has been computed whenever BBP-MCS-Series is called. These results can be used in time $\mathcal{O}(1)$. The new part of the algorithm is the call of BBP-MCS-Cut. To calculate the value of BBP-MCS-Cut, maximum weighted matching in a bipartite graph can be used. This problem is solvable in $\mathcal{O}(n^3)$ by the Hungarian method [21]. The bipartite graph and the weights for the edges are described in Equations 5.1 to 5.4. Due to the construction of BC-trees, the child-parent relation is given by the fact that all bride and non-bridge nodes containing

[2] See also [20].

the cutvertices (except for those associated with the current table) are children of those. Since the tables have size $\mathcal{O}(n^4)$ and the loop in Line 15 requires time $\mathcal{O}(n^2)$, the running time of Procedure BBP-MCS-Series is $\mathcal{O}(n^6)$.

BBP-MCS-Cut can be computed in time $\mathcal{O}(n^5)$ since the size of a block-and-bridge preserving maximum common subgraph of the smaller block split graphs has already been computed and can therefore be used. Again the maximum bipartite matching problem can be solved in $\mathcal{O}(n^3)$ since at most $\mathcal{O}(n^2)$ of these matching problems must be solved during the algorithm, the total running time of Procedure BBP-MCS-Cut is $\mathrm{O}(n^5)$.

As Procedure BBP-MCS-MatchEdges has not been changed, the total running time of the algorithm is $\mathcal{O}(n^5)$. Therefore, the running time of the algorithm is dominated by the running time of Procedure BBP-MCS-Series resulting in a total running time of $\mathcal{O}(n^6)$. □

The algorithm presented in Chapter 4 computes a 2-connected maximum common subgraph of two partial 2-trees in $\mathcal{O}\left(n^4\right)$ if both graphs are outerplanar. Although the computation of the 2-connected maximum common subgraphs of each combination of non-bridge blocks is the dominating factor regarding the running time, the consideration of all combinations of B-nodes whenever two cutvertices are mapped becomes the dominating factor if both graphs are outherplanar.

Theorem 5.3.4. *The block-and-bridge preserving maximum common subgraph problem for outherplanar graphs can be solved in* $\mathcal{O}\left(n^5\right)$.

Proof. The proof is similar to the proof of Theorem 4.3.4. Since the P-nodes in SP-trees of outerplanar graphs have degree two, the running time of the Procedures BBP-MCS-Series and BBP-MCS-MatchEdges is reduced to $\mathcal{O}(n^4)$ and $\mathcal{O}(n^3)$, respectively. There is no need to use the maximum bipartite matching, because the bipartite graphs are K_2's and there is only one S-node to continue the algorithm. The result may not be 2-connected but there still is only one S-node which can be considered.

Procedure BBP-MCS-Cut on the other hand considers all adjacent nodes in the BC-tree. The number of adjacent nodes in the BC-tree is not restricted if the graph is outerplanar and therefore still unbounded. Since this procedure has a total running time of $\mathcal{O}(n^5)$, this running time is the dominating time, if both graphs are outherplarnar. Therefore the total running time of the algorithm if both input graphs are outerplanar is $\mathcal{O}(n^5)$. □

5.4 Summary

In this chapter an algorithm which solves the block-and-bridge preserving maximum common subgraph problem for partial 2-trees problem is presented. It is shown, that the algorithm solves the problem in polynomial time. Therefore, the algorithm presented

in chapter 4 which solves the 2-connected maximum common subgraph problem for 2-connected partial 2-trees in polynomial time is extended. As, unlike 2-connected partial 2-trees, a partial 2-tree cannot be represented as a SP-tree decomposition, the BC-tree decomposition is used. Containing SP-tree decompositions for each 2-connected subgraph, the adjustments of the original algorithm are based on the changed data structure. Summarized algorithm 5.1 computes the size for each combination of B-nodes using the original algorithm to compute the size of the maximum common subgraph for the contained SP-tree decomposition in the B-nodes. If vertices pertinent to C-nodes are mapped, the size of the maximum common subgraph of the adjacent B-nodes may be added if appropriate.

In [26] an algorithm for the block-and-bridge preserving maximum common subgraph problem for outerplanar graphs[3] is presented which solves the problem in time $\mathcal{O}\left(n^7\right)$. The discrepancy is a result of changes which are required to compute an arbitrary maximum common subgraph of two outerplanar graphs in [4]. Nevertheless the algorithm presented in this chapter computes a block-and-bridge preserving maximum common subgraph of two outerplanar graphs in time $\mathcal{O}\left(n^5\right)$.

[3] In this paper, the edge induced maximum common subgraph problem is considered. The difference in the problem may explain the difference in the running time.

Chapter 6

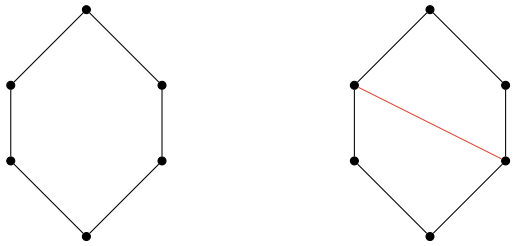
The Maximum Common Subgraph Problem in Partial 2-Trees with a Bounded Number of Chordless Cycles

The maximum common subgraph problems tackled in the previous chapters have restricted the common subgraph, either to be 2-connected or to be block-and-bridge preserving. Since restricting the solutions may not be possible in some applications, in this chapter an algorithm solving the maximum common subgraph problem in partial 2-trees with a bounded number of chordless cycles in 2-connected components is presented. To do so, first characteristics of cycles and their representation in BC-trees are presented, then the concept of *configurations* is introduced. The BC-trees and the configurations are then used to apply the approach, which is used to solve the block-and-bridge preserving maximum common subgraph problem, to this problem.

Before solving the problem in partial 2-trees with a bounded number of chordless cycles in 2-connected components the problem is solved in partial 2-trees with bounded number of cycles in 2-connected components.

6.1 Configurations, Cycles and SP-trees

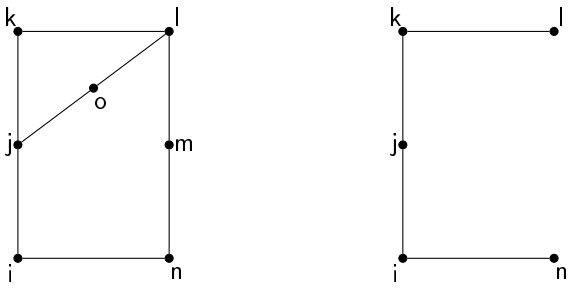
Let $G = (V, E)$ be a graph. A *configuration* is a set of vertices $C \subseteq V$ such that there are no adjacent vertices in the configuration. The subgraph $G' \subseteq G$ is called subgraph of G *with respect to the configuration* C if $G' = G \setminus C$. Let G be a graph and $R = (v_1, v_2, \ldots, v_n, v_1)$ a cycle in that graph. Then R has a chord if $(v_i, v_j) \in E(G)$ for any $i, j \in [n]$, such that $|i - j| \neq 1$ or $|i - j| \neq n - 2$. If there is no such edge, then the cycle is called chordless.



(a) (b)

Figure 6.1: A chordless cycle (a) and a cycle containing a chord highlighted in red (b).

The idea presented in Section 6.2 to compute the size of a maximum common subgraph in partial 2-trees with a bounded number of chordless cycles G and H is to compare all subgraphs with respect to a configuration of G to those of H. To do so, all feasible configurations need to be considered. It is easy to see that the number of configurations is $2^{|V(G)|}$ and $2^{|V(H)|}$, respectively. It must be noted, that not the number of chordless cycles in the whole graph must be bounded but the number of chordless cycles in the maximal 2-connected components of a graph must be bounded.



(a) (b)

Figure 6.2: A 2-connected graph (a) and the subgraph of the graph with respect to the configuration $\{m, o\}$ (b).

Configurations are used to compute all connected subgraphs of the input graphs. Since the maximum common subgraph needs to be connected and all of these subgraphs are considered, it is sufficient to compute the size of a block-and-bridge preserving maximum common subgraph.

As the number of configurations is exponential, there cannot be a polynomial time algorithm comparing all subgraphs with respect to the configurations. If the maximum common subgraph is not required to be connected, the problem is **NP**-complete[14]. Therefore, there is no need to consider subgraphs with respect to a configuration which are not connected. Hence, it is not necessary to consider configurations which are a separator of the graph.

Lemma 6.1.1. *Let G be a 2-connected partial 2-tree and T_G^{SP} the SP-tree of G. Also let n be the number of leaf S-nodes and closed P-nodes in the SP-tree, then a subgraph with respect to a configuration containing at least n vertices is not connected.*

Proof. Let G be a 2-connected partial 2-tree and T_G^{SP} the SP-tree of G. Also let $V_S^l \subseteq V_S(T_G^{\mathrm{SP}})$ be the leaf S-nodes of T_G^{SP}, $n = |V_S^l|$ and $C = \{c_1, c_2, \ldots, c_m\}$ with $n \leq m$ be a configuration. Assume that there is a subgraph $G' \subseteq G$ of G with respect to C which is connected.

Therefore, for any two vertices $u, v \in V(G)$ which are not in the configuration, there is a path $u \rightsquigarrow v$ in G'. Since there are at least n vertices contained in the configuration, there are two cases. Either at least one vertex in the skeleton graph of n S-node is contained in the configuration or second there is a S-node such that two vertices contained in the skeleton graph of that S-node are contained in the configuration.

In the first case, there are n vertices contained in the configuration, each a vertex in a different S-node. First, assume that T_G^{SP} does not contain any closed P-node. Assume that G' is still connected. Thus, for each pair of vertices $u, v \in V(G')$ there is a path $u \rightsquigarrow v$. Since each S-node represents a path between the vertices contained in the skeleton graph of the P-node, the S-node is pertinent to, there are two cases. First, if all vertices are contained in the leaf S-nodes, then there is a P-node λ such that there is no path between the vertices in the skeleton graph of λ. In the second case, if there is a vertex contained in the configuration which is not in the skeleton graph of a leaf S-node, then the case is analogously to the first one, since the vertex is contained in more than one path between two vertices contained in a P-node, while each vertex in a leaf S-node is contained in at least one path between any two vertices contained in the skeleton graph of a P-node.

If on the other hand, there are closed P-nodes in T_G^{SP}, then there is always a path between the vertices contained in the skeleton graphs of the closed P-nodes. Since there is an additional vertex in the configuration for each closed P-node, there must be at least one P-node such that the skeleton graphs of each S-node, pertinent to that P-node, contains a vertex which is also in the configuration. If that P-node is an open P-node, then the vertices in the skeleton graph of the P-node are not connected in G'. If that P-node is closed, then there is a vertex w in G' such that there is no path from w to the vertices in the skeleton graph of the P-node. Therefore G' cannot be connected.

In the second case, the subgraph G' cannot be connected since there are two vertices in the skeleton graph of one S-node. Since each S-node represents a path between the vertices in the skeleton graph of the P-node it is pertinent to, there is at least one vertex in the skeleton graph for which no path to none of the vertices in the skeleton graph of the P-node exists. □

Regarding a 2-connected partial 2-tree and its SP-tree, the number of vertices in a configurations can be bounded by the number of leaf S-nodes of the SP-tree. Each additional

vertex in the configuration would imply, that each subgraph with respect to the configuration would be disconnected. In addition to the number of vertices which are sufficient in a configuration also the position of the vertices in the graph can be restricted.

Let G be a partial 2-tree, $T_G^{\mathrm{BC},r}$ the BC-tree decomposition rooted at $r \in E(G)$ and T_Λ^{SP} the SP-tree of a non-bridge node Λ, rooted with respect to r[1]. A *required configuration* $C_\Lambda^{req} := \{s_1, s_2, \ldots, s_n\}$ for a non-bridge node Λ consists of n vertices where n is the number of S-nodes in T_Λ^{SP}. Each s_i represents a S-node λ_i and can either be a vertex in the skeleton graph of λ_i or $\emptyset$ for all $i \in [n]$. For each P-node λ_j and the adjacent S-nodes $\lambda_{j_1}, \ldots, \lambda_{j_s}$, there must be at least one s_k with $s_k = \emptyset$ for $k = j_i, \ldots, j_s$, except the vertex represented by all the vertices is contained in the skeleton graph of a P-node.

Therefore, for a non-bridge node Λ a required configuration $C^{req} = \{s_1, s_2, \ldots, s_n\}$ needs to satisfy the following conditions:

(RC1) $|C^{req}| = |V_S(T_\Lambda^{\mathrm{SP}})|$, where T_Λ^{SP} is the SP-tree of Λ,

(RC2) let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the S-nodes of T_Λ^{SP}, then $s_i \in \mathrm{skel}(\lambda_i) \cup \{\emptyset\}$ for all $i \in [n]$,

(RC3) let λ' be a P-node, then the value for at least one adjacent S-node must be $\emptyset$, except the values are all a vertex in the skeleton graph of that P-node.

It must be noted that a vertex which is contained in the skeleton graph of a P-node appears in more than one entry of the required configuration. Hence a required configuration is a multi-set. These conditions are true for non-bridge nodes of a BC-tree. A configuration for the whole graph is the union of a configuration for each non-bridge node. Let G be a graph. Each subgraph $G' \subseteq G$ of G with respect to a configuration C is called a subgraph of G *with respect to a required configuration* C if the configuration C is a required configuration.

Lemma 6.1.2. *Let G be a 2-connected partial 2-tree, then each subgraph of $G' \subseteq G$ with respect to a required configuration is connected.*

Proof. Assume that there is a subgraph with respect to a required configuration which is not connected since there is at most one vertex for each S-node which is also contained in the skeleton graph of that S-node in the configuration, (RC1), (RC2) and also there is no P-node for which all adjacent S-nodes have a vertex in their skeleton graph contained in the configuration, (RC3). Therefore, for each pair of vertices u, v in the subgraph G', there is a path $u \rightsquigarrow v$ since each S-node represents a path between the vertices contained in the skeleton graph of the pertinent P-node. At least one of these vertices is contained in G' for each P-node per definition.

If there is only one vertex w contained in the skeleton graph of a P-node which is contained in $V(G')$, then no other vertex contained in the skeleton graph of an adjacent

[1] For more informations regarding the rooting of Λ see Chapter 5.

S-node is contained in the configuration. Thus, for all vertices in the skeleton graphs of these S-nodes there is a path to w.

If both vertices w, w' contained in the skeleton graph of a P-node are contained in $V(G')$, then there is at least one S-node adjacent to the P-node such that no vertex contained in the skeleton graph of that S-node is contained in the configuration. Therefore, there is a path $w \rightsquigarrow w'$ in G' and also each vertex in the skeleton graphs of the S-nodes adjacent to the P-node must have a path to w or w'.

These two cases are true for all P-nodes so that G' must be connected. □

Because of that, required configurations do not result in disconnected graphs if the graph is 2-connected. If the graph is 2-connected, then a subgraph with respect to a required configuration can be disconnected but if the subgraph is disconnected, then the configuration must contain a cutvertex.

Lemma 6.1.3. *Let G be a partial 2-tree, then each subgraph of $G' \subseteq G$ with respect to a required configuration is connected if and only if the configuration does not contain any cutvertex.*

Proof. In Lemma 6.1.2 it has been proven, that the vertices in 2-connected components in G are contained in a connected component in G', so a subgraph with respect to a required configuration cannot disconnect at a 2-connected component and if there is no cutvertex contained in the configuration, the subgraph must be connected. Since the removal of a cutvertex results in a decomposition of the graph due to the definition of a cutvertex, a subgraph with respect to a required configuration cannot be connected if a cutvertex is contained in the configuration. □

Hence cutvertices contained in a required configuration result in subgraphs which are not connected. Although only connected subgraphs need to be considered, it is not possible to exclude cutvertices from the configurations since it is possible that two or more cutvertices are adjacent and one is not contained in a maximum common subgraph.

The required configurations are later used in the algorithm to compute feasible subgraphs which then are compared if they are common subgraphs. To do so in a polynomial time algorithm, the number of these configurations must be bounded polynomial by the size of the the input graphs.

Lemma 6.1.4. *Let G be a partial 2-tree and T_G^{BC} the BC-tree decomposition, $\Lambda \in V_{Bl}(T_G^{\mathrm{BC}})$ a non-bridge node in the BC-tree and T_Λ^{SP} the SP-tree of that node. Then the number of cycles in the skeleton graph S_Λ is $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ where n is the number of leaf S-nodes and closed P-nodes in T_Λ^{SP}.*

Proof. Proof by induction regarding the number n of leaf nodes in a SP-tree. It is easy to see that for the base case $n = 2$ there is exactly one cycle and for $n = 3$ there are three

cycles. The induction step: A graph which has a SP-tree containing $n+1$ leaf S-nodes contains $\frac{n(n+1)}{2}$ cycles under the assumption that a graph which has a SP-tree containing n leaf S-nodes contains $\frac{n(n-1)}{2}$ cycles. The additional S-node is an additional path between two vertices. This path can be contained in any existing cycle containing these vertices. The number of cycles containing these vertices is $n-1$. Since there are already $\frac{n(n-1)}{2}$ cycles this results in additional n cycles because the new path also results in a cycle containing the new path and the shortest path between the vertices.

The same induction can be done for closed P-nodes since each closed P-node represents a path between two vertices and a graph which has a SP-tree consisting of a closed P-node and two S-nodes has three cycles. □

Therefore by bounding the number of cycles in the graph, the number of S-nodes and closed P-nodes is bounded which bounds the number of required configurations.

Lemma 6.1.5. *Let n be the number of chordless cycles in a 2-connected partial 2-tree and m the number of S-nodes in the tree, then $n \geq m$.*

Proof. Let G be a 2-connected partial 2-tree and T^{SP} the SP-tree of G. If each P-node in T^{SP} is closed and G is outerplanar, then the number of chordless cycles is equal to the number of S-nodes in the SP-tree, since each chordless cycle can only consist of the vertices in the skeleton graph of one S-node and a P-node. Each cycle containing vertices, which are in more than one S-node and a P-node, contains at least one chord which is the edge incident to the vertices in the skeleton graph of the P-node.

In Lemma 6.1.4 the relation between S-nodes and the number of cycles is explained. So in both cases the addition of each S-node results in at least one new cycle. Therefore the number of chordless cycles is always greater than the number of S-nodes in the SP-tree.□

Therefore, by bounding the number of chordless cycles in the partial 2-tree, the number of S-nodes in the SP-trees is restricted. Since the number of required configurations is bounded exponentially by the number of S-nodes, this results in a restriction of the required configurations. Since a configuration applies to a 2-connected partial 2-tree, a partial 2-tree has a configuration for each non-bridge block. Since these configurations are independent during the computation, this does not result in an exponential number of subgraphs with respect to a configuration.

6.2 An Algorithm for the Maximum Common Subgraph Problem in Partial 2-Trees with a Bounded Number of Chordless Cycles

The main idea of the algorithm for two input graphs G and H is to compute the size of a maximum common subgraph of two subgraphs $G' \subseteq G$ and $H' \subseteq H$ with respect to

required configurations C and C', respectively. By restricting the considered graph to a subgraph respecting a required configuration the algorithm used to compute the block-and-bridge preserving maximum common subgraph presented in Chapter 5 can be adjusted and reused. To do so, it is important that the number of required configurations is bounded polynomial regarding the size of the input graphs. Since the number of configurations is bounded by the number of chordless cycles in the non-bridge nodes of the BC-tree and this number is restricted, the configurations can be computed in polynomial time. ReqConfigurations returns all required configurations for a non-bridge node.

Since the algorithm needs to respect the required configurations, vertices in the input graphs are deleted when they are contained in a configuration, and then added again if another configuration is considered. To do so, it is important to notice, that the used data structures namely BC-trees and SP-trees can both be updated in linear time, see [27] and [17], respectively.

Algorithm 6.1 still compares all non-bridge nodes and all bridge nodes separately. There is no need to use configurations in bridge nodes, since each vertex in a bridge is a cutvertex and there is no need to forbid these when the algorithm starts the construction of a common subgraph in a bridge node. The vertices in the skeleton graphs of the bridge nodes are either also contained in a non-bridge node where a configuration can be applied to them or are only contained in bridge nodes, where they can be ignored by not including them in a common subgraph.

With respect to the algorithm computing the size of a block-and-bridge preserving maximum common subgraph, Algorithm 6.1 is extended to work with configurations. The algorithm still starts at a node in the BC-tree and applies all required configurations for that node on the BC-tree. Update updates the BC-tree with respect to the configuration, therefore it *hides* the vertices contained in the configuration. If update is called again for a node which already has a configuration applied to it, the new configuration is used. So applying an empty configuration to all nodes in the BC-tree restores the BC-tree. Since only one configuration should be applied to a non-bridge node, all cutvertices in the BC-tree of G and H, respectively, are marked as *original* in Line 3. This label is later used to determine, whether a configuration must be applied to a non-bride node in the updated BC-tree, since during the computation all non-bride nodes are either considered directly in Algorithm 6.1 or during Procedure 6.4 when a cutvertex has been mapped. Since a configuration can result in new cutvertices and the no configuration should be applied to the non-bridge nodes adjacent to those, the label *original* is used.

6.2.1 Computation of MCS-BCC-Series and MCS-BCC-MatchEdges

Procedure 6.2 is not changed in its function regarding the computation of a block-and-bridge preserving maximum common subgraph since the idea of the algorithm is to compare

Algorithm 6.1 MCS-BCC(G, H)

Input: Two partial 2-trees G and H.

Output: Size of a maximum common subgraph of the partial 2-trees G and H.

1: $T_G^{\mathrm{BC}} \leftarrow \mathrm{BC}(G)$
2: $T_H^{\mathrm{BC}} \leftarrow \mathrm{BC}(H)$
3: mark all vertices associated with a cutvertex as *original*
4: $mcs \leftarrow 0$
5: **for all** $(\Xi, \Xi') \in V_{\mathrm{Bl}}(T_G^{\mathrm{BC}}) \times V_{\mathrm{Bl}}(T_H^{\mathrm{BC}})$ **do**
6: **for all** $(C, C') \in$ ReqConfigurations$(\Xi) \times$ ReqConfigurations(Ξ') **do**
7: Update$(T_G^{\mathrm{BC}}, \Xi, C)$; Update$(T_H^{\mathrm{BC}}, \Xi', C')$
8: **for all** $(\Lambda, \Lambda') \in V_{\mathrm{Bl}}(T_G^{\mathrm{BC}}) \times V_{\mathrm{Bl}}(T_H^{\mathrm{BC}})$ **do**
9: $T_\Lambda^{\mathrm{SP}} \leftarrow \mathrm{SP}(S_\Lambda^{\mathrm{BC}})$
10: $T_{\Lambda'}^{\mathrm{SP}} \leftarrow \mathrm{SP}(S_{\Lambda'}^{\mathrm{BC}})$
11: **for all** $(\lambda, \lambda') \in V_S(T_\Lambda^{\mathrm{SP}}) \times V_S(T_{\Lambda'}^{\mathrm{SP}})$ **do**
12: $r \leftarrow$ arbitrary edge $(u, v) \in E(S_\lambda) \cap E(G)$
13: root T_G^{BC} at r
14: **for all** *edges* $r' = (u', v') \in E(S_{\lambda'}) \cap E(H)$ **do**
15: root T_H^{BC} at r'
16: $p_1 \leftarrow$ sMCS-Series$(u, v, \lambda, u', v', \lambda') +$ sMCS-Cut(u, u')
17: $p_2 \leftarrow$ sMCS-Series$(u, v, \lambda, v', u'\lambda') +$ sMCS-Cut(u, u')
18: $mcs \leftarrow \max\{mcs, p_1, p_2\}$
19: **for all** $(\Lambda, \Lambda') \in V_{\mathrm{Br}}(T_G^{\mathrm{BC}}) \times V_{\mathrm{Br}}(T_H^{\mathrm{BC}})$ **do**
20: $r = (u, v) \leftarrow E(S_\Lambda^{\mathrm{BC}})$; root T_G^{BC} at r
21: $r' = (u', v') \leftarrow E(S_{\Lambda'}^{\mathrm{BC}})$; root T_H^{BC} at r'
22: $p_1 \leftarrow$ sMCS-Cut$(u, u') +$ sMCS-Cut(v, v')
23: $p_2 \leftarrow$ sMCS-Cut$(u, v') +$ sMCS-Cut(v, u')
24: $mcs \leftarrow \max\{mcs, p_1, p_2\}$
25: **return** mcs $+2$

all subgraphs with respect to a required configuration and computing a block-and-bridge preserving maximum common subgraph of these subgraphs. The procedure is only listed to give a full overview of the algorithm. The same is true for Procedure 6.3 which is also only listed to give a full overview of Algorithm 6.1.

6.2.2 Computation of MCS-BCC-Cut

Procedure 6.4 is extended to work with configurations. To do so, first it has to be determined whether the cutvertices u and u' are original. In Lines 6 and 9 the configurations for the adjacent child non-bridge nodes are computed only if the associated cutvertex is

Procedure 6.2 MCS-BCC-SERIES$(u, v, \lambda, u', v'\lambda')$

Input: Base vertices u, v of G and u', v' of H and S-nodes $\lambda \in S(T_G)$ resp. $\lambda' \in S(T_H)$.

Output: Size of a maximum common subgraph of the split graphs G^r_{uv} and $H^r_{u'v'}$ under the condition that u is mapped to u'.

1: $e = (v, w) \leftarrow$ NEXT(v, λ)
2: $e' = (v', w') \leftarrow$ NEXT(v', λ')
3: $mcs \leftarrow 0$
4: **if** $e = \mathrm{ref}(\lambda)$ **then return** MSC-SERIES$(u, v, \mathrm{pS}(\lambda), u', v', \lambda')$
5: **if** $e' = \mathrm{ref}(\lambda')$ **then return** MSC-SERIES$(u, v, \lambda, u', v', \mathrm{pS}(\lambda'))$
6: **if** $w = u$ and $w' = u'$ **then return** MATCHEDGE$(e, \lambda, e', \lambda')$
7: **if** $w = u$ xor $w' = u'$ **then**
8: **return** $-\infty$
9: $mcs \leftarrow$ MATCHEDGE(e, e') + sMCS-SERIES$(u, w, \lambda, u', w', \lambda') + 1$ + sMCS-CUT(w, w')
10: **if** $e \notin E(G)$ or $e' \notin E(H)$ **then**
11: **if** $e \in E(G)$ **then** $M \leftarrow \{\lambda\}$
12: **else** $M \leftarrow \mathrm{sC}(e)$
13: **if** $e' \in E(H)$ **then** $M' \leftarrow \{\lambda'\}$
14: **else** $M' \leftarrow \mathrm{sC}(e')$
15: **for all** $(\eta, \eta') \in M \times M'$ **do**
16: $p \leftarrow$ MCS-SERIES$(u, w, \eta, u', w', \eta')$ + sMCS-CUT(w, w')
17: $mcs \leftarrow \max\{mcs, p\}$
18: **return** mcs

an original cutvertex. Otherwise there is only the empty configuration not changing the current BC-tree. Then starting in Line 12 all non-bridge nodes are compared.

If at least one of the cutvertices is an original cutvertex, the required configurations are applied to the BC-tree. Otherwise only the non-bridge node is considered. Then in Line 15 all adjacent child B-nodes are considered again. This is necessary since a configuration has been applied before and therefore the block split graphs $T^{\mathrm{BC},r}_u$ and $T^{\mathrm{BC},r}_{u'}$ may have been changed. The rest of the procedure is the same as Procedure 5.3 since at this point, all adjacent child B-nodes must be compared. It must be noticed, that in Lines 24 and 25 only edges incident to the cutvertex are considered. This is necessary since not the maximum common subgraph of the split graphs $T^{\mathrm{BC},r}_u$ and $T^{\mathrm{BC},r}_{u'}$ must be returned by the procedure, but the maximum common subgraph in which u is mapped to u'.

Procedure 6.3 MCS-BCC-MatchEdges$(e, \lambda, e', \lambda')$

Input: Edges $e = (u, v) \in E(S_\lambda)$ and $e' = (u', v') \in E(S'_\lambda)$

Output: Size of a maximum common subgraph of the split graphs G^r_{uv} and $H^r_{u'v'}$ under the condition that u is mapped to u'.

1: **if** $e \in E(G)$ xor $e' \in E(H)$ **then return** $-\infty$
2: **if** e or e' is a *real* edge in S_λ resp. $S_{\lambda'}$ **then return** 0
3: $M \leftarrow \mathrm{cS}(e)$
4: $M' \leftarrow \mathrm{cS}(e')$
5: **for all** $f = (\eta, \eta') \in M \times M$ **do**
6: $\quad w(f) \leftarrow$ MCS-Series$(u, v, \eta, u', v'\eta')$
7: $p \leftarrow$ MwbMatching(M, M', w)
8: **if** $p = 0$, $e \notin E(G)$ or $e' \notin E(H)$ **then return** $-\infty$
9: **return** p

6.3 Correctness and Running Time

Lemma 6.3.1. *The result of Algorithm 6.1 is the size of a maximum common subgraph of the input graphs.*

Proof. The algorithm compares all subgraphs respecting required configurations and computes a block-and-bridge preserving maximum common subgraph of these subgraphs. Since the subgraphs with respect to a configuration are connected partial 2-trees, it must be proven that all connected subgraphs are considered. As each combination of vertices not resulting in a disconnected graph is used as required configuration, all those graphs are considered and therefore the algorithm computes the size of the maximum common subgraph. □

The algorithm computes the size of a maximum common subgraph of two partial 2-trees with a bounded number of chordless cycles. Now it has to be shown that the algorithm computes the size in polynomial time regarding the size of the input graphs. It is not easy to see that the algorithm computes the size of the maximum common subgraph in polynomial time since all connected subgraphs of the input graphs are considered. Since the number of required configurations is bounded for each non-bridge node of the BC-tree, it is possible to transform the algorithm in a dynamic programming algorithm and reuse the computed solutions for smaller block split graphs. Due to this, the number of configurations for each non-bridge node can be bounded by the number of chordless cycles in that node.

Let $n = \max\{|V(G)|, |V(H)|\}$ and k be the maximum number of chordless cycles in a 2-connected component of either G or H.

Theorem 6.3.2. *The maximum common subgraph problem in partial 2-trees with a bounded number of chordless cycles can be solved in time* $\mathcal{O}(n^{6+2k})$.

Proof. The algorithm computes the size of block-and-bridge preserving maximum common subgraphs for subgraphs respecting a required configuration. To do so, the size is computed for all required configurations. Therefore, the total running time of the algorithm is the running time of the algorithms solving the block-and-bridge preserving maximum common subgraph problem times the number of required configurations.

The number of required configurations for a non-bridge node is bounded by $\mathcal{O}(n^k)$. It is assumed that the common subgraphs for each smaller block split graph have already been computed and can therefore be used in this computation. Since for each configuration the BC-tree is adjusted, which can be done in linear time and also the SP-tree is adjusted, which can be done in linear time, too, the algorithm computes a block-and-bridge preserving maximum common subgraph for the subgraphs with respect to the required configuration. This can be done in $\mathcal{O}(n^6)$. Due to the fact, that the results for smaller block split graphs have already been computed, the required configurations for non-bridge nodes in these split graphs do not have to be considered during this computation. Therefore, the total running time of the algorithm is $\mathcal{O}(n^{6+2k})$. □

6.4 Summary

In this section an algorithm solving the maximum common subgraph problem in partial 2-trees is presented. The algorithm uses the concept of configurations which are sets of vertices which cannot appear in a common subgraph with respect to that configuration. Since the number of configuration is exponential by the size of the graph, required configurations are introduced. It is shown that the number of required configurations is bounded polynomial by the size of the graph if the number of chordless cycles in the graph is bounded. Hence, the algorithm computes the size of a maximum common subgraph in polynomial time if the number of chordless cycles is bounded.

These required configurations are then used together with the approach used in Chapter 5 to solve the problem, since a common subgraph of two subgraphs respecting required configurations is block-and-bridge preserving for these graphs. The common subgraph may not be block-and-bridge preserving with respect to the input graphs.

Since the running time of the algorithm is $\mathcal{O}(n^{6+2k})$, is does not compute the maximum common subgraph of two outerplanar graphs in polynomial time as the number of chordless cycles in outerplanar graphs is bounded by the size of the graph.

Procedure 6.4 MCS-BCC-CUT(u, u')

Input: Two cutvertices $u \in V(G)$, $u' \in V(H)$.

Output: Size of a preserving maximum common subgraph of the block split graphs $T^{\mathrm{BC},r}_{G,u}$ and $T^{\mathrm{BC},r}_{H,u'}$ under the condition that u is mapped to u'.

1: **if** u or v is not pertinent to a node C-node **then**
2: **return** 0
3: $N \leftarrow \mathrm{cB}(u)$; $N' \leftarrow \mathrm{cB}(u')$
4: $C^{req_u} \leftarrow \{\emptyset\}$; $C^{req_{u'}} \leftarrow \{\emptyset\}$
5: $mcs \leftarrow 0$
6: **if** u is an original cutvertex **then**
7: **for all** $\Xi \in V_{\mathrm{Bl}}(N)$ **do**
8: $C^{req_u} \leftarrow C^{req_u} \cup$ REQCONFIGURATIONS(Ξ)
9: **if** u' is an original cutvertex **then**
10: **for all** $\Xi' \in V_{\mathrm{Bl}}(N')$ **do**
11: $C^{req_{u'}} \leftarrow C^{req_{u'}} \cup$ REQCONFIGURATIONS(Ξ')
12: **for all** $(\Xi, \Xi') \in V_{\mathrm{Bl}}(N) \times V_{\mathrm{Bl}}(N)$ **do**
13: **for all** $(C, C') \in C^{req_u} \times C^{req_{u'}}$ **do**
14: UPDATE(T^{BC}_G, C); UPDATE(T^{BC}_H, C')
15: $M \leftarrow \mathrm{cB}(u)$; $M' \leftarrow \mathrm{cB}(u')$
16: **for all** $f = (\Lambda, \Lambda') \in V_{\mathrm{Bl}}(M) \times V_{\mathrm{Bl}}(M)$ **do**
17: $T^{\mathrm{SP}}_\Lambda \leftarrow \mathrm{SP}(S^{\mathrm{BC}}_\Lambda)$; $T^{\mathrm{SP}}_{\Lambda'} \leftarrow \mathrm{SP}(S^{\mathrm{BC}}_{\Lambda'})$
18: **if** $\exists \lambda \in V_P(T^{\mathrm{SP}}_\Lambda)$ such that $u \in V(S_\lambda)$ **then** $N \leftarrow \mathrm{cS_P}(\lambda)$
19: **else** $N \leftarrow \{\lambda\}$
20: **if** $\exists \lambda' \in V_P(T^{\mathrm{SP}}_{\Lambda'})$ such that $u' \in V(S_{\lambda'})$ **then** $N' \leftarrow \mathrm{cS_P}(\lambda')$
21: **else** $N' \leftarrow \{\lambda'\}$
22: $mcs_{\mathrm{tmp}} \leftarrow 0$
23: **for all** $(\lambda, \lambda') \in N \times N'$ **do**
24: $(\mathbf{u}, v) \leftarrow$ arbitrary edge in $E(S_\lambda) \cap E(G)$
25: **for all** edges $(\mathbf{u}', v') \in E(S_{\lambda'}) \cap E(H)$ **do**
26: $mcs_{\mathrm{tmp}} \leftarrow \max\{mcs_{\mathrm{tmp}}, \text{sMCS-SERIES}(u, v, \lambda, u', v', \lambda')\}$
27: $w(f) \leftarrow mcs_{\mathrm{tmp}}$
28: **for all** $f = (\eta, \eta') \in V_{\mathrm{Br}}(M) \times V_{\mathrm{Br}}(M)$ **do**
29: $(u, v) \leftarrow$ NEXT(u, λ)
30: $(u', v') \leftarrow$ NEXT(u', λ')
31: $w(f) \leftarrow$ sMCS-CUT$(v, v') + 1$
32: $mcs \leftarrow \max\{mcs, \text{MWBMATCHING}(M, M', w)\}$
33: **return** mcs

Chapter 7

Conclusion and Outlook

In this thesis the maximum common subgraph problem in partial 2-trees has been considered. The problem is known to be **NP**-hard if there are no restrictions regarding the input graphs or the maximum common subgraph. Hence, different restrictions have been considered to analyze the complexity.

First in Chapter 3 it is shown that the maximum common subgraph problem in partial 2-trees is **NP**-hard, even if all vertices except for one in each of the input graphs have bounded degree and the degree is bounded by three. This is done by proving that there is a polynomial-time reduction from the maximum common subgraph problem in partial 2-trees to the numerical matching with target sums problem. Therefore the maximum common subgraph problem in partial 2-trees with bounded degree of three for all but one vertex is **NP**-hard[1].

Then in Chapter 4 an approach for the 2-connected maximum common subgraph problem in 2-connected partial 2-trees is discussed. The algorithm has first been introduced in [20] and is also explained in this thesis. The algorithm uses the characteristic of 2-connected graphs that each vertex in such a graph must be contained in at least one cycle. The algorithm also uses SP-trees as unique data structure to represent 2-connected partial 2-trees. All this results in an algorithm solving the 2-connected maximum common subgraph problem in 2-connected partial 2-trees in $\mathcal{O}(n^6)$.

In Chapter 5 the block-and-bridge preserving maximum common subgraph problem in partial 2-trees is introduced. Unlike the 2-connected maximum common subgraph problem, this problem does not restrict the input graphs. BC-trees are used as unique data structure in addition to the graphs. The algorithm solving the block-and-bridge preserving maximum common subgraph problem uses the same characteristic as the algorithm before. Each vertex contained in a non-bridge block in an input graph must be contained in a non-bridge node in a common subgraph and therefore must be contained in at least one cycle. This problem can be solved in $\mathcal{O}(n^6)$.

[1] The decision version of the problem is **NP**-complete.

The block-and-bridge preserving maximum common subgraph problem[2] in outerplanar graphs is already discussed in [26], where an algorithm solving the problem in $\mathcal{O}(n^7)$ is developed. The algorithm presented in Chapter 5 can solve the block-and-bridge preserving maximum common subgraph problem in outerplanar graphs in $\mathcal{O}(n^5)$.

The problems discussed before have restricted the common subgraph first to be 2-connected and then to be block-and-bridge preserving. In Chapter 6 the maximum common subgraph problem in partial 2-trees with a bounded number of chordless cycles is discussed. In this chapter, the concept of configurations and subgraphs respecting these configurations is introduced. It is shown that the number of configurations which need to be considered is bounded by the number of chordless cycles. Also it is shown that the approach of the previous chapter can be reused to solve the problem in $\mathcal{O}(n^{6+2k})$, where k is the maximum number of chordless cycles in a block.

In this thesis the maximum common subgraph problem in partial 2-trees has been tackled with different restrictions. It has been proven, that there are efficient algorithms for various restrictions. It has also been shown that some restrictions are not sufficient to obtain an efficient algorithm unless $\mathbf{P} = \mathbf{NP}$.

Even though the maximum common subgraph problem in partial 2-trees has not been fully solved. An interesting open question is whether the maximum common subgraph problem in partial 2-trees is solvable in polynomial time if all vertices in both input graphs have bounded degree. During the creation of this thesis, this problem has been tackled but still there is no clear result.

On the one hand, to the best knowledge of the author, there is only one problem which is solvable in polynomial time in outerplanar graphs, but is **NP**-complete in partial 2-trees. The problem is called the edge-disjoint paths problem, see [24], and asks whether there are n pairwise edge-disjoint paths p_i connecting $2n$ given vertices s_i and t_i for all $i \in [n]$.

On the other hand the maximum common subgraph problem in partial 2-trees of bounded degree seems to be too simple for a polynomial-time reduction from any **NP**-complete problem, since it is not possible to pairwise compare an unbounded number of elements with just two graphs if the degree of the graphs is bounded. Also the considered problem must either be **NP**-complete in the strong sense or must have an input which is not a number. Therefore the number of possible candidates which can be used in a reduction is even smaller.

Also the maximum common subgraph problem in other graph classes is still not analyzed. It is known that the maximum common subgraph problem in outerplanar graphs can be solved in polynomial time, see [4]. In this thesis some restrictions of partial 2-trees have been considered. It is also known, that the problem is **NP**-complete in partial 11-trees with degree greater than five, see [3]. It may be interesting to decide whether the maximum common subgraph problem is **NP**-complete in partial 3-trees of bounded degree.

[2] The edge induced maximum common subgraph problem.

Variables, Functions and Abbreviations

$G = (V, E)$:	Graphs
$V, V(G)$:	Set of vertices, Set of vertices of G
$E, E(G)$:	Set of edges, Set of edges of G
$\deg(v)$:	Degree of v
λ_G :	Graph labeling of G
$G[W]$:	Subgraph induced by W in G
$u \rightsquigarrow v$:	Path between u and v
MwbMatching :	Maximum weighted bipartite matching
K_n :	Complete graph with n vertices
$K_{n.m}$:	Complete bipartite graph with bipartitions containing n and m vertices
$G \cong H$:	G is isomorphic to H
$H \preccurlyeq G$:	H is subgraph isomorphic to G
TD(G) :	Tree decomposition of a graph G
$\mathcal{C}(G)$:	The connected components of G
X_i :	Bag in tree decomposition
tw(G) :	Tree width of the graph G
NTD(G) :	Normalized tree decomposition of a graph G
$K_2^{s,t}$:	K_2 where one vertex is denoted by s and the other by t
SPQR(G) :	SPQR-tree of the graph G
SP(G) :	SP-tree of G
S_λ :	Skeleton graph of the node λ in a SP-tree
$V_S(T)$:	Set of S-nodes of a SP-tree T
$V_P(T)$:	Set of P-nodes of a SP-tree T

$\mathrm{ref}(\lambda)$:	Reference of λ
BC(G) :	Block graph of G which is a BC-tree if G is a partial 2-tree
$(X, Y, s, \vec{b})$:	Instance of the numerical matching with target sums problem
$G_s^{X,Y}$:	Graph representing the values of the elements in X and Y of an instance of the numerical matching with target sums problem
$H_{s,\vec{b}}^{X,Y}$:	Graph representing the values in $\vec{b}$ of an instance of the numerical matching with target sums problem
Σ_s :	$\sum_{i=1}^{n} (s(x_i) + s(y_i))$ for an instance $(X, Y, s, \vec{b})$
$\Sigma_{\vec{b}}$:	$\sum_{i=1}^{n} b_i$ for an instance $(X, Y, s, \vec{b})$
$[n]$:	$\{1, \ldots, n\}$
$\overline{G_{uv}^r}$:	Split graph rooted at r, split at u, v containing the root
G_{uv}^r :	Split graph rooted at r, split at u, v not containing the root
$\overline{T_u^{\mathrm{BC},r}}$:	Block split graph rooted at r, split at u containing the root
$T_u^{\mathrm{BC},r}$:	Block split graph rooted at r, split at u not containing the root
C^{req} :	Required configurations

List of Figures

List of Algorithms

Index

Bibliography

[1] Subtree isomorphism in $\mathcal{O}(n^{\frac{5}{2}})$, author=Matula, David W, journal=Annals of Discrete Mathematics, volume=2, pages=91–106, year=1978, publisher=Elsevier.

[2] Tatsuya Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 76(9):1488–1493, 1993.

[3] Tatsuya Akutsu and Takeyuki Tamura. On the complexity of the maximum common subgraph problem for partial k-trees of bounded degree. In *Algorithms and Computation*, pages 146–155. Springer, 2012.

[4] Tatsuya Akutsu and Takeyuki Tamura. A polynomial-time algorithm for computing the maximum common subgraph of outerplanar graphs of bounded degree. In *Mathematical Foundations of Computer Science 2012*, pages 76–87. Springer, 2012.

[5] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[6] Hans L. Bodlaender. A tourist guide through treewidth. *Technical report RUU-CS*, 92, 1993.

[7] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[8] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1 – 45, 1998.

[9] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255–259, 1998.

[10] Markus Chimani and Petr Hliněnỳ. A tighter insertion-based approximation of the crossing number. In *Automata, Languages and Programming*, pages 122–134. Springer, 2011.

[11] Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 436–441. IEEE, 1989.

[12] Richard J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.

[13] Michael R Garey and David S Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[14] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[15] Arvind Gupta and Naomi Nishimura. *Sequential and parallel algorithms for embedding problems on classes of partial k-trees.* Springer, 1994.

[16] Arvind Gupta and Naomi Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoretical Computer Science*, 164(1):287–298, 1996.

[17] Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In *Graph Drawing*, pages 77–90. Springer, 2001.

[18] Tamás Horváth and Jan Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411(31-33):2784 – 2797, 2010.

[19] R Bruce King and Dennis H Rouvray. *Graph theory and topology in chemistry: a collection of papers presented at an international conference held at the University of Georgia, Athens, Georgia, USA, 16-20 March 1987*, volume 51. Elsevier Science Ltd, 1987.

[20] Nils Kriege and Petra Mutzel. Finding maximum common subgraphs in series-parallel graphs (unpublished).

[21] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[22] Andrzej Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63(3):295–302, 1989.

[23] Takao Nishizeki and Norishige Chiba. *Planar graphs: Theory and algorithms.* Access Online via Elsevier, 1988.

[24] Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is np-complete for series–parallel graphs. *Discrete Applied Mathematics*, 115(1):177–186, 2001.

[25] John W Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16(7):521–533, 2002.

[26] Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time metric for outerplanar graphs. In *Workshop on Mining and Learing with Graphs*. Citeseer, 2007.

[27] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[28] Jotph A Wald and Charles J Colbourn. Steiner trees, partial 2–trees, and minimum ifi networks. *Networks*, 13(2):159–167, 1983.

[29] Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

[30] Atsuko Yamaguchi, Kiyoko F Aoki, and Hiroshi Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Information processing letters*, 92(2):57–63, 2004.

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift