The Open University of Israel
Department of Mathematics and Computer Science

INDEX CODING AND CLIQUE COVER ON PLANAR GRAPHS

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science

The Open University of Israel
Department of Mathematics and Computer Science

By
Yossi Berliner

Prepared under the supervision of Prof. Michael Langberg

August 2013

Contents

Chapter 1

Abstract

We study the Index Coding problem with side information graphs which are k - *outerplanar*. For general side information graphs, solving the Index Coding problem implies a solution to the general (non-multicast) Network Coding problem — a central open problem in the field of network communication. For outerplanar side information graphs, we show that the Index Coding problem can be solved efficiently, and characterize its solution in terms of the *clique cover* size of the information graph at hand. The clique cover problem on planar graphs is known to be NP-complete. For $k - outerplanar$ graphs we show that the clique cover problem can be solved efficiently.

Part of this thesis also appeared as: Y. Berliner and M. Langberg, *Index coding with outerplanar side information.* In proceedings of IEEE International Symposium on Information Theory (ISIT), 2011, 869-873.

Chapter 2

Introduction

The *Index Coding* problem is a fundamental non-multicast network coding problem. The problem has a very clean and elegant structure yet it captures many important aspects of the more general network coding problem. The Index Coding problem was introduced by Birk and Kol [4] in the context of data dissemination to clients with local caches. An instance of the Index Coding problem includes a sender node r, a set $C = \{c_1, \ldots, c_n\}$ of wireless clients and a set $P = \{p_1, p_2, \ldots, p_n\}$ of n independent messages that belong to some alphabet Σ. The message p_i needs to be delivered to client c_i. Each client c_i is assumed to hold certain *side information* $\Gamma_i \subseteq P$. It is common to specify the side information by a graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ in which vertex i corresponds to client c_i and edge $(i, j) \in E$ iff $j \in \Gamma_i$.

In each *round* of communication the sender can transmit a single symbol of Σ (i.e., a single message). We assume that all symbols transmitted by the sender are received by all clients without error. The j'th round of communication is specified by an encoding function $g_j : \Sigma^n \to \Sigma$. The objective is to find a set of encoding functions $\Phi = \{g_i\}_{i=1}^{\ell}$ that will allow client c_i to decode the messages p_i it requires while minimizing the number of transmissions $\ell = |\Phi|$. Client $c_i \in C$ can decode packet p_i if there exists a decoding function $\gamma_i : \Sigma^\ell \times \Sigma^{|\Gamma_i|} \to \Sigma$ that allows c_i to obtain p_i from the ℓ characters transmitted by the sender and the $|\Gamma_i|$ characters of side information. The minimum value of ℓ is defined to be the *round complexity* of the index coding instance at hand (in cases in which the alphabet Σ is specified we denote the round complexity as ℓ_Σ).

In this work we focus on information graphs G which are *undirected* (namely $i \in \Gamma_j$ iff $j \in \Gamma_i$), and on the case in which the encoding functions are linear (or actually *scalar* linear by common terminology). In this case we also take Σ to be a finite field. We note that one can extend the definition of the Index Coding problem to directed side information graphs, to encoding functions which are not scalar linear but rather *vector linear* (via time sharing) or non-linear, and to clients c_i which require not a single message but multiple ones. We touch on these extensions in the conclusion of this work.

There exist beautiful connections between combinatorial properties of the undirected side information graph G and the optimal solution to the corresponding Index Coding problem. In [3] it is shown that the problem of finding (scalar) linear solutions to the Index Coding problem is equivalent to the problem of minimizing the rank of a certain matrix with "don't care" entries. The latter problem, referred to as the `MinRank` problem (denoted by $\mathtt{MR}_\Sigma$), has been investigated by Haemers and Peeters [12, 19] in which it is shown that the optimal solution value of $\mathtt{MR}_\Sigma(G)$ is "sandwiched" between the maximum sized independent set in G (denoted by $\alpha(G)$) and the minimum sized clique cover of G (denoted by $\mathtt{CC}(G)$). Here, an independent set in G is a subset of vertices that do not share any edges, a clique in G is a subset of vertices for which every pair share an edge, and a clique cover of G is a collection of cliques in G such that every vertex in G appears in at least a single clique in the collection. Namely, for an field Σ and an instance to the index coding problem defined by an undirected graph G, the scalar linear round complexity $\ell_\Sigma^{(lin)}$ is exactly $\mathtt{MR}_\Sigma(G)$ which in turn satisfies $\mathtt{CC}(G) \geq \mathtt{MR}_\Sigma(G) \geq \alpha(G)$.

The authors of [3] study the Index Coding problem in the setting of (scalar) linear encoding functions. Lubetzky

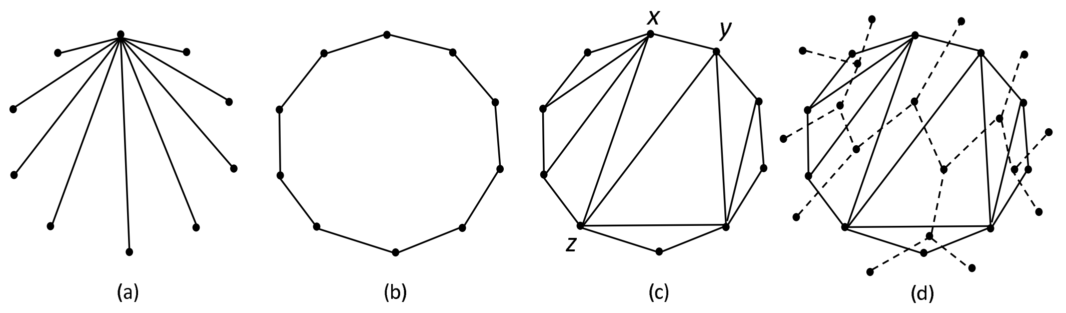

Figure 2.1: Examples of outerplanar graphs. (a) The “star” graph. (b) The “ring” graph. (c) A general outerplanar graph. The vertices x, y and z illustrate the proof of Claim 2.2.2. (d) The outerplanar graph G presented in (c) with its corresponding tree representation $\bar{G}$ (with dotted edges).

et al. [18] show that non-linear codes can significantly outperform linear codes for certain families of problem instances. El Rouayheb et al. [9] and Effros et al. [**?**] show that any instance of the more general network coding problem can be efficiently reduced to an instance of the Index Coding problem. This reduction implies that efficiently finding optimal solutions for the Index coding problem implies an efficient solution to the general (non-multicast) network coding problem — the latter being a central open problem in network communication. In the scalar linear setting, this implies that it is NP-hard to solve the index coding problem (via the hardness results of [15] on scalar linear network coding). Such hardness results were also shown independently in [19]. The hardness of finding approximate solutions for the Index Coding problem has been studied in [14].

2.1 Our contribution

As it is NP-hard to efficiently find scalar linear solutions to the Index Coding problem for arbitrary side information graphs G, in this work we address the following natural question: *On which families of side information graphs can Index Coding be solved efficiently?*

Our work takes a modest step in better understanding this question. Namely, in our study, we investigate “simple nature” side information graphs G that are so-called *k - outerplanar*. A graph G is said to be outerplaner, if it has an embedding in the plane in which (a) all vertices of G lie on the outer (unbounded) face of the embedding, and (b) representing edges of G by straight lines between their corresponding vertices, no two edges of G intersect. A graph G is said to be k-outerplanar, if it has a planar embedding satisfying (b) above in which k consecutive removals of the outer face of the embedding result in the empty graph. Some examples of outerplanar graphs are given in Figure 2.1.

We show that on outerplaner side informations graphs G, the Index Coding problem can be solved efficiently and can be characterized by the clique cover number of G. The first contribution of this work can be summarized by the following theorem.

Theorem 2.1.1. *Let Σ be any finite field. The optimal scalar linear solution to the Index Coding problem over Σ with outerplanar side information graphs G can be found efficiently (i.e., in time which is polynomial in $|G|$). In this case, the scalar linear round complexity $\ell_{\Sigma}^{(lin)} = \mathtt{MR}_{\Sigma}(G)$ of the optimal scalar linear index coding solution is equal to the clique cover size $\mathtt{CC}(G)$ of G.*

Another natural question that motivated our work addresses the relationship between the index coding round complexity $\ell_{\Sigma}^{lin} = \mathtt{MR}_{\Sigma}(G)$ and its combinatorial upper and lower bounds — the clique cover size $\mathtt{CC}(G)$ and the independent set size $\alpha(G)$. It is not hard to find graphs G for which $\ell_{\Sigma}^{lin}(G)$ is strictly greater than $\alpha(G)$, for example the five cycle C_5 satisfies $\ell_{\Sigma}^{lin}(C_5) = 3$ while $\alpha(C_5) = 2$. However, it is significantly more difficult to find graphs G for which $\mathtt{CC}(G)$ is strictly greater than $\ell_{\Sigma}^{lin}(G)$.

Roughly speaking, the index coding solutions that correspond to clique covers of G are extremely simple in nature. Specifically, for each clique C in the cover one defines an encoding function g which equals the sum of messages p_i corresponding the vertices in the clique C. Hence asking whether for a certain G it holds that $\mathtt{CC}(G) > \ell_{\Sigma}^{lin}(G)$ is equivalent to asking whether for a certain side information graph G there is an index coding solution which is better than the trivial one. Such graphs (of rather complicated nature) are known to exist, e.g., [12, 19]. However, in an attempt to construct simpler families of graphs G for which $\mathtt{CC}(G) > \ell_{\Sigma}^{lin}(G)$, it is natural to ask: *For which family of side information graphs G is it the case that $\ell_{\Sigma}^{lin}(G) = \mathtt{CC}(G)$?* This work takes a small step in better understanding this question.

The second contribution of this work is regarding the clique cover problem. Recall that given an undirected graph $G = (V, E)$, a clique cover of G of size k is a disjoint partition $\{V_1, \dots, V_k\}$ of V such that each V_i induces a clique. In the clique cover problem ones objective is to find the minimum sized $\mathtt{CC}(G)$. A k-coloring of G, is a disjoint partition $\{V_1, \dots, V_k\}$ of V such that each V_i induces an independent set. Let $\chi(G)$ denote the minimum sized coloring of G. It holds that $\mathtt{CC}(G) = \chi(\bar{G})$, where $\bar{G} = (V, \bar{E})$ is the complement of G in which $e \in E$ iff $e \notin \bar{E}$. For any constant $\varepsilon > 0$, the clique cover size $\mathtt{CC}(G)$ is NP-hard to approximate beyond a factor of $n^{1-\varepsilon}$ [20].

We address the clique cover problem when restricted to instances which are planar or, to be more precise, k-outerplanar. The clique cover problem on planar graphs is known to be NP-complete [8, 13]. In this work we study the clique cover problem on k-outerplanar graphs and present an algorithm that runs in time $n^{O(k)}$ for its solution. The second contribution of this work can be summarized by the following theorem.

Theorem 2.1.2. *Let k be any constant. The optimal solution to the clique cover problem of a k-outerplanar graph G can be found efficiently (i.e., in time which is polynomial in $|G|$). Moreover, for any $\varepsilon > 0$, there is a $(1+\varepsilon)$ approximation for the clique cover problem on planar graphs with running time $n^{O(1/\varepsilon)}$.*

2.2 Proof techniques

It is well known that many NP-hard graph problems become easier to approximate on planar graphs. One general technique for coping with planarity is via the vertex separators of Lipton and Tarjan [16, 17]. In this approach, given G, one finds a subset C of vertices of size $O(\sqrt{n})$ whose removal separates G into two parts of approximately the same size. Recursing on each set, until one remains with connected components of constant size $1/\varepsilon$, yields a tree of separators of total size $O(\sqrt{\varepsilon}n)$. Solving the clique cover problem on each connected component exhaustively yields an efficient approximation within an additive error of $O(\sqrt{\varepsilon}n)$ (the separator size). As there are no cliques of size 5 or more in planar instances, the optimal solution is at least of size $n/4$. This implies that the additive approximation above yields a multiplicative approximation ratio of $1 + O(\sqrt{\varepsilon})$. The total running time for $\varepsilon > 0$ is bounded by $O(n \max(\log n, 2^{\log(1/\varepsilon)/\varepsilon}))$. Unfortunately, as noted in [5], the approach of [16, 17] is impractical because of large constant factors. For example, to achieve an approximation ratio of just 2, the base case requires exhaustive solutions of graphs of up to $2^{2^{400}}$ vertices.

Another technique widely used in the approximate solution of combinatorial problems on planar instances, that overcomes the large constant factors involved in [16, 17], is that of Baker [2]. Baker's approach is based on a decomposition of the given graph into a set of disjoint k-outerplanar graphs, obtained by removing at most $n/(k+1)$ vertices from G. Given such a decomposition, Baker solves the problem under study on each k-outerplanar instance in the decomposition. Finally, to obtain a solution for G, Baker *glues* the solutions found on the k outerplanar instances of the decomposition with a solution on the $n/(k+1)$ vertices removed. She then argues that the final solution is a $k/(k+1)$ approximation for maximization problems (and similarly a $(k+1)/k$

for minimization problems) to the optimal one (the details of this process depend on the problem under study). The heart of Baker's approach is in the optimal solution to the problem under study on k outerplanar instances (obtained by the process mentioned above). To this end, Baker designs a certain partition of k outerplanar graphs into a tree structure that supports dynamic programming. Based on this paradigm, Baker [2] presents efficient algorithms for approximating several NP complete problems on planar graphs. These include for example the maximum independent set (and minimum vertex cover), minimum dominating set, and minimum edge-dominating set problems. The total running time to obtain a $(1+\varepsilon)$ approximation is typically bounded by $O(2^{1/\varepsilon}n)$. We note that, to the best of our knowledge, the clique cover problem or the `MinRank` problem have not been addressed in the context of [2].

The algorithmic paradigm of Baker [2] lends itself naturally to the problems studied in this Thesis. Namely, in the first part of this Thesis, we tie the clique cover size of a given outerplanar graph to its `MinRank` (or equivalently, its scalar linear index coding round complexity $\ell_{\Sigma}^{(lin)}$). As mentioned above, the `MinRank` of any graph is at most its clique cover. In this part we show that for outerplanar graphs the clique cover size and `MinRank` value are actually equal. Our proof follows the dynamic nature of Baker's algorithm. We start by showing how to apply Baker's paradigm to the problems of computing the `MinRank` and the clique cover of a given graph G. This proves that both `MinRank` and clique cover problems on outerplanar graphs can be solved efficiently. Applying the work of [2] to these problems involves a delicate analysis that does not follow directly from [2]. To tie the clique cover number with the `MinRank` of G, we show that throughout the execution of Baker's algorithm - no matter what intermediate objective is being considered (the clique cover size or the `MinRank`) the value returned in the intermediate step is identical. This implies that the `MinRank` of G equals its clique cover in the outerplanar scenario.

In the second part of this Thesis, we extend the ideas of Baker to show that one can solve clique cover on k-outerplanar graphs efficiently. As mentioned, the dynamic programming approach of Baker does not adapt directly to the setting of clique cover. Accordingly, in this part, our extension involves a delicate analysis of the dynamic programming resulting from an enhanced tree structure when compared to that of [2]. Our running time on k outerplanar graphs is $n^{O(k)}$.

We note that our results on the clique cover of k-outerplanar graphs imply an approximation algorithm for clique cover on planar graphs. Indeed, if one could solve clique cover exactly on k-outerplanar instances - then a trivial gluing procedure via [2] in which each vertex is considered a clique in the cover, yields an additive approximation of $n/(k+1)$. Using the fact that the minimum clique cover size of planar graphs is linear - one obtains a $1+4/(k+1)$ multiplicative approximation.

Additional works related to this Thesis include generalizations of Bakers approach for families of graphs with certain *local* properties. In [10, 7, 6] the notion of *bounded local tree-width* was addressed, leading to $(1+\varepsilon)$ approximations on several additional problems on such graphs including maximum triangle matching, maximum H-matching, maximum tile salvage, and minimum color sum. In [11] the notion of *locally tree-decomposable* graphs was considered, and efficient algorithms for deciding any property expressible in first-order logic in such graphs were developed. In cases, these algorithms imply $(1+\varepsilon)$ approximations on corresponding families of graphs. To the best of our knowledge, the techniques mentioned above do not apply to our study of clique cover or `MinRank`.

Chapter 3

Preliminaries

3.1 Definitions and notation

Definition 3.1.1 ($\mathtt{MR}_\Sigma(G)$)**.** *Let Σ be a finite field. We say that a matrix $A = a_{ij}$ fits an undirected graph G if for all i and j: $a_{ii} = 1$, and $a_{ij} = 0$ whenever (i, j) is not an edge of G.* $\mathtt{MR}_\Sigma(G) \equiv \min\{\textbf{rank}_\Sigma(A) \mid A \text{ fits } G\}$

Definition 3.1.2 ($\mathtt{CC}(G)$)**.** *A clique in an undirected graph G is a subset of vertices for which every pair share an edge. A clique cover of G is a collection of cliques in G such that every vertex in G appears in at least a single clique in the collection. $\mathtt{CC}(G)$ is the size of the minimum clique cover in G.*

For a graph G and a node v, let $G+\{v\}$ denote the graph G with a new isolated node v, and let $G-\{v\}$ denote the graph G in which node $v \in G$ is removed from G. Similarly, we define $G + B$ and $G - B$ for a subgraph B including nodes and edges. For two graphs G_1 and G_2 let $G_1 \cup G_2$ be the graph obtained by taking the union of the corresponding node sets and edge sets.

3.2 Properties of Clique-Cover

Property 3.2.1. *Given a graph G and a node v,* $\mathtt{CC}(G) \leq \mathtt{CC}(G + \{v\}) \leq \mathtt{CC}(G) + 1$ *(Adding a node to a graph can increase the clique-cover by at most 1).*

Proof. The clique which contains the single node v can be added to the original clique cover of G. Thus resulting in new clique cover with size $\mathtt{CC}(G) + 1$. For the other direction, any clique cover of $G + \{v\}$ induces a clique cover of the same size in G. □

Property 3.2.2. *Given a graph G which contains the nodes $\{x, y\}$, and doesn't contain the edge $e = (x, y)$,* $\mathtt{CC}(G) \geq \mathtt{CC}(G + e) \geq \mathtt{CC}(G) - 1$ *(Adding an edge to a graph can decrease the clique-cover by at most 1).*

Proof. Consider a graph G, which includes nodes x, y and does not include the edge (x, y). Suppose that adding an edge (x, y) to the graph can decrease the clique-cover by at least 2, thus $\mathtt{CC}(G+(x, y)) \leq \mathtt{CC}(G)-2$. Removing the node y (and as a result the edge (x, y)) from $(G + (x, y))$ causes a new graph with clique-cover that equals at most $(\mathtt{CC}(G) - 2)$. Thus, adding the node y again to that graph and the edges of G adjacent to y, can increase the clique-cover at most by 1 (according to Property 3.2.2), thus $\mathtt{CC}(G) \leq \mathtt{CC}(G) - 1$, a contradiction. The first inequality follows easily by the definition of a clique cover. □

Property 3.2.3. *Let G_1, G_2 be 2 graphs with one common node x (such that there are no edges between $G_1 - \{x\}$ and $G_2 - \{x\}$). If the following values are known: $\mathtt{CC}(G_1)$, $\mathtt{CC}(G_2)$, $\mathtt{CC}(G_1 - \{x\})$, $\mathtt{CC}(G_2 - \{x\})$, then the clique cover of the union of G_1 and G_2 is known and equal to the following:*

- *If* $\texttt{CC}(G_1) = \texttt{CC}(G_1 - \{x\})$ *or* $\texttt{CC}(G_2) = \texttt{CC}(G_2 - \{x\})$*, then* $\texttt{CC}(G_1 \cup G_2) = \texttt{CC}(G_1 - \{x\}) + \texttt{CC}(G_2 - \{x\})$*.*
- *If* $\texttt{CC}(G_1) = \texttt{CC}(G_1 - \{x\}) + 1$ *and* $\texttt{CC}(G_2) = \texttt{CC}(G_2 - \{x\}) + 1$*, then* $\texttt{CC}(G_1 \cup G_2) = \texttt{CC}(G_1 - \{x\}) + \texttt{CC}(G_2 - \{x\}) + 1$*.*

Proof. First consider the minimum size clique cover of the union of the disjoint graphs $(G_1 - \{x\})$ and $(G_2 - \{x\})$. It holds that the minimum size clique cover of the disjoint union equals the sum of the (disjoint) minimum size clique covers (this follows directly from the definition of a clique cover). Now, we will add the node x to the graph. By the properties above, the clique-cover size of the new graph can be equal to the clique-cover size of the union or it may increase by 1. We study two cases:

- If $\texttt{CC}(G_1) = \texttt{CC}(G_1 - \{x\})$ or $\texttt{CC}(G_2) = \texttt{CC}(G_2 - \{x\})$, then adding x to the union of the disjoint graphs doesn't increase the clique cover because x belongs to one of the cliques in the optimal cover of G_1 or G_2.
- If $\texttt{CC}(G_1) = \texttt{CC}(G_1 - \{x\}) + 1$ and $\texttt{CC}(G_2) = \texttt{CC}(G_2 - \{x\}) + 1$, we show that adding x to the union of the disjoint graphs increases the clique cover size by 1 (in this case $\{x\}$ will be the clique added to the clique cover). Assume by contradiction that $\texttt{CC}(G_1 \cup G_2) = \texttt{CC}(G_1 - \{x\}) + \texttt{CC}(G_2 - \{x\})$. Consider the minimum clique cover on $G_1 \cup G_2$. The clique in this cover that contains x can't contain additional nodes from both the graphs G_1 and G_2 as they are disjoint graphs (excluding x). Assume w.o.l.g. that x belongs to a clique with nodes from G_1. Thus, in the clique cover, the number of cliques that cover G_1 is exactly $\texttt{CC}(G_1 \cup G_2) - \texttt{CC}(G_2 - \{x\}) = \texttt{CC}(G_1 - \{x\})$. This implies that $\texttt{CC}(G_1 - \{x\}) = \texttt{CC}(G_1)$, in contradiction to the assumption.

□

Property 3.2.4. *Let G_1, G_2 be 2 graphs with k common nodes $B = \{b_1, \dots, b_k\}$ such that there are no edges between $(G_1 \setminus B)$ and $(G_2 \setminus B)$. If the following values are known: $\texttt{CC}(G_1)$, $\texttt{CC}(G_2)$, $\texttt{CC}(G_1 - B')$, $\texttt{CC}(G_2 - B')$ for each subset $B' \subseteq B$, then the clique cover of the union of G_1 and G_2 is known and equal to the following:* $\texttt{CC}(G_1 \cup G_2) = \min\{(\texttt{CC}(G_1 \setminus B') + \texttt{CC}(G_2 \setminus (B \setminus B'))) \mid B' \subseteq B\}$

Notice that this property is a generalization of Property 3.2.3

Proof. Let $G_1' = G_1 \setminus B$, and $G_2' = G_2 \setminus B$. Let $C(G)$ be the minimum sized set of cliques in G such that every node in G appears in at least a single clique in the set. Notice that $Size(C(G)) = \texttt{CC}(G)$. Consider the graph $G_1 \cup G_2$. We can separate $C(G_1 \cup G_2)$ to 5 groups of cliques:

1. C_1 - Cliques that contain nodes from G_1' only
2. C_2 - Cliques that contain nodes from G_2' only
3. C_3 - cliques that contain nodes from B only
4. C_4 - cliques that contain nodes from $G_1' \cup B$
5. C_5 - cliques that contain only nodes from $G_2' \cup B$

Let B' be the group of nodes from B that are contained in $(C_3 \cup C_5)$. That is, $B' = (C_3 \cup C_5) \cap B$. It follows that all the nodes from $(B \setminus B')$ are contained in C_4. Thus, $C(G_1 \setminus B') = C_1 \cup C_4$. Moreover, $C(G_2 \setminus (B \setminus B')) = C_2 \cup C_3 \cup C_5$. It follows that $C(G_1 \cup G_2) = C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5$ = $C(G_1 \setminus B') \cup C(G_2 \setminus (B \setminus B'))$. This implies that $\texttt{CC}(G_1 \cup G_2) \geq \min\{(\texttt{CC}(G_1 \setminus B') + \texttt{CC}(G_2 \setminus (B \setminus B'))) \mid B' \subseteq B\}$. The other direction follows by the definition of $\texttt{CC}(G_1 \cup G_2)$.

□

3.3 Properties of `MinRank`

Property 3.3.1. *Given a graph G and a node v, $\texttt{MR}_{\Sigma}(G) \leq \texttt{MR}_{\Sigma}(G+\{v\}) \leq \texttt{MR}_{\Sigma}(G)+1$ (Adding a node to a graph can increase the* `MinRank` *by at most 1).*

Proof. Consider the matrix M that realizes $\texttt{MR}_{\Sigma}(G)$. Adding a new row and column corresponding to the new vertex v, in which all new entries are of value 0 except the new diagonal entry $m_{v,v}$ that is of value 1, we get that $\texttt{MR}_{\Sigma}(G+\{v\}) \leq \texttt{MR}_{\Sigma}(G)+1$. For the lower bound notice that any matrix that fits $G+\{v\}$ has a corresponding restriction (of lower or equal rank) that also fits G. $\square$

Property 3.3.2. *Given a graph G which contains the nodes $\{x, y\}$, and doesn't contain the edge $e = (x, y)$, $\texttt{MR}_{\Sigma}(G) \geq \texttt{MR}_{\Sigma}(G+e) \geq \texttt{MR}_{\Sigma}(G)-1$ (Adding an edge to a graph can decrease the* `MinRank` *by at most 1).*

Proof. Consider a graph G, which includes nodes x, y and does not include the edge (x, y). Suppose that adding an edge (x, y) to the graph can decrease the `MinRank` by at least 2. Namely, that $\texttt{MR}_{\Sigma}(G+(x,y)) \leq \texttt{MR}_{\Sigma}(G)-2$. Removing the node y (and as a result the edge (x, y)) from $(G+(x,y))$ causes a new graph with `MinRank` that equals at most $(\texttt{MR}_{\Sigma}(G)-2)$ (as one can take any matrix that fits G and turn it into one that fits $G-\{y\}$ by removing the row and column that correspond to y). Thus, adding the node y again to that graph and the edges of G adjacent to y, can increase the `MinRank` at most by 1 (according to Property 3.3.1). We conclude that $\texttt{MR}_{\Sigma}(G) \leq \texttt{MR}_{\Sigma}(G)-1$, a contradiction. For the upper bound in the assertion notice that any matrix that fits G also fits $G+e$ by definition. $\square$

Property 3.3.3. *Let G_1, G_2 be 2 graphs with one common node x (such that there are no edges between $G_1-\{x\}$ and $G_2-\{x\}$). If the following values are known: $\texttt{MR}_{\Sigma}(G_1)$, $\texttt{MR}_{\Sigma}(G_2)$, $\texttt{MR}_{\Sigma}(G_1-\{x\})$, $\texttt{MR}_{\Sigma}(G_2-\{x\})$, then the* `MinRank` *of the union of G_1 and G_2 is known and equal to the following:*

- *If $\texttt{MR}_{\Sigma}(G_1) = \texttt{MR}_{\Sigma}(G_1-\{x\})$ or $\texttt{MR}_{\Sigma}(G_2) = \texttt{MR}_{\Sigma}(G_2-\{x\})$, then $\texttt{MR}_{\Sigma}(G_1 \cup G_2) = \texttt{MR}_{\Sigma}(G_1-\{x\}) + \texttt{MR}_{\Sigma}(G_2-\{x\})$.*
- *If $\texttt{MR}_{\Sigma}(G_1) = \texttt{MR}_{\Sigma}(G_1-\{x\})+1$ and $\texttt{MR}_{\Sigma}(G_2) = \texttt{MR}_{\Sigma}(G_2-\{x\})+1$, then $\texttt{MR}_{\Sigma}(G_1 \cup G_2) = \texttt{MR}_{\Sigma}(G_1-\{x\}) + \texttt{MR}_{\Sigma}(G_2-\{x\})+1$.*

Proof. Let M be a matrix which fits $G_1 \cup G_2$. Assume M has the following structure:

$$\left(\begin{array}{c||c|c|c}
 & G'_1 & G'_2 & x \\
\hline\hline
G'_1 & G'_{1l} & 0 & \\
\hline
G'_2 & 0 & G'_{2r} & \\
\hline
x & x_l & 0 & 1
\end{array}\right)$$

In the above description we use the following notation. G'_1 represents the subgraph $(G_1-\{x\})$. G'_2 represents the subgraph $(G_2-\{x\})$. The row and column labels appear to the left or above the double line, while the matrix entries appear to the right and below the double line. Each row (and column) of M corresponds to a vertex v in $G_1 \cup G_2$. An entry m_{uv} in M corresponds to the vertices u and v in $G_1 \cup G_2$. For the row vector $(v_1, \ldots, v_n)$ corresponding to v we denote its entries corresponding to G'_1 by v_{left} or v_l, its entries corresponding to G'_2 by v_{right} or v_r and its entries corresponding to a vertex w by v_w. For example, for the vertex x we have that $x_x = 1$. Also for a vertex $v \in G'_2$ we have that $v_l = 0$ as there are no edges between G'_2 and G'_1. The submatrix G'_{1l} (G'_{2r})

consists of the vectors v_l (v_r) for $v \in G_1'$ ($v \in G_2'$). Finally, for any vertex v we abuse notation and refer to the row vector corresponding to v by the same notation: v. The same goes for subsets of vertices A.

Define M_1 and M_2 as matrices that fit G_1 and G_2 correspondingly, and have the minimum **rank** among all such matrices. By definition, $\texttt{MR}_\Sigma(G_1) = \textbf{rank}(M_1)$ and $\texttt{MR}_\Sigma(G_2) = \textbf{rank}(M_2)$. That is, M_1 and M_2 can be expressed as follows:

$$\begin{pmatrix} \begin{array}{c||c|c} & G_1' & x \\ \hline\hline G_1' & H_1' & \\ \hline x & & 1 \end{array} \end{pmatrix}$$

$$\begin{pmatrix} \begin{array}{c||c|c} & G_2' & x \\ \hline\hline G_2' & H_2' & \\ \hline x & & 1 \end{array} \end{pmatrix}$$

The submatrix H_1' (H_2') consists of the vectors corresponding to $v \in G_1'$ ($v \in G_2'$). We consider two cases:

- If $\texttt{MR}_\Sigma(G_1) = \texttt{MR}_\Sigma(G_1 - \{x\}) = \textbf{rank}(M_1)$, we first claim that the rows corresponding to G_1' in M_1 span a vector space of dimension $\textbf{rank}(M_1)$, and the vector corresponding to x in M_1 is spanned by the rows corresponding to G_1'. Denote the latter vector by x_1. The above follows since the submatrix H_1' fits G_1' and has rank at most that of the the rows corresponding to G_1'. However, there is no matrix that fits G_1' of rank less than $\texttt{MR}_\Sigma(G_1 - \{x\})$.

  Now consider the matrix M above. In what follows we will suggest values for the entries of M that will yield rank equal to $\texttt{MR}_\Sigma(G_1 - \{x\}) + \texttt{MR}_\Sigma(G_2 - \{x\})$. Namely, we set G_{1l}' to be equal to H_1', we set the x'th entry of the rows of G_1' in M to be equal to their corresponding entries in M_1, we set G_{2r}' to be H_2', the (row) vector x_{lx} (the entries of the vector x in M corresponding to vertices in $G_1' \cup \{x\} = G_1$) to be the vector x_1, and x_r to be 0. As we show above, the vector x_{lx} is spanned by the rows of M corresponding to G_1'. Thus x is spanned by the rows corresponding to $G_1' \cup G_2'$. We conclude that the resulting matrix M has rank $\textbf{rank}(M_1) + \textbf{rank}(H_2') = \texttt{MR}_\Sigma(G_1') + \texttt{MR}_\Sigma(G_2')$. This shows that $\texttt{MR}_\Sigma(G) \leq \texttt{MR}_\Sigma(G_1') + \texttt{MR}_\Sigma(G_2')$. To obtain equality, notice that as G_1' and G_2' are disjoint graphs, it is not hard to verify (from the definition of `MinRank`) that these rows span a vector space of dimension at least $\texttt{MR}_\Sigma(G_1') + \texttt{MR}_\Sigma(G_2')$. The same proof can be shown for the case that $\texttt{MR}_\Sigma(G_2) = \texttt{MR}_\Sigma(G_2 - \{x\})$.

- $\texttt{MR}_\Sigma(G_1) = \texttt{MR}_\Sigma(G_1 - \{x\}) + 1$ and $\texttt{MR}_\Sigma(G_2) = \texttt{MR}_\Sigma(G_2 - \{x\}) + 1$. Assume by contradiction that the optimal matrix M satisfies $\textbf{rank}(M) = \texttt{MR}_\Sigma(G_1 \cup G_2) = \texttt{MR}_\Sigma(G_1 - \{x\}) + \texttt{MR}_\Sigma(G_2 - \{x\}) = \texttt{MR}_\Sigma(G_1') + \texttt{MR}_\Sigma(G_2')$. This implies that $x \in \textbf{span}(G_1' \cup G_2')$, as otherwise either $\textbf{rank}(G_{1l}') < \texttt{MR}_\Sigma(G_1')$ or $\textbf{rank}(G_{2r}') < \texttt{MR}_\Sigma(G_2')$ which is a contradiction to the facts that G_{1l}' fits G_1' and G_{2r}' fits G_2'.

  Consider the rows of M that participate in the linear combination that yields the vector x. If these rows are included in G_1', then it follows that the subvector x_{lx} (the left coordinates of the vector x including the coordinate corresponding to x) is in $\textbf{span}(G_1')$ which implies that $\texttt{MR}_\Sigma(G_1) = \texttt{MR}_\Sigma(G_1 - \{x\})$, a contradiction. To see the contradiction, construct M_1 by setting the rows corresponding to G_1' in M_1 to be equal to the corresponding entries in the rows corresponding to G_1' in M, and the vector corresponding to x in M_1 to be equal to x_{lx}. A similar analysis can be done for the case that the rows of M that participate in the linear combination that yields the vector x are included in G_2'.

If the the rows of M that participate in the linear combination that yields the vector x combine vectors from G'_1 and G'_2, one may consider the partial linear combination from G'_1 and G'_2 separately. Let x_1 be the linear combination resulting from the rows in G'_1 and x_2 be the linear combination corresponding to the rows in G'_2. Namely $x = x_1 + x_2$. Let the coordinate in x_i corresponding to the vertex x be a_i. As the entry $x_x = 1$, we have that $a_1 + a_2 = 1$. Thus, it cannot be the case that both a_1 and a_2 are 0. Assume w.l.o.g. that $a_1 \neq 0$. Also assume w.l.o.g. that $a_1 = 1$ (otherwise the entries of M_1 to be constructed shortly can be scaled accordingly). Now construct M_1 by setting the rows corresponding to G'_1 in M_1 to be equal to the corresponding entries in the rows corresponding to G'_1 in M, and the vector corresponding to x in M_1 to be equal to the corresponding coordinates in a revised version of x_1 in which the entry corresponding to the vertex x in x_1 is changed to a_1. It is not hard to verify that the modified version of x_1 is spanned by the rows corresponding to G'_1 in M_1 and thus $\mathtt{MR}_\Sigma(G_1) = \mathtt{MR}_\Sigma(G_1 - \{x\})$, a contradiction.

$\square$

Part I

Index coding with outerplanar side information

Chapter 1

Overview of Baker's algorithm for outerplanar graphs

In this part of the Thesis we show that on outerplaner side informations graphs G, the Index Coding problem can be solved efficiently and can be characterized by the clique cover number of G. We start by showing how to apply Baker's paradigm to the problems of computing the `MinRank` and the clique cover of a given graph G. We then tie the clique cover number with the `MinRank` of G. The conclusion of this part is the proof of Theorem 2.1.1.

Our work is based on Baker's algorithmic paradigm [2]. In what follows we give a brief (and rough) overview of the main ideas that govern the algorithm of [2]. We will identify and isolate the major points that need to be addressed in order to apply the paradigm at hand to the case of the `MinRank` and clique cover problems. We start by presenting Baker's algorithm in general, and then proceed to the major points we address in this work (which are discussed in greater detail). We will not give a full and comprehensive description of Baker's algorithm and refer the reader to [2] for proofs and details we omit.

Given an outerplanar graph G, the algorithm of Baker [2] has two major steps. In the first step, a tree representation $\bar{G}$ of G is constructed. Every node in $\bar{G}$ corresponds to a subgraph of G, where the root of $\bar{G}$ corresponds to G itself, each leaf in $\bar{G}$ corresponds to an edge, and internal nodes in $\bar{G}$ correspond to the subgraph of G induced by their children in $\bar{G}$. The construction of $\bar{G}$ from G is very simple in nature, and the tree $\bar{G}$ tightly resembles the standard notion of the *dual* to an (outer)planar graph. See Figure 2.1(d) (of the Introduction) and Figure 1.1 (in this section of the thesis).

In slightly more detail: $\bar{G}$ is constructed as follows (in this presentation we suppose there are no cutpoints in G, i.e., a vertex whose deletion disconnects the graph). Place a vertex in each interior face and on each exterior edge of G, and draw an edge from each vertex representing a face f to each vertex representing either an adjacent face (i.e., a face sharing an edge with f) or an exterior edge of f. (This tree is closely related to the dual of the graph; however, the dual would lack vertices for exterior edges and would have an additional vertex for the exterior face.) An example (taken from [2]) is shown in Figure 1.1.

The planar embedding induces a cyclic ordering on the edges of each vertex in the tree. Choosing a face vertex v as the root and choosing which child of v is to be its leftmost child determine the parent and ordering of children for every other vertex of $\bar{G}$. Label the vertices of $\bar{G}$ recursively, as follows: Label each leaf of the tree with the oriented exterior edge it represents. Label each face vertex with the first and last nodes in its children's labels. If a face vertex is labeled (x, y), the leaves of its subtree represent a directed walk of exterior edges in a counterclockwise direction from x to y. For the root, $x = y$ and the directed walk covers all the exterior edges. For any other face vertex v, $x \neq y$, and (x, y) is an interior edge shared by the face represented by v and the face represented by its parent in the tree. We define $G(x, y)$ to be the subgraph corresponding to the subtree of $\bar{G}$ rooted at tree-vertex (x, y). Namely, $G(x, y)$ contains all edges corresponding to leaves in it's subtree with the addition of the edge (x, y).

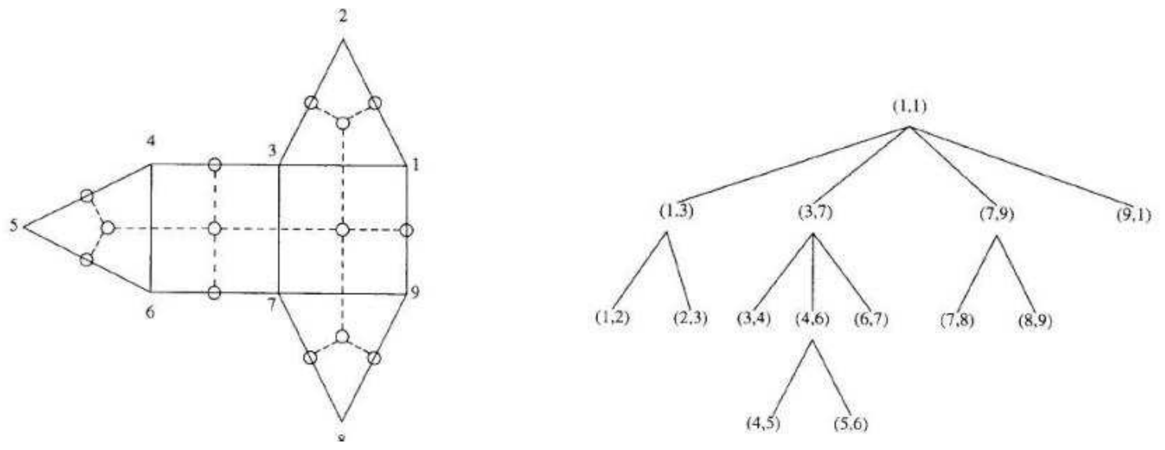

Figure 1.1: An outerplanar graph and its corresponding tree representation. Taken from [2].

For example, in Figure 1.1, the leaves of the node labeled $(3, 7)$ represent a walk along nodes $3, 4, 5, 6, 7$. The leaves of the root $(1, 1)$ represent a counterclockwise walk around the exterior edges beginning and ending at node 1. The vertex labeled $(1, 3)$ represents the face containing nodes $1 - 3$, its parent represents the face containing nodes $1, 3, 7, 9$, and $(1, 3)$ is the interior edge shared by these faces. We refer the interested reader to Baker's original work [2] for a full presentation of the tree structure $\bar{G}$.

Once the tree $\bar{G}$ is given, the objective in [2] is to dynamically compute the objective function Obj at hand (e.g., MinRank) in a *bottom up* manner from the leaves to the root. Namely, for each vertex (x, y) of $\bar{G}$, based on the algorithm of [2], we will define a table for this vertex which contains 4 values:

- The solution to the objective function at hand for the subgraph $G(x, y)$ including the nodes x and y.
- The solution to the objective function at hand for the subgraph $G(x, y)$ including the node x and not y.
- The solution to the objective function at hand for the subgraph $G(x, y)$ including the node y and not x.
- The solution to the objective function at hand for the subgraph $G(x, y)$ excluding the nodes x and y

The algorithm of [2] to compute the table of a tree vertex v is given in Figure 1.2. We now show how to compute the table for each vertex in our case, in which Obj is either the MinRank or clique cover. First, for a leaf vertex (x, y), it holds that $G(x, y) = (x, y)$, and it's table is defined as follows:

- The solution to the objective function for a leaf vertex (x, y) including the nodes x and y is 1.
- The solution to the objective function for a leaf vertex (x, y) including the node x and not y is 1.
- The solution to the objective function for a leaf vertex (x, y) including the node y and not x is 1.
- The solution to the objective function for a leaf vertex (x, y) excluding the nodes x and y is 0.

It's easy to see that when Obj is either MinRank or clique cover it holds that Obj(x, y) equals 1. The table for every other vertex will be computed recursively by merging the tables of its children according to the algorithm of [2] given in Figure 1.2.

There are two major operations that need to be addressed in the above procedure. The **merge** operation takes as input the intermediate table T and the table of a tree node u and returns a "merged" table of the two. More specifically, let v be a vertex which represents a face vertex (x, y) with the following children: (x, a), (a, b), ...

```
Procedure table($v$)
if $v$ is a level 1 leaf corresponding to an edge with label $(x, y)$
  then
    return a table representing the edge $(x, y)$;
  else   * $v$ is a face vertex*
    begin
    T = table($u$), where $u$ is the leftmost child of $v$;
    for each other child $u$ of $v$ from left to right
      T = merge (T, table($u$));
    return (adjust(T));
    end
```

Figure 1.2: The algorithm `table` from [2].

(i, z), (z, w), $\ldots$ (j, y). Assume that we are considering the child (z, w) of (x, y) in $\bar{G}$. Let $G(x, z)$ be the subgraph of G that includes all the edges that are in the union of the subgraphs $G(x, a) \cup G(a, b) \cup ... \cup G(i, z)$. The current table T includes information for the subgraph $G(x, z)$. Namely (by induction) assume the current table T has a value for each *bit pair* representing x and z. By the term "bit pair" we refer to each possibility for the existence of x and z in the subgraph $G(x, z)$, meaning that the table T has 4 solutions to the objective function at hand, each of a subgraph $G(x, z)$ (solution for the subgraph containing x and z, solution for the subgraph containing x and not containing z, solution for the subgraph containing z and not containing x, solution for the subgraph without both x and z.) The child u has label (z, w) for some w, and table(u) has a value for each bit pair representing z and w. The goal of procedure merge is to construct an updated table T with a solution of the objective function on $G(x, w) \cup G(w, z)$ for every bit pair representing x and w.

The **adjust** operation takes as input a table T consisting of the merge of all the children of v, and returns a table corresponding to v. Specifically, let v be a vertex which represents a face vertex (x, y) with the following children: (x, a), (a, b), $\ldots$ (i, z), (z, w), $\ldots$ (j, y). After merging all of the children, we get a table for the subgraph $G(x, y)$, which includes the objective function of the subgraph $G(x, y)$ for each bitpair of (x, y), but does not include the edge (x, y) if such an edge exists. The goal of procedure adjust is to solve the objective function at hand after adding this edge.

To apply Baker's algorithm to an objective function `Obj` of our choice, we must show how to implement the subroutines **merge** and **adjust**. It is not hard to verify that to prove that one can implement the **merge** operation, it suffices to present an algorithm that takes as input two induced subgraphs G_1 and G_2 of G that intersect at a single vertex x and the solutions `Obj`(G_1), `Obj`$(G_1 - \{x\})$, `Obj`(G_2) and `Obj`$(G_2 - \{x\})$ and returns a solution for `Obj`$(G_1 \cup G_2)$. Here, as the graphs G_1 and G_2 are induced subgraphs of G notice that there are no edges between $G_1 - \{x\}$ and $G_2 - \{x\}$. Similarly, for the **adjust** operation, it suffices to present an algorithm which takes as input an outerplanar graph G that includes vertices x and y but does not include the edge (x, y) on the outer face of G, the solutions `Obj`$(G - \{x\})$, `Obj`$(G - \{y\})$, `Obj`$(G - \{x, y\})$, and `Obj`(G) and returns a solution for `Obj`$(G + e)$. The implementation of these tasks for the clique cover and `MinRank` objective functions are in cases highly non-trivial, and to the best of our knowledge have not been addressed in the past. In the upcoming sections we will address the task of implementing these subroutines efficiently.

Chapter 2

The MinRank objective function

As described in the Introduction, finding the optimal (scalar) linear solution to an index coding instance with undirected side information graph G is equivalent to determining the matrix A for which $\texttt{MinRank}(G) = \textbf{rank}(A)$, e.g., [3]. Thus, in the remainder of our presentation we will only consider the MinRank problem. Recall that for a graph G, vertices x and y and an edge e, we denote by $G + \{x\}$, $G - \{x\}$, $G + e$, $G - e$, $G + (x, y)$, and $G - (x, y)$ the new graph obtained by adding or removing the vertex x, the edge e, or the edge (x, y) to G respectively.

2.1 Merge

In the **merge** operation one takes two subgraphs that are *almost* disjoint for which the optimal MinRank is known, and returns the MinRank of their union (including the corresponding matrix A of minimum rank). If the graphs were disjoint then the MinRank of the union is just the union of the corresponding MinRanks, however, as the subgraphs share a vertex, the MinRank of their union might be smaller according to the claim below.

Claim 2.1.1. *Let G_1 and G_2 be induced subgraphs of G that have a single vertex x in common. If $\texttt{MR}_\Sigma(G_1) = \texttt{MR}_\Sigma(G_1 - \{x\})$ or $\texttt{MR}_\Sigma(G_2) = \texttt{MR}_\Sigma(G_2 - \{x\})$, then $\texttt{MR}_\Sigma(G_1 \cup G_2) = \texttt{MR}_\Sigma(G_1 - \{x\}) + \texttt{MR}_\Sigma(G_2 - \{x\})$. Otherwise, $\texttt{MR}_\Sigma(G_1 \cup G_2) = \texttt{MR}_\Sigma(G_1 - \{x\}) + \texttt{MR}_\Sigma(G_2 - \{x\}) + 1$. Moreover, in both cases the corresponding matrix A of minimum rank that fits $G_1 \cup G_2$ can be obtained efficiently from the matrices corresponding to the MinRank of $G_1 - \{x\}$ and $G_2 - \{x\}$.*

Proof. G_1 and G_2 intersect at a single node, thus, using Property 3.3.3, we can find the $\texttt{MR}_\Sigma$ of $G_1 \cup G_2$. $\square$

2.2 Adjust

In the "adjust" operation one takes an outerplanar graph G with vertices x and y but without the edge (x, y) that lies on the outer face of G; and computes the MinRank of $G + e$ based on the MinRank of the graph $G - \{x\}$, the graph $G - \{y\}$, the graph $G - \{x, y\}$ and the graph G. As we have shown, it holds that $\texttt{MR}_\Sigma(G + e)$ either equals $\texttt{MR}_\Sigma(G)$ or is smaller and equals $\texttt{MR}_\Sigma(G) - 1$, however the correct answer depends strongly on the the values of the MinRank in the subgraphs of G that do not include the vertices x or y. The following two claims summarize the adjust operation when applied to the MinRank objective function. The first claim covers almost all possible settings except one, and can be proven relatively straightforward from the basic properties of MinRank combined with Property 3.3.3. The second claim addresses the last setting, and is more challenging.

Claim 2.2.1. *Let G be an outerplanar graph that includes vertices x and y but does not include the edge $e = (x, y)$ that sits on the outer face of G. Then the value of $\texttt{MR}_\Sigma(G + e)$ is determined by the following table which expresses the possible input values of $\texttt{MR}_\Sigma(G - \{x, y\})$, $\texttt{MR}_\Sigma(G - \{x\})$, and $\texttt{MR}_\Sigma(G - \{y\})$ as a function of $\lambda = \texttt{MR}_\Sigma(G)$; and the resulting value of $\texttt{MR}_\Sigma(G + e)$ as a function of $\lambda = \texttt{MR}_\Sigma(G)$:*

$\mathtt{MR}_\Sigma(G-\{x,y\})$	$\mathtt{MR}_\Sigma(G-\{x\})$	$\mathtt{MR}_\Sigma(G-\{y\})$	$\mathtt{MR}_\Sigma(G+e)$
λ			λ
$\lambda-2$			$\lambda-1$
$\lambda-1$	λ	$\lambda-1$	λ
$\lambda-1$	$\lambda-1$	λ	λ

Moreover, the minimum rank matrix A corresponding to $\mathtt{MR}_\Sigma(G+e)$ can be computed efficiently using the matrices corresponding to the `MinRank` *of the subgraphs above.*

Proof. We consider the different cases stated in the assertion:

1. $\mathtt{MR}_\Sigma(G-\{x,y\}) = \mathtt{MR}_\Sigma(G)$. As $e=(x,y)$, it holds that $\mathtt{MR}_\Sigma(G+e) \geq \mathtt{MR}_\Sigma(G-\{x,y\})$ (as any matrix that fits $G+e$ has a minor which fits $G-\{x,y\}$). Thus, adding the nodes $\{x,y\}$ and the edge (x,y) can't decrease the `MinRank` of the former graph $\mathtt{MR}_\Sigma(G-\{x,y\})$ which is equal to $\mathtt{MR}_\Sigma(G)$.

2. $\mathtt{MR}_\Sigma(G-\{x,y\}) = \mathtt{MR}_\Sigma(G)-2$. Adding the nodes $\{x,y\}$ plus the edge (x,y) to $G-\{x,y\}$ increases its `MinRank` by at most 1 (one can just append to a matrix that fits $G-\{x,y\}$ two row vectors that are 1 on coordinates x and y and 0 otherwise). Thus we get $\mathtt{MR}_\Sigma(G+e) \leq (\mathtt{MR}_\Sigma(G)-2)+1 = \mathtt{MR}_\Sigma(G)-1$. The value $\mathtt{MR}_\Sigma(G+e)$ can't be less than or equal to $\mathtt{MR}_\Sigma(G)-2$ because of Property 3.3.2 (adding an edge to a graph can decrease the `MinRank` by 1 at most).

3. $\mathtt{MR}_\Sigma(G-\{x\}) = \mathtt{MR}_\Sigma(G)$. As in the first case it holds that $\mathtt{MR}_\Sigma(G+e) \geq \mathtt{MR}_\Sigma(G-\{x\})$ (as any matrix that fits $G+e$ has a minor which fits $G-\{x\}$).

4. Same as 3.

□

Claim 2.2.2. *Let G be an outerplanar graph that includes vertices x and y but does not include the edge $e=(x,y)$ that sits on the outer face of G. Then if $\mathtt{MR}_\Sigma(G-\{x,y\}) = \mathtt{MR}_\Sigma(G-\{x\}) = \mathtt{MR}_\Sigma(G-\{y\}) = \mathtt{MR}_\Sigma(G)-1$ the* `MinRank` *of $G+e$ is determined by the following cases. (a) If there are no vertices z in $G+e$ such that x,y,z form a triangle (a clique) in $G+e$ then $\mathtt{MR}_\Sigma(G+e) = \mathtt{MR}_\Sigma(G)$. (b) If there exists a vertex z in $G+e$ such that x,y,z form a triangle (a clique) in $G+e$ then $\mathtt{MR}_\Sigma(G+e)$ depends on $\mathtt{MR}_\Sigma(G-\{x,y,z\})$. Namely, if $\mathtt{MR}_\Sigma(G-\{x,y,z\}) = \mathtt{MR}_\Sigma(G)-2$ then $\mathtt{MR}_\Sigma(G+e) = \mathtt{MR}_\Sigma(G)-1$, otherwise $\mathtt{MR}_\Sigma(G+e) = \mathtt{MR}_\Sigma(G)$. Moreover, the minimum rank matrix corresponding to $\mathtt{MR}_\Sigma(G+e)$ can be computed efficiently using the matrices corresponding to the* `MinRank` *of the subgraphs above.*

Claim 2.2.2 is the most challenging claim proven in this part. Examples in which the `MinRank` of G after adding the edge (x,y) are equal to $\mathtt{MR}_\Sigma(G)$ or $\mathtt{MR}_\Sigma(G)-1$ are given below.

- If $x-z-y$ is a path, then adding the edge (x,y) closes a triangle and the `MinRank` is decreased by 1.

- If $x-z-u-v-y$ is a path, then adding the edge (x,y) doesn't decrease the `MinRank`.

Proof. We consider the following cases:

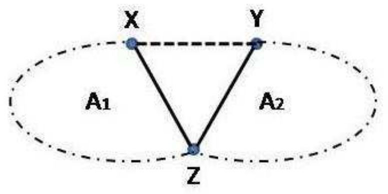

Figure 2.1: The partition of G into 3 graphs in the proof of Claim 2.2.2 (case 1).

Case 1: There is a node $'z'$ such that x, y, z close a triangle. According to the tree construction, (x, z) and (z, y) are children of the node (x, y). Otherwise, the triangle (x, y, z) doesn't belong to the subgraph represented by $G(x, y)$. When executing the procedure `Table(`v`)` on the vertex (x, y), there is an inductive call to `Table(`x, z`)` and `Table(`z, y`)`. Thus, the following values are known prior to the "adjust" operation on the vertex (x, y): $\mathtt{MR}_\Sigma(G(x, z))$, $\mathtt{MR}_\Sigma(G(x, z) - \{x, z\})$, $\mathtt{MR}_\Sigma(G(z, y))$, and $\mathtt{MR}_\Sigma(G(z, y) - \{z, y\})$. Recall that according to our definitions these values are stored in the tables of (x, z) and (z, y) accordingly. Moreover, $\mathtt{MR}_\Sigma(G - \{x, y, z\})$ is known and equals $\mathtt{MR}_\Sigma(G(x, z) - \{x, z\}) + \mathtt{MR}_\Sigma(G(z, y) - \{z, y\})$ (the subgraphs $G(x, z)$ and $G(z, y)$ excluding the nodes x, z, y are disjoint graphs because of the outerplanarity constraint, see, e.g., Figure 2.1).

We consider two cases: If $\mathtt{MR}_\Sigma(G - \{x, y, z\}) = \mathtt{MR}_\Sigma(G) - 2$, we claim that $\mathtt{MR}_\Sigma(G + e) = \mathtt{MR}_\Sigma(G) - 1$. This follows since, given a matrix A that fits $G - \{x, y, z\}$ one can construct one that fits $G + e$ by expanding A in a natural way (adding rows/columns corresponding to x, y, z), in which the only new entries that are non-zero are those corresponding to a pair in $\{x, y, z\}$ (which can be set to equal 1). Thus, $\mathtt{MR}_\Sigma(G + (x, y)) = \mathtt{MR}_\Sigma(G) - 1$ (the `MinRank` is decreased by 1 after adding the edge (x, y)).

For the second case, if $\mathtt{MR}_\Sigma(G - \{x, y, z\}) = \mathtt{MR}_\Sigma(G) - 1$, we now show that adding the edge (x, y) can't decrease the `MinRank`, thus $\mathtt{MR}_\Sigma(G + (x, y)) = \mathtt{MR}_\Sigma(G)$.

Claim 2.2.3. *If* $\mathtt{MR}_\Sigma(G - \{x, y, z\}) = \mathtt{MR}_\Sigma(G) - 1$, *then* $\mathtt{MR}_\Sigma(G + (x, y)) = \mathtt{MR}_\Sigma(G)$.

Proof. We start by noting that the removal of the three vertices $\{x, y, z\}$ in G disconnects G into 2 disjoint components. See, e.g., Figure 2.1. We denote these components by A_1 and A_2. As $G + e$ is outerplanar, and the edge (x, y) is on the outer face of $G + e$, we can assume w.l.o.g. that vertices in A_1 are not connected by an edge to vertices in A_2. Moreover, vertices in A_1 are not connected by an edge to y, while vertices in A_2 are not connected by an edge to x.

Define M as a matrix which fits $G+(x, y)$, and has the minimum **rank** among all such matrices. By definition, $\mathtt{MR}_\Sigma(G+(x, y)) = \min\{\textbf{rank}(M) \mid M \text{ fits } (G+(x, y))\}$. Assume in contradiction that after adding the edge (x, y), $\mathtt{MR}_\Sigma(G)$ is decreased by 1, meaning that $\textbf{rank}(M) = \mathtt{MR}_\Sigma(G+(x, y)) = \mathtt{MR}_\Sigma(G) - 1$. We will see that by changing a few entries in M we obtain a new matrix M', with the same **rank** as M, which fits G (the original graph without (x, y)), implying that $\mathtt{MR}_\Sigma(G) \leq \textbf{rank}(M') = \textbf{rank}(M) = \mathtt{MR}_\Sigma(G) - 1$, a contradiction. We will thus conclude that $\mathtt{MR}_\Sigma(G + (x, y)) = \mathtt{MR}_\Sigma(G)$. Consider the matrix M:

$$\left(\begin{array}{c||c|c|c|c|c}
 & A_1 & x & y & z & A_2 \\
\hline\hline
A_1 & A_{1l} & & 0 & & 0 \\
\hline
x & x_l & 1 & x_y & * & 0 \\
\hline
y & 0 & y_x & 1 & * & y_r \\
\hline
z & z_l & * & * & 1 & z_r \\
\hline
A_2 & 0 & 0 & & & A_{2r}
\end{array}\right)$$

In the above description we use the following notation. A_1 represents the subgraph corresponding to the face vertex (x, z) in $\bar{G}$, excluding the nodes x, z. A_2 represents the subgraph corresponding to the face vertex (z, y) in $\bar{G}$ excluding the nodes z, y. An entry m_{uv} in M corresponds to the nodes labeled by u and v in G. Each row of M corresponds to a node v in G. For the row vector $(v_1 \dots v_n)$ corresponding to v we denote its entries corresponding to A_1 by v_{left} or v_l, its entries corresponding to A_2 by v_{right} or v_r and its entries corresponding to a vertex w by v_w. For example, for the vertex x we have that $x_x = 1$. Also for a vertex $v \in A_2$ we have that $v_l = 0$ as there are no edges between A_2 and A_1. Finally, for any vertex v we abuse notation and refer to the row vector corresponding to v by the same notation: v. Similarly for a set of vertices A we refer to the row vectors corresponding to A in M by A.

In the description of M above we have specified entries that must be 0, entries that must be 1, and some entries of interest that can be either 0 or 1 (denoted by *). For each matrix which fits the graph G (the graph that doesn't include the edge (x, y)), it is known that $x_y = 0$ and $y_x = 0$, because there is no edge between x and y. Consider the matrix M above which fits $G + (x, y)$. If $x_y = 0$ and $y_x = 0$, then M also fits G, implying that $\texttt{MR}_\Sigma(G) \leq \textbf{rank}(M) = \texttt{MR}_\Sigma(G) - 1$, a contradiction. Thus, one of the values x_y or y_x must equal 1. Recall that we are assuming that $\texttt{MR}_\Sigma(G - \{x, y, z\}) = \texttt{MR}_\Sigma(G) - 1$, meaning that $\textbf{rank}(M) = \texttt{MR}_\Sigma(G) - 1 = \texttt{MR}_\Sigma(G - \{x, y, z\}) = \texttt{MR}_\Sigma(A_1 \cup A_2)$. Consider the rows of M corresponding to A_1 and A_2. We now claim that these rows span a vector space of dimension $\texttt{MR}_\Sigma(G) - 1$ and thus span M. Indeed, if the rows of A_1 and A_2 spanned a vector space of lower dimension, then one could construct a matrix that fits the subgraph induced by $A_1 \cup A_2$ with **rank** lower than $\texttt{MR}_\Sigma(G) - 1$. However, as stated above, $\texttt{MR}_\Sigma(A_1 \cup A_2) = \texttt{MR}_\Sigma(G) - 1$, a contradiction.

We conclude that the row vector x (corresponding to the vertex x) is in $\textbf{span}(A_1 \cup A_2)$. The same holds for the vectors y and z. Consider the sub vector x_l. Potentially, x_l could be spanned by the left coordinates in the vectors of A_1 (denoted as A_{1l}) and the left coordinates in the vectors of A_2 (denoted as A_{2l}). However, A_{2l} is all zero (as there are no edges between vertices of A_2 and those of A_1). Thus, $x_l \in \textbf{span}(A_{1l})$. Moreover, it also holds that the vector (x_l, x_x) (i.e, the entries of x corresponding to vertices $(A_1 \cup x)$) is spanned by the coordinates of A_1 corresponding to the vertices $(A_1 \cup x)$.

We now suggest to change the vector x to x' such that $x' \in \textbf{span}(A_1)$. This can be done by zeroing out the value of x in the coordinate y and by changing the value of x in coordinate z according to the following rule. By the discussion above, let the coefficients $\{\alpha_i\}$ satisfy : $(x_l, x_x) = \sum_{a_i \in A_1} \alpha_i(a_{il}, a_{ix})$. Now set $x' = \sum_{a_i \in A_1} \alpha_i a_i$. Notice that $x'_y = 0$. In a similar (and symmetric) way we can change the vector y to $y' \in \textbf{span}(A_2)$ such that $y'_x = 0$. The resulting matrix M' still has rank at most $\texttt{MR}_\Sigma(G) - 1 = \textbf{rank}(A_1 \cup A_2)$, as all new and old row vectors are spanned by $A_1 \cup A_2$. Moreover, M' fits G. Thus, $\texttt{MR}_\Sigma(G) \leq \textbf{rank}(M') = \texttt{MR}_\Sigma(G) - 1$, a contradiction. $\square$

Case 2: There is no node $'z'$ such that x, y, z closes a triangle. In Figure 2.2 we present a decomposition of G into 4 graphs: A_1, A_2, Z, $\{x, y, m, n\}$. We denote by m the neighbor of x (on the corresponding face) which is furthest away from x in a counterclockwise direction. We denote by n the neighbor of y which is furthest away

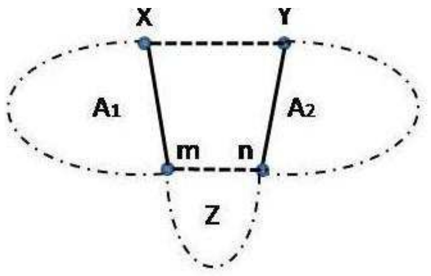

Figure 2.2: The partition of G into 4 graphs in the proof of Claim 2.2.3 (case 2).

from y (on the corresponding face) in a clockwise direction. Removing the vertices x, y, m and n from G, we get the 4 disconnected components that appear in Figure 2.2.

Define M as a matrix which fits $G+(x,y)$, and has the minimum **rank** among all such matrices. By definition, $\mathtt{MR}_\Sigma(G+(x,y)) = \min\{\textbf{rank}(M) \mid M \text{ fits } (G+(x,y))\}$. Assume in contradiction that after adding the edge (x,y), $\mathtt{MR}_\Sigma(G)$ is decreased by 1, meaning that $\textbf{rank}(M) = \mathtt{MR}_\Sigma(G+(x,y)) = \mathtt{MR}_\Sigma(G) - 1$. We will see that by changing a few entries in M we obtain a new matrix M', with the same **rank** as M, which fits G (the original graph without (x,y)), implying that $\mathtt{MR}_\Sigma(G) \le \textbf{rank}(M') = \textbf{rank}(M) = \mathtt{MR}_\Sigma(G) - 1$, a contradiction. We will thus conclude that $\mathtt{MR}_\Sigma(G+(x,y)) = \mathtt{MR}_\Sigma(G)$. Consider the matrix M (we use a similar representation to that given in the analysis of the previous case).

$$\left(\begin{array}{c||c|c|c|c|c|c|c}
 & A_1 & x & m & Z & n & y & A_2 \\
\hline\hline
A_1 & & & & 0 & 0 & 0 & 0 \\
\hline
x & x_l & 1 & * & 0 & 0 & x_y & 0 \\
\hline
m & m_l & * & 1 & & & 0 & 0 \\
\hline
Z & 0 & 0 & & & & 0 & 0 \\
\hline
n & 0 & 0 & & & 1 & * & n_r \\
\hline
y & 0 & y_x & 0 & 0 & * & 1 & y_r \\
\hline
A_2 & 0 & 0 & 0 & 0 & & &
\end{array}\right)$$

For each matrix which fits the graph G (the graph that doesn't include the edge (x,y)), it is known that $x_y = 0$ and $y_x = 0$, because there is no edge between x and y. Consider the matrix M above which fits $G+(x,y)$. If $x_y = 0$ and $y_x = 0$, then M also fits G, implying that $\mathtt{MR}_\Sigma(G) \le \textbf{rank}(M) = \mathtt{MR}_\Sigma(G) - 1$, a contradiction. Thus, one of the values x_y or y_x must equal 1. Recall that we are asuuming that $\mathtt{MR}_\Sigma(G - \{x,y\}) = \mathtt{MR}_\Sigma(G) - 1$, meaning that $\mathtt{MR}_\Sigma(G) - 1 = \textbf{rank}(M) = \textbf{rank}(G - \{x,y\}) = \mathtt{MR}_\Sigma(A_1 \cup A_2 \cup Z \cup m \cup n)$ (here, "$A_1 \cup A_2 \cup Z \cup m \cup n$" refers to the subgraph induced on these vertices of G). Consider the rows of M corresponding to A_1 , A_2, Z, m, n. We now claim that these rows span a vector space of dimension $\mathtt{MR}_\Sigma(G) - 1$ and thus span M. Indeed, if the rows of A_1 , A_2, Z, m, n spanned a vector space of lower dimension, then one could construct a matrix that fits the subgraph induced by $A_1 \cup A_2 \cup Z \cup m \cup n$ with **rank** lower than $\mathtt{MR}_\Sigma(G) - 1$. However, as stated above, $\mathtt{MR}_\Sigma(A_1 \cup A_2 \cup Z \cup m \cup n) = \mathtt{MR}_\Sigma(G) - 1$, a contradiction.

As before, for a vertex x we abuse notation and refer to x as the corresponding row vector in M. Similarly for a set of vertices A we refer to the row vectors corresponding to A in M by A. We conclude that the row vector x (corresponding to the vertex x) is in $\textbf{span}(A_1 \cup A_2 \cup Z \cup m \cup n)$. The same holds for the vector y.

The following cases must be considered:

- Both m and n are in **span**$(A_1 \cup A_2 \cup Z)$, meaning that **rank**$(A_1 \cup A_2 \cup Z) = \mathtt{MR}_\Sigma(G) - 1$. In this case, the row vector x is in **span**$(A_1 \cup A_2 \cup Z)$. Consider the sub vector x_l. Potentially, x_l could be spanned by the left coordinates in the vectors of A_1 (denoted as A_{1l}), the left coordinates in the vectors of A_2 (denoted as A_{2l}), and the left coordinates in the vectors of Z (denoted as Z_l). However, A_{2l} is all zero (as there are no edges between vertices of A_2 and those of A_1), and Z_l is all zero (as there are no edges between vertices of Z and those of A_1). Thus, $x_l \in$ **span**(A_{1l}). Moreover, it also holds that the vector (x_l, x_x) (i.e, the entries of x corresponding to vertices $(A_1 \cup x)$) is spanned by the coordinates of A_1 corresponding to the vertices $(A_1 \cup x)$.

  We now suggest to change the vector x to x' such that $x' \in$ **span**(A_1). This can be done by zeroing out the values of x in the coordinate y and by changing the value of x in coordinate m according to the following rule. By the discussion above, let the coefficients $\{\alpha_i\}$ satisfy : $(x_l, x_x) = \sum_{a_i \in A_1} \alpha_i(a_{il}, a_{ix})$. Now set $x' = \sum_{a_i \in A_1} \alpha_i a_i$.

  Similarly, we change the vector y to $y' \in$ **span**(A_2) with $y'_x = 0$. The resulting matrix M' still has rank at most $\mathtt{MR}_\Sigma(G) - 1 =$ **rank**$(A_1 \cup A_2 \cup Z)$, as all new and old row vectors are spanned by $A_1 \cup A_2 \cup Z$. Moreover, M' fits G. Thus, $\mathtt{MR}_\Sigma(G) \leq$ **rank**$(M') = \mathtt{MR}_\Sigma(G) - 1$, a contradiction.

- n is in **span**$(A_1 \cup A_2 \cup Z)$, and m not. We first suggest to change the vector m to m' by zeroing out the values of m in the coordinates Z and n. Now consider the vector x, or more specifically the sub vector x_l. Potentially, x_l could be spanned by the left coordinates in the vectors of A_1 (denoted as A_{1l}), the left coordinates in the vectors of A_2 (denoted as A_{2l}), the left coordinates in the vectors of Z (denoted as Z_l), and the left coordinates in the vector m (denoted as m_l). However, A_{2l} is all zero (as there are no edges between vertices of A_2 and those of A_1), and Z_l is all zero (as there are no edges between vertices of Z and those of A_1). Thus, $x_l \in$ **span**$(A_{1l} \cup m_l) =$ **span**$(A_{1l} \cup m'_l)$. Moreover, it also holds that the vector (x_l, x_x) (i.e, the entries of x corresponding to vertices $(A_1 \cup x)$) is spanned by the coordinates of A_1 and m' corresponding to the vertices $(A_1 \cup x)$.

  We now suggest to change the vector x to x' such that $x' \in$ **span**$(A_1 \cup m')$. This can be done by zeroing out the values of x in the coordinate y and by changing the value of x in coordinate m according to the following rule. By the discussion above, let the coefficients $\{\alpha_i\}$ satisfy : $(x_l, x_x) = \sum_{a_i \in A_1 \cup \{m'\}} \alpha_i(a_{il}, a_{ix})$. Now set $x' = \sum_{a_i \in A_1 \cup \{m'\}} \alpha_i a_i$. Notice that as m' is 0 is coordinates corresponding to Z, n, y and A_2 the same holds for so is x'.

  We now address the vector y as we did in the former case (recall that n is in **span**$(A_1 \cup A_2 \cup Z)$). Namely, we change the vector y to $y' \in$ **span**(A_2) with $y_x = 0$. The matrix M' with the new rows m', x' and y' still has rank at most **rank**$(A_1 \cup A_2 \cup Z \cup \{m'\}) \leq$ **rank**$(A_1 \cup A_2 \cup Z \cup \{m\}) = \mathtt{MR}_\Sigma(G) - 1$. This follows as all new and old row vectors are spanned by $A_1 \cup A_2 \cup Z \cup \{m'\}$. Moreover, M' fits G. Thus, $\mathtt{MR}_\Sigma(G) \leq$ **rank**$(M') = \mathtt{MR}_\Sigma(G) - 1$, a contradiction.

- n is in **span**$(A_1 \cup A_2 \cup Z)$ and m not. Similar to the former case.

- both m and n are not in **span**$(A_1 \cup A_2 \cup Z)$. We need to consider the following cases:

  – m is not in **span**$(A_1 \cup A_2 \cup Z \cup n)$ and n is not in **span**$(A_1 \cup A_2 \cup Z \cup m)$. This implies that **span**$(A_1 \cup A_2 \cup Z) = \mathtt{MR}_\Sigma(G) - 3$. We suggest to change the vector m to m' by zeroing out the values of m in the coordinates Z and n. We also suggest to change the vector n to n' by zeroing out the values of n in the coordinates Z and m. Now we can change the vector x to x' such that $x' \in$ **span**$(A_1 \cup m')$. This can be done by zeroing out the values of x in the coordinate y and by changing the value of x in coordinate m as done before. Similarly, we change the vector y to $y' \in$ **span**$(A_2 \cup n')$. The resulting matrix M' (with m', n', x' and y') has rank at most **rank**$(A_1 \cup A_2 \cup Z \cup \{m'\} \cup \{n'\}) \leq \mathtt{MR}_\Sigma(G) - 1$, as all new and old row vectors are spanned by $A_1 \cup A_2 \cup Z \cup m' \cup n'$, and we noted that **span**$(A_1 \cup A_2 \cup Z) = \mathtt{MR}_\Sigma(G) - 3$. Moreover, M' fits G. Thus, $\mathtt{MR}_\Sigma(G) \leq$ **rank**$(M') = \mathtt{MR}_\Sigma(G) - 1$, a contradiction.

– n is in **span**$(A_1 \cup A_2 \cup Z \cup m)$. As $n \notin$ **span**$(A_1 \cup A_2 \cup Z)$ it follows that $n = \alpha_m m + \sum_{a_i \in A_1 \cup A_2 \cup Z} \alpha_i a_i$, where $a_m \neq 0$. Consider the sub vector $n_l = \alpha_m m_l + \sum_{a_i \in A_1 \cup A_2 \cup Z} \alpha_i a_{il}$. A_{2l} and Z_l are all zero, meaning that $n_l = \alpha_m m_l + \sum_{a_i \in A_1} \alpha_i a_{il}$. Moreover, n_l must equal 0 as there are no edges between vertices of A_1 and n. Thus, $0 = \alpha_m m_l + \sum_{a_i \in A_1} \alpha_i a_{il}$ where $\alpha_m \neq 0$. This implies that m_l is in **span**(A_{1l}). Consider the sub vector x_l. Potentially, x_l could be spanned by A_{1l}, A_{2l}, Z_l, m_l and n_l. A_{2l} , Z_l, and n_l are all zero, meaning that $x_l \in$ **span**$(A_{1l} \cup m_l)$. As we know that $m_l \in$ **span**(A_{1l}), we conclude that $x_l \in$ **span**(A_{1l}). Now we can change the vector x to x' such that $x' \in$ **span**(A_1) exactly as done before.

  To address y, notice that in this case we have that (y_r, y_y) is spanned by the corresponding coordinates of A_2. Thus as done before we change the vector y to $y' \in$ **span**(A_2) with $y'_x = 0$. The resulting matrix M' (with x' and y') still has rank at most $\mathtt{MR}_\Sigma(G) - 1 =$ **rank**$(A_1 \cup A_2 \cup Z \cup m)$, as all new and old row vectors are spanned by $A_1 \cup A_2 \cup Z \cup m$. Moreover, M' fits G. Thus, $\mathtt{MR}_\Sigma(G) \leq$ **rank**$(M') = \mathtt{MR}_\Sigma(G) - 1$, a contradiction.

– m is in **span**$(A_1 \cup A_2 \cup Z \cup n)$. Similar to the former case.

□

Chapter 3

The clique cover objective function

As in Section 2 one may prove properties for the clique cover objective on the routines “merge” and ”adjust” in the algorithm of [2]. The claims corresponding to the clique cover objective function are exactly those presented in Section 2 when the objective `MinRank` (or $\mathtt{MR}_\Sigma(G)$) is replaced by the term “clique cover” (or $\mathtt{CC}(G)$). We note that the first and second claims we discuss for the clique cover objective follow relatively straightforwardly from the basic properties of clique covers in undirected graphs, while the third claim (analogous to Claim 2.2.2) is more challenging to prove.

3.1 Merge

Claim 3.1.1. *Let G_1 and G_2 be induced subgraphs of G that have a single vertex x in common. If $\mathtt{CC}(G_1) = \mathtt{CC}(G_1 - \{x\})$ or $\mathtt{CC}(G_2) = \mathtt{CC}(G_2 - \{x\})$, then $\mathtt{CC}(G_1 \cup G_2) = \mathtt{CC}(G_1 - \{x\}) + \mathtt{CC}(G_2 - \{x\})$. Otherwise, $\mathtt{CC}(G_1 \cup G_2) = \mathtt{CC}(G_1 - \{x\}) + \mathtt{CC}(G_2 - \{x\}) + 1$. Moreover, in both cases the optimal clique cover of $G_1 \cup G_2$ can be obtained efficiently from those of $G_1 - \{x\}$ and $G_2 - \{x\}$.*

Proof. G_1 and G_2 intersect at a single node, thus, using Property 3.2.3, we can find the clique cover of $G_1 \cup G_2$. $\square$

3.2 Adjust

We prove the following claims for k-outerplanar graphs (and not only outerplanar graph) as we will use them in part II of this Thesis.

Claim 3.2.1. *Let G be a k-outerplanar graph that includes vertices x and y but does not include the edge $e = (x, y)$ that sits on the outer face of G. Then the value of $\mathtt{CC}(G + e)$ is determined by the following table which expresses the possible input values of $\mathtt{CC}(G - \{x, y\})$, $\mathtt{CC}(G - \{x\})$, and $\mathtt{CC}(G - \{y\})$ as a function of $\lambda = \mathtt{CC}(G)$; and the resulting value of $\mathtt{CC}(G + e)$ as a function of $\lambda = \mathtt{CC}(G)$:*

$\mathtt{CC}(G - \{x,y\})$	$\mathtt{CC}(G - \{x\})$	$\mathtt{CC}(G - \{y\})$	$\mathtt{CC}(G + e)$
λ			λ
$\lambda - 2$			$\lambda - 1$
$\lambda - 1$	λ	$\lambda - 1$	λ
$\lambda - 1$	$\lambda - 1$	λ	λ

Moreover, the optimal clique cover of $G + e$ can be computed in linear time using the optimal clique covers of the subgraphs above.

Proof. We consider the following cases of the assertion:

1. $\texttt{CC}(G-\{x,y\})$ equals to $\texttt{CC}(G)$. As $e=(x,y)$, it holds that $\texttt{CC}(G+e)\geq\texttt{CC}(G-\{x,y\})$ (as any clique cover for $G+e$ is one for $G-\{x,y\}$).

2. $\texttt{CC}(G-\{x,y\})=\texttt{CC}(G)-2$. Adding the nodes $\{x,y\}$ plus the edge (x,y) increases $\texttt{CC}(G)$ by 1, because the new clique cover consists of the cliques of $\texttt{CC}(G-\{x,y\})$ with the addition of the clique (x,y). The new clique cover is $(\texttt{CC}(G)-2)+1=\texttt{CC}(G)-1$. The new clique cover (after adding the edge (x,y)) can't be less than or equal to $\texttt{CC}(G)-2$ because of Property 3.2.2 (adding an edge to a graph can decrease the clique-cover by 1 at most).

3. $\texttt{CC}(G-\{x\})=\texttt{CC}(G)$. As in the first case, it holds that $\texttt{CC}(G+e)\geq\texttt{CC}(G-\{x\})$ (as any clique cover for $G+e$ is one for $G-\{x\}$).

4. Same as 3.

□

Claim 3.2.2. *Let G be a k-outerplanar graph that includes vertices x and y but does not include the edge $e=(x,y)$ that sits on the outer face of G. Then if* $\texttt{CC}(G-\{x,y\})=\texttt{CC}(G-\{x\})=\texttt{CC}(G-\{y\})=\texttt{CC}(G)-1$ *the optimal clique cover of $G+e$ is determined by the following cases. (a) If there are no vertices z in $G+e$ such that x,y,z from a triangle (a clique) in $G+e$ then* $\texttt{CC}(G+e)=\texttt{CC}(G)$. *(b) If there exists a vertex z in $G+e$ such that x,y,z from a triangle (a clique) in $G+e$ then* $\texttt{CC}(G+e)$ *depends on* $\texttt{CC}(G-\{x,y,z\})$. *Namely, if* $\texttt{CC}(G-\{x,y,z\})=\texttt{CC}(G)-2$ *then* $\texttt{CC}(G+e)=\texttt{CC}(G)-1$, *otherwise* $\texttt{CC}(G+e)=\texttt{CC}(G)$. *Moreover, the optimal clique cover of $G+e$ can be computed in linear time using the optimal clique covers of the subgraphs above.*

We note that the clique cover of G after adding the edge (x,y) can be equal to $\texttt{CC}(G)$ or $\texttt{CC}(G)-1$, for example:

- If $x-z-y$ is a path, then adding the edge (x,y) closes a triangle and the clique cover is decreased by 1.
- If $x-z-u-v-y$ is a path, then adding the edge (x,y) doesn't decrease the clique cover.

Proof. We consider the following cases:

Case 1: There is no node $'z'$ such that x,y,z close a clique. Consider the clique cover of G of size $\texttt{CC}(G)$. Assume that adding the edge (x,y) decreases $\texttt{CC}(G)$ by 1. The minimum clique cover of $G+e$ must use the edge e and contain the clique $\{x,y\}$, otherwise, the new edge (x,y) doesn't have any influence and it can't change the value of $\texttt{CC}$. Here we use the fact that any clique containing $\{x,y\}$ must be of size 2 at most. Thus, the new graph without the clique $\{x,y\}$ contains $\texttt{CC}(G)-2$ cliques, in contradiction to the assumption that $\texttt{CC}(G-\{x,y\})=\texttt{CC}(G)-1$.

Case 2: There is a node $'z'$ such that x,y,z close a triangle. As stated in Section 2, according to the tree construction (x,z) and (z,y) are children of the node (x,y), Otherwise, the triangle (x,y,z) doesn't belong to the subgraph represented by $G(x,y)$. When executing the procedure $\texttt{Table}(v)$ on the vertex (x,y), there is an inductive call to $\texttt{Table}(x,z)$ and $\texttt{Table}(z,y)$. Thus, the following values are known prior to the "adjust" operation on the vertex (x,y): $\texttt{CC}(G(x,z))$, $\texttt{CC}(G(x,z)-\{x,z\})$, $\texttt{CC}(G(z,y))$, and $\texttt{CC}(G(z,y)-\{z,y\})$. Recall that according to definition these values are stored in the tables of (x,z) and (z,y) accordingly. Moreover, $\texttt{CC}(G-\{x,y,z\})$ is known and equals $\texttt{CC}(G(x,z)-\{x,z\})+\texttt{CC}(G(z,y)-\{z,y\})$ (the subgraphs $G(x,z)$ and $G(z,y)$ excluding the nodes x,z,y are disjoint graphs because of the outerplanarity constraint, see, e.g., Figure 2.1.

If $\texttt{CC}(G-\{x,y,z\})$ equals $\texttt{CC}(G)-2$, then adding the clique $\{x,y,z\}$ causes the clique cover of $G+e$ to be of size $\texttt{CC}(G)-1$ (the clique cover is decreased by 1). If $\texttt{CC}(G-\{x,y,z\})$ equals $\texttt{CC}(G)-1$, then adding the edge (x,y) can't decrease the clique cover size. This follows as the new clique cover must contain a clique including the set $\{x,y\}$ (otherwise the added edge e doesn't have any influence), and can be assumed without loss of generality to contain $\{x, y, z\}$. Thus, the clique cover of the remaining graph (without $\{x, y, z\}$) equals $\texttt{CC}(G) - 2$ in contradiction to our assumption. $\square$

Chapter 4

Proof of Theorem 2.1.1

We now prove Theorem 2.1.1 given in the Introduction and show that $\texttt{MR}_\Sigma(G) = \texttt{CC}(G)$ in outerplanar graphs (and that both objective functions can be computed efficiently). In order to find the `MinRank` or the clique cover for outerplanar graphs, we processed a tree $\bar{G}$ (defined in Section 1) that represents the structure of G. We then used Baker's algorithm [2] via procedure `Table`(v) presented in Figure 1.2 to calculate for each tree vertex and corresponding subgraph G_1 of G the solution to the corresponding objective function. These computations were based on the "merge" and "adjust" operations studied in Sections 2 and 3. As the analysis for the `MinRank` and minimum clique cover given in Sections 2 and 3 are analogous, it follows by induction on the execution of Baker's algorithm that for any intermediate vertex in the tree $\bar{G}$ corresponding to a subgraph G_1 it holds that $\texttt{MR}_\Sigma(G_1) = \texttt{CC}(G_1)$. In particular, this also holds for G itself.

For the base case, consider a *level 1* vertex in the tree, i.e. a leaf v representing an edge (x, y). The table of v specifies that the size of the `MinRank` or the clique cover is 1 if exactly one endpoint of (x, y), or both of them are in the corresponding subgraph, and 0 if neither x nor y are in the subgraph. Thus, for a leaf v, its table values are equal for both the clique cover and `MinRank` problems. Assume that the statement is true for level $j < k$ vertices in the tree $\bar{G}$ (here, a level j vertex is one that has a path of length $j - 1$ to some leaf). The table for a level k vertex is calculated according to tables of level j vertices, using the procedures "merge" and "adjust" analyzed in Sections 2 and 3. As the claims states in Sections 2 and 3 are completely equivalent (given the change in the objective function), the inductive assertion follows. All in all, for any level k vertex, the calculation of table(v) depends only on the tables of level $j < k$ vertices. By assumption, these tables are equal for `MinRank` and clique cover and thus procedures "merge" and "adjust" give the same results for the same input. We conclude that `MinRank` and clique cover are equal for a level k vertex as well. This suffices to conclude our proof. The efficiency of our calculations follow directly from our constructive proofs.

Part II

A PTAS for clique cover on planar graphs

Chapter 1

Overview of Baker's algorithm for k - outerplanar graphs

In this part of the Thesis we address the clique cover problem when restricted to instances which are k-outerplanar. Recall that the clique cover problem on planar graphs is known to be NP-complete [8, 13]. We extend the ideas of Baker to show that one can solve clique cover on k-outerplanar graphs efficiently. We present an algorithm that runs in time $n^{O(k)}$ for its solution. The conclusion of this part is the proof of Theorem 2.1.2.

As in Part I, our work is based on Baker's algorithmic paradigm [2], focusing not only on outerplanar graphs, but also on the more complex setting on k-outerplanar graphs. We give a brief (and rough) overview of the main ideas that govern the algorithm of [2], and as before identify the major points that need to be addressed in order to apply the paradigm at hand to the case of the clique cover problem. The full description of Baker's algorithms for k-outerplanar graphs is rather complex and we refer the reader to [2] for complete details.

Given a k outerplanar graph G, the algorithm of Baker [2] has two major steps. In the first step, a tree representation $\bar{G}$ of G is constructed. Every node in $\bar{G}$ corresponds to a subgraph of G, where the root of $\bar{G}$ corresponds to G itself, each leaf in $\bar{G}$ corresponds to an edge, and internal nodes in $\bar{G}$ correspond to the subgraph of G induced by their children in $\bar{G}$. The construction of $\bar{G}$ from G is very simple in nature, for outerplanar graphs G, the tree $\bar{G}$ tightly resembles the standard notion of the *dual* to an outerplanar graph, while for k-outerplanar graphs, the construction of $\bar{G}$ has a inductive flavor based on k consecutive removal of vertices on the outer face.

In slightly more detail: For outerplanar graphs G, $\bar{G}$ is constructed as showed in Part I[1]. Place a vertex in each interior face and on each exterior edge, and draw an edge from each vertex representing a face f to each vertex representing either an adjacent face (i.e., a face sharing an edge with f) or an exterior edge of f.[2] An example (taken from [2]) is shown in Figure 1.1 of Part I.

Given a k-outerplanar graph G, and a corresponding planar embedding E, we define level i nodes. A node is at level 1 if it is on the exterior face. A cycle of level 1 nodes is called a level 1 face. For each level i face f, let G_f be the subgraph induced by the nodes placed inside f in this embedding. Then, the nodes of the exterior face of G_f are level $i+1$ nodes. Now, a cycle of level i nodes is called a level i face. A k outerplanar graph is a planar graph in which there are no nodes of level $> k$ in it's embedding. Throughout the paper, we assume that the graph is connected and that the level i nodes within every level $i-1$ face induce a connected subgraph.[3] We refer to this connected induced subgraph as a level i component. For example, in Figure 1.1, the nodes A, B, C, D, E, F are level 1 nodes, a, b, c, d, e, s, t, u are level 2 nodes, and x, y, z, w, m, n are level 3 nodes. In addition, $\{A, B, C, D\}$

[1]In this presentation we suppose there are no cutpoints in G, i.e., a vertex whose deletion disconnects the graph.

[2]This tree is closely related to the dual of the graph; however, the dual would lack vertices for exterior edges and would have an additional vertex for the exterior face.

[3]If not, as explained in [2] one adds some *fake* edges to obtain connectivity, but in computing values in the dynamic programming tables, ignores these edges.

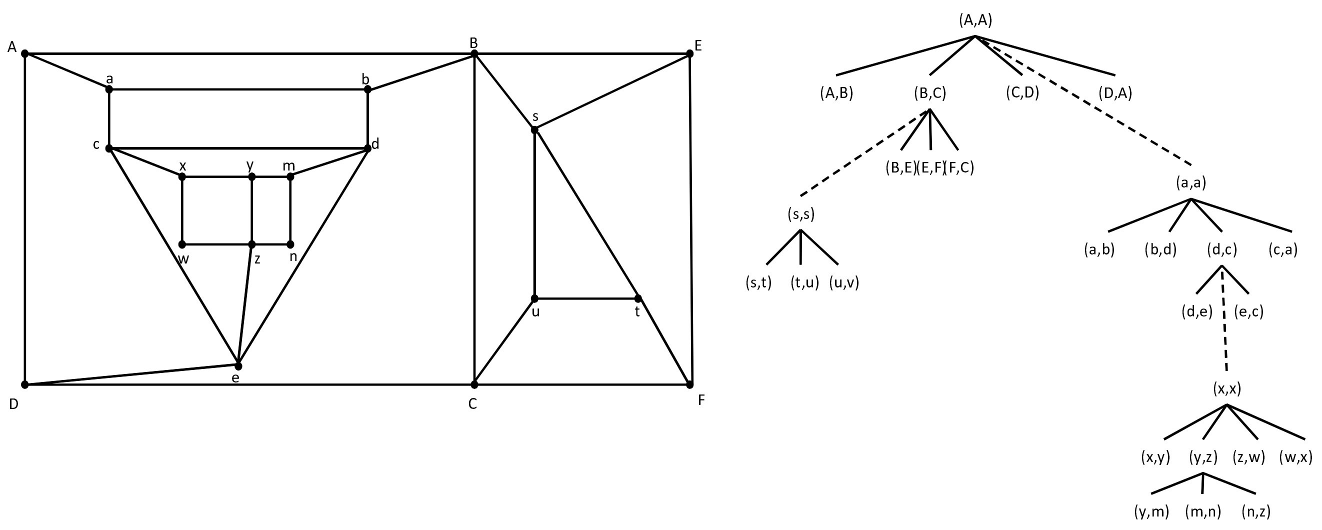

Figure 1.1: A k-outerplanar graph and its corresponding tree representation.

and $\{B, E, F, C\}$ are level 1 faces on the same level 1 component, $\{a, b, c, d\}$, $\{c, d, e\}$ are level 2 faces on a level 2 component, and $\{s, t, u\}$ is a level 2 face on other level 2 component. $\{x, y, z, w\}$ and $\{y, m, n, z\}$ are level 3 faces on the same level 3 component. The first step of the algorithm is constructing a tree representation $\bar{G}$ of G. Note that any level i component is an outerplanar graph, and hence we can use the method of Part I to construct a tree for it. As before, the leaves of a level i tree represent edges exterior to the level i component, the other vertices represent faces of the level i component, and the leaves from left to right represent a counterclockwise walk around the exterior edges of the component. After constructing a tree for each level i component, we construct $\bar{G}$ as follows: Starting from the level 1 tree, each face vertex of this tree can represent a face which contains a level 2 component. If so, add an edge from this vertex to the root of the tree that represents the level 2 component. Now, repeat this step for each level i component, and add edges from face vertices of level i tree representation of level i components, to the root of level $i + 1$ tree representations of level $i + 1$ components which are induced in corresponding faces of level i components. We refer the interested reader to Baker's original work [2] for a full presentation of the tree structure $\bar{G}$.

To summarize, the tree representation $\bar{G}$ of G will hold a vertex for each edge of G (and at times additional special "root" vertices corresponding to nodes of G). Each vertex (x, y) in $\bar{G}$ will represent a subgraph of G which is called $G(x, y)$ spanned by the so called *slice* of (x, y). In addition, with each vertex (x, y) in $\bar{G}$ we associate an array referred to as the *table* of (x, y) which holds information regarding the clique cover objective function of certain subgraphs of $G(x, y)$. Finally each vertex (x, y) in $\bar{G}$ is associated with a set of nodes in G called the *boundary* of G. We elaborate on the slice, table and boundary of vertices in $\bar{G}$ in Section 2.

Roughly speaking, once the tree $\bar{G}$ is given, the objective in [2] is to dynamically compute the objective function at hand (e.g., $\texttt{CC}$) in a *bottom up* manner from the leaves to the root. To that end, with each vertex in $\bar{G}$, [2] associates a table including useful information regarding the objective function being computed on the subgraph of G corresponding to v. There are two major operations that need to be addressed in the algorithm specified in [2]. The **merge** operation which takes as input an intermediate table T and the table of a tree node u and returns a "merged" table of the two. The **adjust** operation which takes as input a table T consisting of the merge of all the children of v, and returns a table corresponding to v. To apply Baker's algorithm to the objective function $\texttt{Obj}$ of our choice (here, we study the clique cover problem), we must show how to implement the subroutines **merge** and **adjust**. It is not hard to verify that to prove that one can implement the **merge** operation, it suffices to present an algorithm that takes as input two induced subgraphs G_1 and G_2 of G with intersection B of size $\leq k$ and the solutions $\texttt{Obj}(G_1 \setminus B')$, $\texttt{Obj}(G_2 \setminus B')$ for each $B' \subseteq B$; and returns a solution for $\texttt{Obj}(G_1 \cup G_2)$. Implementing this operation for the clique cover objective follows from Property 3.2.4.

Similarly, for the **adjust** operation as presented in [2], it suffices to present an algorithm which takes as input a k outerplanar graph G that includes vertices x and y on the outer face of G, but does not include the edge (x, y) (also on the outer face of G), and additional solutions to $\texttt{Obj}(G')$ for $G' \subset G$ that are stored in the tables computed recursively by the algorithm, and returns a solution for $\texttt{Obj}(G + e)$. The implementation of this latter tasks for the clique cover objective function is highly non-trivial and is the main contribution of this part of the thesis. This implementation does not follows directly from [2], and to the best of our knowledge, has not been addressed in the past.

```
**Procedure** table($v$)
    Let $v = (x, y)$ be a vertex on a level $i$ component
    **if** $v$ **is a face vertex and** $face(v)$ **encloses no level** $i + 1$ **component**
        $T = table(u)$ where $u$ is the leftmost child of $v$
        for each other child $c$ of $v$ from left to right
            $T = merge(T, table(c))$
        return adjust(T);
    **else if** $v$ **is a face vertex and** $face(v)$ **encloses a level** $i + 1$ **component** $C$
        return $adjust(contract(table(root(\bar{C}))))$
    **else if** $v$ **is a level** $1$ **leaf**
        return a table representing the edge $(x, y)$
    **else**   **/*** $v$ **is a level** $i$ **leaf,** $i > 1$ ***/**
        Let $f$ be the level $i$ face enclosing the component for $v$
        Let the labels of the children of $vertex(f)$ be $(z_1, z_2)$, $(z_2, z_3) \ldots (z_m, z_{m+1})$
        /* find $z_p$ - a point between nodes adgacent to $x$ and nodes adgacent to $y$ */
        **if** $y$ is adjacent to some $z_r$, $LB(v) \leq r \leq RB(v)$
            let $p$ be the least such $r$
        **else** $p = RB(v)$
        $T = create(v, p)$
        /* extend the leftmost $p$ tables to include $x$ and edges from $x$ to $z_r$, $r \leq p$, and merge with $T$ */
        $j = p - 1$
        **while** $j \geq LB(v)$
            $T = merge(extend(x, Table(u_j), T))$, $u_j$ is the $j_{th}$ child of $vertex(f)$
            $j = j - 1$
        /* extend the remaining tables to include $y$ and edges from $y$ to $u_r$, $r \geq p$, and merge with $T$ */
        $j = p$
        **while** $j \leq RB(v)$
            $T = merge(extend(y, Table(u_r), T))$, $u_j$ is the $j_{th}$ child of $vertex(f)$
            $j = j + 1$
        return $(T)$
```

Figure 1.2: The algorithm `table` from [2].

Chapter 2

The clique cover objective function for k - outerplanar graphs

In this section, we modify Baker's algorithm to compute the clique cover objective for k-outer planner graphs. For constant k our algorithm will run in time polynomial in the input graph size $|G|$. More specifically, our running time will be $n^{O(k)}$ for graphs G with n vertices.

Similar to the outer planer case studied in Part I, the major modification to Baker's algorithms lies in the details of the "merge" and "adjust" operations. Roughly speaking, the major modification needed to adapt to the clique cover objective function is in the adjust operation in which for a subgraph G' of G that includes the vertices x, y on its outer face, but does not include the edge (x, y), we seek to compute $\texttt{CC}(G' + (x, y))$. As noted in Claim 3.2.2, to compute $\texttt{CC}(G' + (x, y))$ we need to use the value of $\texttt{CC}(G' \setminus \{x, y, z\})$ for any vertex z that closes a clique with x, y. In the outerplanar case, the value $\texttt{CC}(G' \setminus \{x, y, z\})$ was stored in tables computed before the task at hand of computing $\texttt{CC}(G' \cup (x, y))$. There are two major differences between the k-outerplanar case, and the outerplanar one. In the outerplanar case, there may only be a single vertex z that closes a clique with x, y, and in such a case the information $\texttt{CC}(G' \setminus \{x, y, z\})$ is obtainable due to the fact that the vertex z sits on the "boundary" of the slice at hand when computing the adjust operation. In the k-outerplanar case, there may be several vertices z that close a clique with x, y. In addition, the information $\texttt{CC}(G' \setminus \{x, y, z\})$ for such vertices is no longer accessible via the tables (before our modification) computed by Baker's algorithm.

To overcome the need to access $\texttt{CC}(G' \setminus \{x, y, z\})$ in the adjust operations, we enhance the original tables suggested by Baker significantly. In the original tables, each tree vertex will have a set of *boundary nodes B* and there is an entry for each subset B' of B. In these entries the value $\texttt{CC}(G' \setminus B)$ is stored. In our modified tables, we will carefully define a set system denoted by the term $OuterSubsets(G')$, and our tables will include the value of $\texttt{CC}(G' \setminus B' \setminus S')$ for each subset B' of boundary nodes and each subset S' in the set system $OuterSubsets(G')$. The construction of the set system $OuterSubsets(G')$ and its properties will be defined in detail below. As we will show, this addition to the tables we define will allow us to use the value of $\texttt{CC}(G' \setminus \{x, y, z\})$ when it is needed.

The outline of this chapter is as follows: we first define in detail several notions from [2] that will set the foundations for our enhanced analysis. We start by discussing the notion of *slices* and the notion of an "execution tree" referred to as G_{alg}. We then define the notion of *boundary nodes* and the new notion of *outer nodes* used in this work. Finally we define the enhanced tables that will be used in this work, and turn to study the merge and adjust operations.

2.1 Slices and Boundary nodes

For each vertex in the tree representation $\bar{G}$, we define a corresponding *slice* (a subgraph of G). A slice is a subgraph of G, which is obtained by "cutting" its embedding by analogy with cutting a pie into slices. The slice of a level i leaf vertex (x, y) contains the edge (x, y) and nodes from lower level components. The *boundaries* of a slice of a level i vertex contain i nodes (for each boundary), one for each level from 1 to i, listed in order of decreasing level. We refer to two groups of boundary nodes, $LeftBoundaries(x, y)$ and $RightBoundaries(x, y)$, each boundary contains exactly i nodes. All the nodes in the embedding that exists inside the boudaries and the edge (x, y) are part of the slice. The slice of a level i face vertex contains also higher level nodes - the nodes enclosed by the face. To define the notion of slices recursively, in [2] every level i vertex in $\bar{G}$ is associated with a number of level $i-1$ vertices. This mapping associates each level $i-1$ vertex of $\bar{G}$ with a single level i vertex; thus implying that any two slices in G are either disjoint or one is included in the other. In more detail, let v be a tree vertex labeled (x, y):

- If v represents a level i face with no enclosed nodes, $i > 1$, its slice is the union of the slices of its children, plus the edge (x, y).
- If v represents a level i face enclosing a level $i+1$ component C, its slice is that of the root of the tree for C plus the edge (x, y).
- If v represents a level 1 leaf, its slice is the subgraph consisting of (x, y).
- If v represents a level i leaf, $i > 1$, then its slice includes (x, y), edges from x and y to level $i-1$ nodes, and the slices computed recursively for appropriate level $i-1$ vertices. Here, 'appropriate' is determined exactly by the level $i-1$ vertices associated in [2] to v as mentioned above.

For example, consider the vertex (y, z) on Figure 2.1. The set $LeftBoundaries(y, z)$ is equal to $\{y, d, B\}$, and the set $RightBoundaries(y, z)$ is equal to $\{z, e, C\}$. The slice of the vertex (y, z) contains the follwing edges (and the corresponding nodes): (y, m), (m, n), (n, z), (d, e), (z, e), (B, C), (B, E), (E, F), (F, C), (s, t), (t, u), (u, v), (u, C), (t, F), (s, E), and (s, E).

Note that the Slice of the root of G is simply the graph G itself. Once the tree $\bar{G}$ is given, the objective is to dynamically compute the clique cover in a bottom up manner from the leaves to the root.

2.2 The execution tree G_{alg} and its properties

In our analysis, it will be useful to refer to the *execution tree* of our algorithm. The execution tree, denoted G_{alg}, is the tree (consisting of the vertices of the standard tree representation $\bar{G}$) who's structure represents the recursive execution of algorithm `Table` given in Figure 1.2. Namely, the root of G_{alg} is the vertex in $\bar{G}$ representing the graph G, and the children of each vertex v in G_{alg} are the vertices $u \in \bar{G}$ for which there is a recursive call to `Table`(u) from procedure `Table`(v). It is shown in [2] that every vertex in $\bar{G}$ appears exactly once in G_{alg}.

Definition 2.2.1 (G_{alg} - execution tree)**.** *G_{alg} is obtained from Baker's tree representation $\bar{G}$ as follows: The root of G_{alg} is the same root as the root of $\bar{G}$. Let v be a tree vertex labeled (x, y) in G_{alg}:*

- *If v represents a level i face with no enclosed nodes, $i > 1$, its children in G_{alg} are the same children as v has in $\bar{G}$, namely, the vertices (x, a), (a, b), $\ldots$ (z, y), which represent a directed walk of exterior edges of v's component, in a counterclockwise direction from x to y.*
- *If v represents a level i face enclosing a level $i+1$ component C, its only child is the vertex which represents the root of the tree for C in $\bar{G}$.*
- *If v represents a level 1 leaf, it is a leaf on G_{alg}.*
- *If v represents a level i leaf, $i > 1$, then its children are the level $i-1$ vertices that are determined by the slice of v described above.*

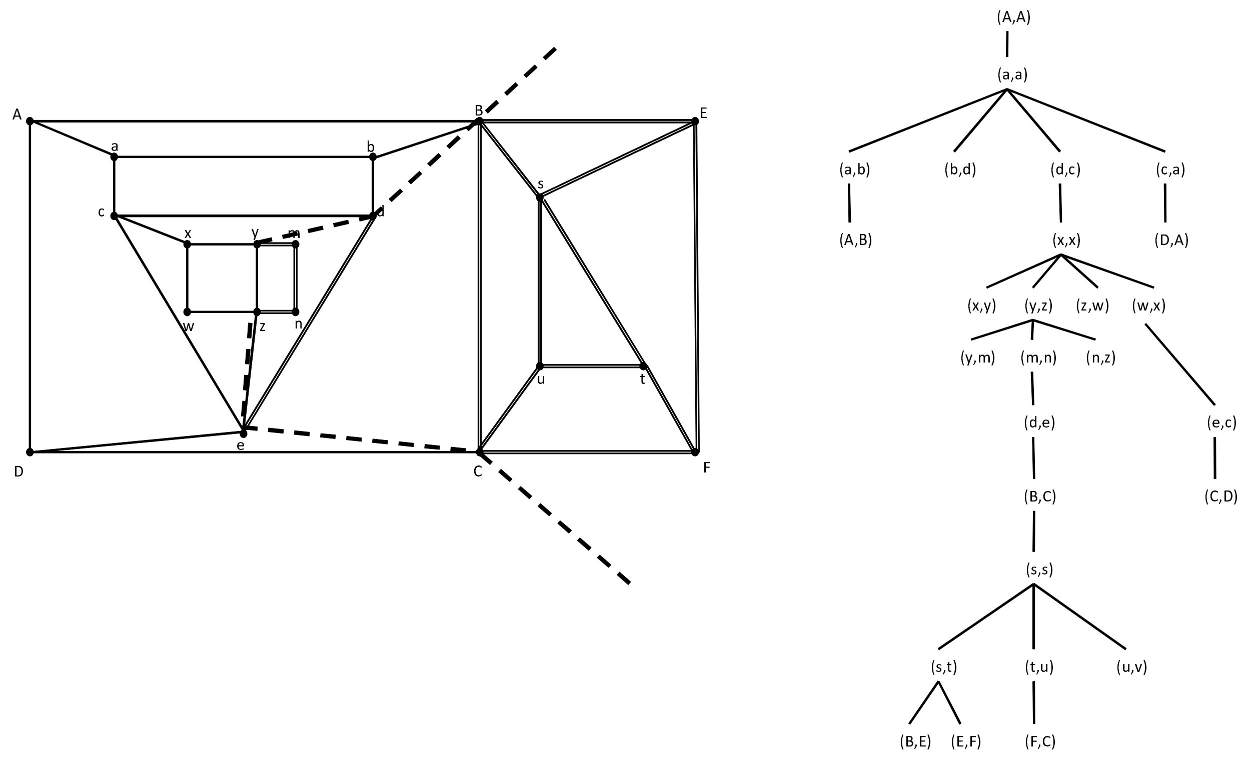

Figure 2.1: A k-outerplanar graph and its corresponding execution tree G_{alg}.

The tree G_{alg} represents the execution of the algorithm presented in Figure 1.2. In what follows we present some useful properties of G_{alg} that follow directly by the definition of G_{alg} (and by the analysis appearing in [2]).

Property 2.2.1. *Each vertex slice is obtained by its children slices in the following manner:*

- *If $v = (x, y)$ represents a level i face with no enclosed nodes, its slice is the merge of its children's slices in G_{alg} (corresponding to the "merge" operation), with the addition of the edge (x, y) (corresponding to the "adjust" operation).*
- *If v represents a level i face enclosing a level $i + 1$ component C, its slice is the slice of its sole child in G_{alg} with the addition of the edge (x, y) (corresponding to the "adjust" operation).*
- *If v represents a level 1 leaf, its slice is the subgraph consisting of (x, y) only.*
- *If v represents a level i leaf, $i > 1$, its slice is the merge of its children's slices in G_{alg} (according to the "merge" operation), with the addition of the edge (x, y) (corresponding to the "create" operation).*

Property 2.2.2. *If w_1 is a descendant of w_2 in G_{alg}, then $Slice(w_1) \subseteq Slice(w_2)$.*

Property 2.2.3. *If w_1 is a descendant of w_2, then during the algorithm's execution, the table for w_1 is computed before the table for w_2.*

Property 2.2.4. *Let α be a level i vertex in G_{alg} that represents a level i edge e_α in G, such that e_α is enclosed in a level j face F, $j < i$. Let β be a level j vertex in G_{alg} that represents a level j edge e_β in G, such that e_β is on the face F, and β is not the face vertex corresponding to F. Each vertex on the path $\beta, \ldots, \alpha$ in G_{alg} other than β is of level $> j$.*

For example, to better understand Property 2.2.4, consider the execution tree in Figure 2.1. Let $\alpha = (y, z)$, and $\beta = (d, e)$. The only edge in the path from β to α is (m, n). This edge is at the same level as α, and according to Property 2.2.4 each vertex on the path is of level $> j$, j is the level of β. A slight variation of Property 2.2.4 that we will also need later in our proof follows. One may follow the same example for the property below also.

Property 2.2.5. *Let α be a level i vertex in G_{alg} that represents a level i edge e_α in G, such that e_α is enclosed in a level j face F, $j = i - 1$. Let β be a level j vertex in G_{alg} that represents a level j edge e_β in G, such that e_β is on the face F, and β is not the face vertex corresponding to F. Each level i vertex on the path $\beta, \ldots, \alpha$ in G_{alg} is on the same level i component as α.*

2.3 The set $OuterNodes$

We now turn to define a set of nodes, referred to as $OuterNodes$ that will be central to the definition of the system $OuterSubsets$ discussed above.

Definition 2.3.1 ($OuterNodes(v)$)**.** *Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$ in the tree representation of Baker ($\bar{G}$). A node z is in $OuterNodes(v)$ iff: $z \in Slice(v)$, and z belongs to some level j component C_j, $j \leq i$, such that $\bar{C}_j$ is on the route from v to the root of $\bar{G}$. $z \in OuterNodes_j(v)$ if $z \in OuterNodes(v)$ and z belongs a level j component C_j.*

Notice that if v belongs to a level i component, there are exactly i components in the route from v to the root of $\bar{G}$ (namely, at most k components). For example, in Figure 1.1, the outer nodes of the slice of (x, y) are the nodes in the components corresponding to the following components in the tree representation $\bar{G}$: the component with the root (x, x), the component with the root (a, a), and the component with the root (A, A). Notice that the component with the root (s, s) is not on the route from (x, y) to the root (A, A), and thus it's nodes do not belong to $OuterNodes(v)$.

We are now ready to define the set system $OuterSubsets(v)$ for vertices v in the tree representation $\bar{G}$. As defined above, each and every level i vertex v will hold a corresponding set $OuterNodes(v) = \cup_{j \leq i} OuterNodes_j(v)$. From each set $OuterNodes_j(v)$, we will define 4 subsets referred to as $OuterNodes_{j_1}(v)$, $OuterNodes_{j_2}(v)$, $OuterNodes_{j_3}(v)$, and $OuterNodes_{j_4}(v)$. Roughly speaking, the sets $OuterNodes_{j_\ell}(v)$ are defined inductively, based on the recursion in Baker's algorithm. In the merge operation, certain sets $OuterNodes_{j_\ell}(v)$ are merged together, while in the adjust operation certain sets are *canceled* (i.e., *deleted*) or their content changed. The set system $OuterSubsets(v)$ now consists of any subset of nodes obtained by taking at most a single node from each subset $OuterNodes_{j_\ell}(v)$ where $j \leq i$ (recall that v is in level i). Notice that the size of $OuterSubsets(v)$ is bounded by n^{4k}, and thus can be implemented in the tables constructed throughout the algorithm.

Definition 2.3.2 ($OuterSubsets(v)$)**.** *Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$ in the tree representation of Baker ($\bar{G}$). An outer subset $S' \in OuterSubsets(v)$ is a subset from $OuterNodes(v)$ which contains at most 4 nodes from each level j component, $j \leq i$. The outer subsets are defined as follows: For each level j component, given 4 subsets of $OuterNodes_j(v)$: $OuterNodes_{j_1}(v)$, $OuterNodes_{j_2}(v)$, $OuterNodes_{j_3}(v)$, and $OuterNodes_{j_4}(v)$; we define $OuterSubsets(v)$ to consist of all subsets $S' \subseteq OuterNodes(v)$ which contain at most 1 node from each $OuterNodes_{j_\ell}(v)$, $j \leq i$, $\ell \leq 4$.*

As an example, if in Figure 1.1 we set $OuterNodes_{j_\ell}(v)$ to be $OuterNodes_j(v)$ for all j and ℓ, the following subsets are part of $OuterSubsets(x, y)$: $S_1 = \{x, d, A\}, S_2 = \{x, y, c, d, A, B\}, S_3 = \{x, y, d\}, S_3 = \{y\}$.

2.4 Table(v)

We now discuss the content and recursive construction of Table(v), the main data structure that enables the computation of the clique cover objective. The algorithm from [2] to compute Table(v) is given in Figure 1.2. As mentioned, the major components of the algorithm are the merge and adjust operations to be discussed in detail shortly.

Definition 2.4.1 ($Table(v)$). *For each slice corresponding to a vertex $v = (x, y)$ of $\bar{G}$, based on the algorithm of [2], we define a table for this slice which contains for each subset $S' \in OuterSubsets(v)$ and for each subset $B' \subseteq Boundaries(v)$, the value of $\mathtt{CC}(Slice(v) \setminus B' \setminus S')$. Note that the size of $Table(v)$ is at most $n^{4k}2^{2k}$.*

To compute Table(v), one needs to define the set $Boundaries(v)$ which represents the boundary of the slice of v, and the set $OuterNodes(v) = \cup OuterNodes_j(v)$. These sets are constructed recursively as described in the upcoming sections.

2.5 Computing Table(v) for level 1 leafs

The case when v is a level 1 leaf corresponding to (x, y) in G is the base of Baker's recursive algorithm. In our setting, the following sets will be defined for v: $Slice(v)$ is the edge (x, y), $Outernodes_{j_\ell}(v) = \phi$ for $j > 1$, $Outernodes_{j_\ell}(v) = \{x, y\}$ for $j = 1$, and the boundary of v includes the nodes x and y. Given these settings, the content of Table(v) follows.

2.6 Computing Table via the merge operation

In Figure 1.2 we present the recursive structure of Baker's algorithm. In what follows we will specify the *merge* operation. In the merge operation one takes two subgraphs $G_1 = Slice(v_1)$ and $G_2 = Slice(v_2)$ with at most k common nodes for which their tables are known, and returns the table corresponding to their union $G_T = G_1 \cup G_2$. To define the table of a union of 2 subgraphs we first specify the sets $OuterNodes_{j_\ell}(G_T)$. Namely, for each level j component, we define $OuterNodes_{j_\ell}(G_T) = OuterNodes_{j_\ell}(G_1) \cup OuterNodes_{j_\ell}(G_2)$. Now, $OuterSubsets(G_T)$ is defined from $OuterNodes_{j_\ell}(G_T)$, $j \leq i$, $\ell \leq 4$, via Definition 2.3.2. For example, for a level j component, if the following are the subsets corresponding to G_1:

- $OuterNodes_{j_1}(G_1) = \{a, b, c, d\}$
- $OuterNodes_{j_2}(G_1) = \{a, b, c\}$
- $OuterNodes_{j_3}(G_1) = \{a\}$
- $OuterNodes_{j_4}(G_1) = \{b\}$

and the following are the subsets corresponding to G_2:

- $OuterNodes_{j_1}(G_2) = \{s, t, u, v\}$
- $OuterNodes_{j_2}(G_2) = \{s, t, u, v\}$
- $OuterNodes_{j_3}(G_2) = \{s, t\}$

- $OuterNodes_{j_4}(G_2) = \{\}$

then the following are the subsets corresponding to G_T:

- $OuterNodes_{j_1}(G_T) = \{a, b, c, d, s, t, u, v\}$
- $OuterNodes_{j_2}(G_T) = \{a, b, c, s, t, u, v\}$
- $OuterNodes_{j_3}(G_T) = \{a, s, t\}$
- $OuterNodes_{j_4}(G_T) = \{b\}$

In the case above, the set $OuterSubsets(G_T)$ includes several subsets, among them are: $S_1 = \{a, b\}, S_2 = \{b, s\}, S_3 = \{c, s, b\}, S_3 = \{d, c, s, b\}$.

We continue with the definition of the merge operation: as in the original work of Baker [2], the boundary set is defined by $Boundaries(G_T) = LeftBoundaries(G_1) \cup RightBoundaries(G_2)$. The table for G_T contains for each subset $S_T \in OuterSubsets(G_T)$ and for each subset $B_T \subseteq Boundaries(G_T)$, the value of $\texttt{CC}(G_T \setminus B_T \setminus S_T)$. If the graphs G_1 and G_2 were disjoint then the clique cover of their union is just the union of the corresponding clique covers, however, as the subgraphs share few nodes, the clique cover of their union might be smaller according to the claim below. Recall that in our setting G_1 and G_2 may share at most k nodes.

Claim 2.6.1. *Let G_1 and G_2 be induced subgraphs of G that have k boundary nodes in common. Let $Boundaries(G_T)$ and $OuterSubsets(G_T)$ be the boundary nodes and outer subsets of $G_T = G_1 \cup G_2$. One can compute for any $B_T \in Boundaries(G_T)$ and $S_T \in OuterSubsets(G_T)$ the value of $\texttt{CC}(G_T \setminus B_T \setminus S_T)$ given the tables of G_1 and G_2 which hold the values $\texttt{CC}(G_i \setminus B_i \setminus S_i)$ for every $B_i \in Boundaries(G_i)$ and $S_i \in OuterSubsets(G_i)$.*

Proof. By definition, each $S_T \in OuterSubsets(G_T)$ is composed of two subsets: S_1 and S_2, with S_i in the set $OuterSubsets(G_i)$. In addition, $Boundaries(G_T) \subseteq LeftBoundaries(G_1) \cup RightBoundaries(G_2)$. Thus, for each $S_T \in OuterSubsets(G_T)$, and each $B_T \subseteq Boundaries(G_T)$, the graph $(G_T \setminus B_T \setminus S_T)$ is the union of 2 subgraphs: $G'_1 = (G_1 \setminus B'_1 \setminus S'_1)$ and $G'_2 = (G_2 \setminus B'_2 \setminus S'_2)$. These subgraphs have (at most) k boundary nodes in common, and in addition for any B' on the common boundary of G'_1 and G'_2 the value of $\texttt{CC}(G'_1 \setminus B')$ and $\texttt{CC}(G'_2 \setminus B')$ appears in the tables corresponding to G_1 and G_2. We conclude that using Property 3.2.4 we can find $\texttt{CC}(G'_1 \cup G'_2)$ as desired. $\square$

All in all, we have shown that `Table` can be constructed recursively in the case of the merge operation. We now specifically address the case of level i leafs.

2.7 Computing `Table`(v) for a level i leaf

Let $v = (x, y)$ be a leaf vertex on a level i component $\bar{C}_i$. As described in Figure 1.2, in order to calculate the table for v, we merge certain level $i - 1$ tables with the edge (x, y) and additional edges from x and y to the level $i - 1$ nodes. Precisely following the line of analysis in [2], the basic idea for the construction of `Table`(v) is to create an initial table which includes the edge (x, y) and a node z from level $i - 1$, turn each relevant level $i - 1$ table into a level i table by adding the node x or y, and merging these tables together. Because of the planarity constraint, there is some level $i - 1$ node z such that all the nodes other than z that are adjacent to x are on one side of z, while all the nodes other than z adjacent to y are on the other. Only z can be adjacent to both x and y. Thus, the approach is to find z, use “create” to construct an initial level i table for a subgraph containing z, x, y, and then to extend the tables on one side using x and the tables on the other side using y. The “create” operation is discussed shortly. For example, in Figure 2.2, the slice of level 2 leaf vertex $v = (s, t)$ is given. In order to compute the table of (s, t), we create an initial table with the level 1 node E. That is, this subgraph contains the nodes (s, t, E). The next steps are merging this initial table with the tables of slices defined for (B, E) and (E, F).

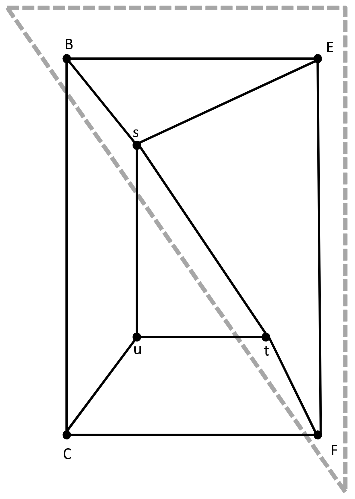

Figure 2.2: The slice of the leaf vertex (s, t).

2.7.1 Create

In the create operation we first find the level $i-1$ node z such that all the nodes other than z that are adjacent to x are on one side of z. The following sets are defined for the level i table for the subgraph containing x, y, z: $Outernodes_{j_\ell}(v) = \{x, y\}$ for $j = i$ and $Outernodes_{j_\ell}(v) = z$ for $j = i-1$. Otherwise, $Outernodes_{j_\ell}(v) = \phi$. The boundary of v includes the nodes x, y, z. Given these settings, the content of `Table`(v) follows and is summarized by the following claim.

Claim 2.7.1. *Let $v = (x, y)$ be a leaf vertex on a level i component $\bar{C}_i$. $OuterNodes_{i_1}(v) = OuterNodes_{i_2}(v) = OuterNodes_{i_3}(v) = OuterNodes_{i_4}(v) = OuterNodes_i(v) = \{x, y\}$.*

2.8 Computing `Table`(v) via the adjust operation

In the "adjust" operation one takes a planar subgraph G' with (outer face) nodes x and y but without the edge $e = (x, y)$ and computes the clique cover of $G' + e$ based on the clique covers of the graphs G', $G' - \{x\}$, $G' - \{y\}$, $G' - \{x, y\}$ and others. There are two kinds of "adjust" operations during the algorithm. A "simple adjust" operation takes place when trying to calculate the table of a level i face vertex with no enclosed nodes (in this case its slice is the merge of the slices of its children, plus the edge (x, y)). For example, in Figure 1.1, after merging the slices of vertices (y, m), (m, n), (n, z), the table of the slice of (y, z) is calculated using the "simple adjust" operation (adding the edge (y, z)). A "complex adjust" operation takes place when trying to calculate the table of a level i face vertex enclosing a level $i+1$ component C (its slice is that of the root of the tree for C including the edge (x, y)). For example, in Figure 1.1, after calculating the slice of the vertex (x, x), which is the root of a component enclosed by the face vertex (d, c), the table of the slice of (d, c) is calculated using the "complex adjust" operation (adding the edge (d, c)).

2.8.1 Outline of proof technique

The adjust operation is one of the most technically difficult parts in our analysis. Roughly speaking, the major challenge in our analysis is two fold and has an inductive flavor. First of all we need to prove that at the time we perform an adjust operation at vertex v of $\bar{G}$ corresponding to a certain subgraph G' (which is in fact, $Slice(v) \setminus$

$B' \setminus S'$ for certain sets B' and S', or is it $Slice(v)$ itself), the information in the tables already computed by the algorithm is rich enough to allow the computation of $\mathtt{CC}(G' \setminus \{x, y, z\})$ for vertices z that close a clique with the edge (x, y) of the adjust operation (here, we follow the discussion in the beginning of Chapter 2). Secondly, we need to prove that the information stored in $Table(v)$ after the adjust operation will suffice to support future adjust operations on vertices w in $\bar{G}$. Namely, that we store the value of $\mathtt{CC}(G' + (x, y) \setminus S)$ for a rich family of subsets S. This is exactly where we will use the definitions given for the subsets in $OuterSubsets(\cdot)$, or more specifically the sets $OuterNodes_{j_\ell}(\cdot)$.

For the discussion below, consider the case of a complex adjust operation (performed on a face vertex v that encloses some component). The case of a simple adjust is very similar. In this case, to compute the table of v, we use the table of u corresponding to the component enclosed in v.

We start by an example (a thought experiment) which will justify our definition of $OuterNodes_{j_\ell}(\cdot)$ and $OuterSubsets(\cdot)$. Consider the case in which in G' (corresponding to v) there is only one node z that closes a clique with the edge (x, y), and in addition, for u, all sets $OuterNodes_{j_\ell}(u)$ are equal to $OuterNodes_j(u)$ (we refer the reader to Definition 2.3.2 while noticing the subscript j_ℓ in the former expression as apposed to j in the latter). It is not hard to verify (we will do this later in detail) that in this case our first challenge above will be satisfied. Namely, that $z \in OuterNodes_{j_\ell}(u)$, and thus the set $\{z\}$ is in $OuterSubsets(u)$, which will lead to the fact that at the time of executing the adjust operation on v it holds that we are able to compute $\mathtt{CC}(G' \setminus \{x, y\} \setminus \{z\})$ as desired.

We are now left to define $OuterNodes_{j_\ell}(v)$ and accordingly the subsets in $OuterSubsets(v)$. Recall that $OuterSubsets(v)$ includes all subsets that consist of (at most) a single node from any subset $OuterNodes_{j_\ell}(v)$, and for every $S \in OuterSubsets(v)$ the table of v has to include $\mathtt{CC}(G' + (x, y) \setminus S)$. However, exactly as in the case of computing $\mathtt{CC}(G' + (x, y))$, to compute $\mathtt{CC}(G' + (x, y) \setminus S)$ we will also need to compute $\mathtt{CC}(G' \setminus S \setminus \{x, y, z\})$, which can be seen to essentially boil down to requiring that the set $S \cup \{z\}$ be included in $OuterSubsets(u)$. Assume that the vertex z is a level i vertex. A closer look at the system $OuterSubsets(u)$ reviles that it includes subsets with at most 4 level i nodes. Thus, if $S \cup \{z\}$ is to be included in $OuterSubsets(u)$, it must be the case that S includes at most 3 level i nodes. Namely, when defining $OuterNodes_{i_\ell}(v)$, one must rule out the case that $OuterSubsets(v)$ includes sets S with more than 3 level i nodes (in $Slice(u)$). This is done in a simple manner. Instead of defining $OuterNodes_{i_\ell}(v)$ to be equal to $OuterNodes_{i_\ell}(u)$ (which may be a very natural, but naive, choice), we will delete one of the sets $OuterNodes_{i_\ell}(u)$, in the sense that its counterpart corresponding to v will be defined as the empty set (e.g., we may define $OuterNodes_{i_1}(v) = \phi$ and $OuterNodes_{i_\ell}(v) = OuterNodes_{i_\ell}(u)$ for $\ell \geq 2$). This way the set system $OuterSubsets(v)$ will satisfy the requirement stated above, and it will still remain rich enough to satisfy future adjust operations. Or will it?

At this point in time we have that $OuterNodes_{i_1}(v)$ is empty but $OuterNodes_{i_\ell}(v)$ for $\ell \geq 2$ is still full. However, potentially, it may be the case that future adjust operations "zero-out" additional sets $OuterNodes_{i_\ell}(w)$ derived inductively from vertex v, to the extreme that $OuterNodes_{i_\ell}(w) = \phi$ for all ℓ. In such a case, adjust operations that rely on information from $Table(w)$ may not be able to complete the clique cover computations needed.

Indeed, in our construction and proof to follow, we deal with this problem and show that defining 4 sets of vertices from each level in $Slice(v)$ will suffice to satisfy all future adjust operations. This is essentially done on a case analysis based on the relative level of z when compared to that of v. More specifically, for a level i node v and a level j node z, we define a certain set $OuterNodes_{j_\ell}(v)$ that will be deleted if, when applying the adjust operation on v, the node z closes a clique with (x, y). This mapping is many to one, however, we show (based on the planar structure of G) that each set $OuterNodes_{j_\ell}(v)$ will be used (and thus deleted) only once.

2.8.2 Proof of the adjust operation

Before proceeding with the proof, we first define the sets $OuterNodes_{j_\ell}(v)$ and $OuterSubsets(v)$ for a vertex v after performing the 'adjust" operation. Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$. We will consider the two cases for the adjust operation: In the complex adjust case, the face which is represented by the vertex v encloses a component with a root represented by a vertex u. According to our definition of slices, $Slice(v) = Slice(u) + (x, y)$. That is, $Slice(v)$ is obtained by adding the edge (x, y) to $Slice(u)$. In the simple adjust case, the face which is represented by the vertex v has no enclosed nodes and $Slice(v)$ is obtained by merging all the slices corresponding to children of v, and then adding the edge (x, y). Let $Slice(u)$ be the subgraph which is obtained by merging the children's slices. Here, again, $Slice(v) = Slice(u) + (x, y)$. The sets $OuterNodes_{j_\ell}(v)$ are obtained from the sets $OuterNodes_{j_\ell}(u)$ in the following manner:

- For $j \leq i - 2$, and $1 \leq \ell \leq 4$: $OuterNodes_{j_\ell}(v) = OuterNodes_{j_\ell}(u)$.

- For $j = i-1$ the definition of $OuterNodes_{j_\ell}$ differs for different values of ℓ. Specifically, $OuterNodes_{j_1}(v) = OuterNodes_{j_1}(u)$ and $OuterNodes_{j_2}(v) = OuterNodes_{j_2}(u)$.

  If (x, y) doesn't close any clique with a node from level j component, then $OuterNodes_{j_3}(v) = OuterNodes_{j_3}(u)$ and $OuterNodes_{j_4}(v) = OuterNodes_{j_4}(u)$.

  If (x, y) closes a clique with one node z_1 from level j component, and $z_1 \in OuterNodes_{j_3}(u)$, then $OuterNodes_{j_3}(v) = \{z_1\}$. Otherwise, $z_1 \in OuterNodes_{j_4}(u)$ (we will prove later that any level $i - 1$ node z that closes a clique with (x, y) is included in $OuterNodes_{j_3}(u)$ or in $OuterNodes_{j_4}(u)$) and $OuterNodes_{j_4}(v) = \{z_1\}$.

  If (x, y) closes a clique with two nodes z_1, z_2 from level j component (we will prove in Claim 2.8.2 that follows that there are only 2 nodes from level $i - 1$ component that can close a clique with (x, y)), then according to Claim 2.8.1 (that follows) one of the following holds:

  1. $z_1 \in OuterNodes_{j_3}(u)$ and $z_2 \in OuterNodes_{j_4}(u)$. In this case we set $OuterNodes_{j_3}(v) = \{z_1\}$ and $OuterNodes_{j_4}(v) = \{z_2\}$.
  2. $z_1 \in OuterNodes_{j_4}(u)$ and $z_2 \in OuterNodes_{j_3}(u)$. In this case we set $OuterNodes_{j_4}(v) = \{z_1\}$ and $OuterNodes_{j_3}(v) = \{z_2\}$.

  In other words, if (x, y) closes a clique with a node from a level j component, then $OuterNodes_{j_3}(u)$ and / or $OuterNodes_{j_4}(u)$ do not appear in the table of v. Rather $OuterNodes_{j_3}(v)$ and $OuterNodes_{j_4}(v)$ are set to include the node that closes a clique with (x, y). Otherwise, $OuterNodes_{j_3}(v) = OuterNodes_{j_3}(u)$ and $OuterNodes_{j_4}(v) = OuterNodes_{j_4}(u)$. The sets $OuterNodes_{j_3}(u)$ and $OuterNodes_{j_4}(u)$ contain nodes z from a level $i-1$ component (the component which encloses v), which potentially may close a clique with the edge (x, y). Removing these sets from $\texttt{Table}(v)$, and the appropriate outer subsets to be defined in $OuterSubsets(v)$, will allow us to find for each such z, and for each subset $S \in OuterSubsets(v)$, the clique cover value of $Slice(v) \setminus (S \cup \{z\})$.

- For $j = i$, as before, the definition of $OuterNodes_{j_\ell}$ differs for different values of ℓ. Specifically, for $\ell = 1$, $OuterNodes_{j_1}(v) = OuterNodes_{j_1}(u)$. For $\ell = 2$ we set $OuterNodes_{j_2}(v) = \{x, y\}$, meaning that $OuterNodes_{j_2}(u)$ is removed and does not appear in $\texttt{Table}(v)$, rather the nodes $\{x, y\}$ are added. The set $OuterNodes_{j_2}(u)$ is a set that contain nodes z from the same component of v, which may potentially close a clique with the edge (x, y). As in the case above, removing this set will allow us the necessary slackness in the definition of $OuterSubsets(v)$. In addition, we set $OuterNodes_{j_3}(v) = OuterNodes_{j_4}(v) = \{x, y\}$.

- For $j = i+1$, we set $OuterNodes_{j_1}(v) = \phi$, meaning that $OuterNodes_{j_1}(u)$ does not appear in $\texttt{Table}(v)$. The set $OuterNodes_{j_1}(u)$ is a set that contain nodes z from a level $i + 1$ component (the component which is enclosed by the face represented by v), which may potentially close a clique with the edge (x, y). As in the cases above, removing this set will allow us the necessary slackness in the definition of $OuterSubsets(v)$. In addition, we set $OuterNodes_{j_\ell}(v) = OuterNodes_{j_\ell}(u)$ for $\ell \geq 2$.

The sets $OuterSubsets(v)$ are obtained from the sets $OuterNodes_{j_\ell}(v)$ as defined in Definition 2.3.2. In the following claim, we show that for each node z that closes a clique with (x, y), it holds that z belongs to a certain set $OuterNodes_{j_\ell}(u)$.

Claim 2.8.1. *Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$. Let $Slice(u)$ be the subgraph of the component enclosed by v (represented by u) in the complex adjust case, and the subgraph which is obtained by merging the children of v in the simple adjust case. Let $G_v = Slice(v)$. Let $G_u = Slice(u) = Slice(v) \setminus (x, y)$. For each node $z \in Slice(u)$ that closes a clique with (x, y), the following states hold:*

- *If $z \in C_j,\ j = i + 1$ (z in the component which is enclosed in the face represented by (x, y)), then $z \in OuterNodes_{j_1}(u)$.*
- *If $z \in C_j,\ j = i$ (z in the same component of (x, y)), then $z \in OuterNodes_{j_2}(u)$.*
- *If $z \in C_j,\ j = i - 1$ (z in the component which encloses the face represented by (x, y)), then $z \in OuterNodes_{j_3}(u)$ or $z \in OuterNodes_{j_4}(u)$.*
- *If there are 2 nodes $z_1, z_2 \in C_j,\ j = i - 1$ (z_1, z_2 in the component which encloses the face represented by (x, y)) that close a clique with (x, y), then one of the following holds: (a) $z_1 \in OuterNodes_{j_3}(u)$ and $z_2 \in OuterNodes_{j_4}(u)$, or (b) $z_1 \in OuterNodes_{j_4}(u)$ and $z_2 \in OuterNodes_{j_3}(u)$.*

Note that these are the only cases in which z can close a clique with (x, y).

Proof. We prove our claim in an inductive manner on the execution of Baker's algorithm. Namely, based on the structure of G_{alg}. We consider the following cases:

- $z \in C_j,\ j = i+1$ (z is in the component which is enclosed in the face represented by (x, y)). In this case, u is a vertex of level $i + 1$. As $z \in Slice(u)$, it holds that there exists a path $w_1, w_2, \ldots, w_r$ in G_{alg} such that (a) $w_r = u$, (b) w_1 is the "first" vertex in G_{alg} that includes z in it's slice. That is, all children of w_1 in G_{alg} do not include z in their slice. It follows that w_1 is a vertex that represents an edge incident to z (i.e, (z, a) or (a, z)), and it's a level i leaf (otherwise, if w_1 is a face vertex, then there must be another leaf vertex incident to z), (c) for all vertices w_s on the path it holds that $z \in Slice(w_s)$, (d) the table for w_s is computed before the table for w_{s+1}. We show by induction on s that $z \in OuterNodes_{j_1}(w_s)$. Notice that (c) and (d) are properties of G_{alg}. For the base case, as w_1 represents an edge incident to z in G, and it's a level $j (= i + 1)$ leaf, it holds that $z \in OuterNodes_j(w_1) = OuterNodes_{j_1}(w_1)$, according to Claim 2.7.1.

  Assume the assertion holds for a certain value of s. To prove the assertion for w_{s+1}, notice that the table for w_{s+1} is defined by one (or a combination) of the three following possibilities: the merge operation (including the vertex w_s), by creating a table for a leaf vertex (if w_{s+1} is a leaf vertex), or by the adjust operation on w_s as described above. In the case of a merge operation, as $OuterNodes_{j_1}(w_s) \subseteq OuterNodes_{j_1}(w_{s+1})$ it follows that $z \in OuterNodes_{j_1}(w_{s+1})$. In the case of a table creation for a leaf vertex, as the set $OuterNodes_{j_1}(w_{s+1})$ is obtained by merging certain sets including the set $OuterNodes_{j_1}(w_s)$ it follows that $z \in OuterNodes_{j_1}(w_{s+1})$. For the case of adjust, assume by contradiction that $z \notin OuterNodes_{j_1}(w_{s+1})$. That is, we delete $OuterNodes_{j_1}(w_s)$ when performing adjust on w_{s+1}. Thus, w_{s+1} is a level i face vertex that represents a face which encloses the component that includes z. This is a contradiction to the fact that $v = (x, y)$ is the only level i face vertex that encloses the component which includes z.

- $z \in C_j,\ j = i$ (z in the same component of (x, y)). In what follows, in a slight change of notation, if v is subject to a complex adjust, let w be the root of the component enclosed by v. Similarly, if v is subject to a simple adjust, let w be a child of v in the tree representation for which $z \in Slice(w)$. Notice that $x, y, z \in C_j$ and thus x, y, z is a triangle.

As $z \in Slice(w)$, it holds that there exists a path $w_1, w_2, \ldots, w_r = w$ in G_{alg} such that (a) w_1 is a level i vertex represented by (x,z) or (z,y) (represents the edges (x,z) or (z,y)) , (b) for all vertices w_s on the path it holds that $z \in Slice(w_s)$, (c) the table for w_s is computed before the table for w_{s+1}. Notice that each vertex (x,z) or (z,y) can be a level i leaf, or a face vertex.

We show by induction on s that $z \in OuterNodes_{j_2}(w_s)$. Assume w.l.o.g that $w_1 = (x,z)$. For the base case, if w_1 is a leaf, we have by definition that $z \in OuterNodes_j(w_1) = OuterNodes_{j_2}(w_1)$, according to Claim 2.7.1. Otherwise, (x,z) is a face vertex, and by definition after calculating $Table(x,z)$ we perform adjust on (x,z), and $OuterNodes_{j_2}(x,z) = \{x,z\}$. Thus, $z \in OuterNodes_{j_2}(w_1)$.

Assume the assertion holds for a certain value of s. To prove the assertion for w_{s+1}, notice as before that the table for w_{s+1} is defined by the merge operation (including the vertex w_s), by creating a table for a leaf vertex (if w_{s+1} is a leaf vertex), or by the adjust operation on w_s as described above. Exactly as in the previous case, in the case of a merge operation or a table creation for a leaf vertex, we have $OuterNodes_{j_2}(w_s) \subseteq OuterNodes_{j_2}(w_{s+1})$ and thus $z \in OuterNodes_{j_2}(w_{s+1})$. In the case of adjust, if w_{s+1} is of level at least $i+1$, then in the adjust operation the set $OuterNodes_{j_2}(w_{s+1})$ equals $OuterNodes_{j_2}(w_s)$ by definition, and thus by induction $z \in OuterNodes_{j_2}(w_{s+1})$. If w_{s+1} is of level i it must be the case that $w_{s+1} = w_1 = (x,z)$ or (z,y) and this is the base case. In order to prove this assertion, assume, in cotradiction, that $w_{s+1} \neq w_1$. Notice that w_{s+1} is a level i face vertex (otherwise, there is no need of adjust operation), and $z \in Slice(w_{s+1})$. Moreover, w_{s+1} is descendant of v in G_{alg} (it's in the path $w_1, w_2, \ldots, w$ and w is a child of v in G_{alg}). (x,z) and (z,y) (and maybe (z,z)) are the only level i children of v in $\bar{G}$. Thus, w_{s+1} is descendant of (x,z) or (z,y) in $\bar{G}$, (and in G_{alg} too), and according to the slices construction, $z \notin Slice(w_{s+1})$, a contradiction.

- $z \in C_j,\ j = i-1$ (z in the component which encloses the face represented by (x,y)). Similar to the previous case, if v is subject to a complex adjust, let w be the root of the component enclosed by v. If v is subject to a simple adjust, let w be a child of v in the tree representation for which $z \in Slice(w)$.

  As $z \in Slice(w)$, it holds that there exists a path $w_1, w_2, \ldots, w_r = w$ in G_{alg} such that (a) w_1 is a vertex that represents an edge incident to z (i.e, (z,a) or (a,z)) in G_{alg} (w_1 is a level $i-1$ leaf or a level $i-1$ face vertex), (b) no vertex above w_1 in G_{alg} is incident to z, (c) for all vertices w_s on the path it holds that $z \in Slice(w_s)$, (d) the table for w_s is computed before the table for w_{s+1}. We show by induction on s that $z \in OuterNodes_{j_3}(w_s)$ or $z \in OuterNodes_{j_4}(w_s)$. For the base case, as w_1 represents an edge incident to z in G, it holds that $z \in OuterNodes_{j_3}(w_1) = OuterNodes_{j_4}(w_1)$. If w_1 is a level $j (= i-1)$ leaf then according to Claim 2.7.1 $OuterNodes_{j_\ell}(w_1) = OuterNodes_{j_3}(w_1) = OuterNodes_{j_4}(w_1) = \{a,z\}$. If w_1 is a level $j (= i-1)$ face vertex then $OuterNodes_{j_3}(w_1)$ and $OuterNodes_{j_4}(w_1)$ contain the nodes represented by w_1 (i.e, (a,z)). Here, we use the definitions of $OuterNodes_{j_\ell}$ given in Section 2.8.2 for $j = i-1$.

  Assume the assertion holds for a certain value of s. To prove the assertion for w_{s+1}, notice that the table for w_{s+1} is defined by the merge operation (including the vertex w_s), by creating a table for a leaf vertex (if w_{s+1} is a leaf vertex), or by the adjust operation on w_s as described above. In the case of a merge operation, as $OuterNodes_{j_3}(w_s) \subseteq OuterNodes_{j_3}(w_{s+1})$ it follows that $z \in OuterNodes_{j_3}(w_{s+1})$. The same holds for $OuterNodes_{j_4}(w_{s+1})$. In the case of a table creation for a leaf vertex, as $OuterNodes_{j_3}(w_{s+1})$ is created by merging certain subsets including $OuterNodes_{j_3}(w_s)$ it follows that $z \in OuterNodes_{j_3}(w_{s+1})$. The same holds for $OuterNodes_{j_4}(w_{s+1})$.

  In the case of adjust, if w_{s+1} is of level $\neq i$ or $\neq i-1$, then in the adjust operation the set $OuterNodes_{j_3}(w_{s+1})$ equals $OuterNodes_{j_3}(w_s)$, and the set $OuterNodes_{j_4}(w_{s+1})$ equals $OuterNodes_{j_4}(w_s)$, by definition ($j = i-1$, and thus, only if w_{s+1} is of level i or $i-1$, $OuterNodes_{j_3}(\cdot)$ and $OuterNodes_{j_4}(\cdot)$ can be changed). Thus, it follows that $z \in OuterNodes_{j_3}(w_{s+1})$ or $z \in OuterNodes_{j_4}(w_{s+1})$.

If w_{s+1} is of level $i-1$, then $w_{s+1} = w_1$, and this is the base case. This follows from Property 2.2.4. We are in the case that z is in the component which encloses the face represented by (x, y). Thus, since w_1 represents an edge incident to z in G, it holds that w_1 is on the face that encloses the edge in G represented by w. According to Property 2.2.4, since w_1 is a level $i-1$ vertex, Each vertex on the path $w_1, \ldots, w$ in G_{alg} other than w_1 is of level $> i-1$. Thus, if w_{s+1} is a level $i-1$ vertex in this path, it must be the case that $w_{s+1} = w_1$.

If w_{s+1} is of level i, and it's corresponding nodes in G don't close a clique with any level $i-1$ nodes z' then $OuterNodes_{j_3}(w_{s+1}) = OuterNodes_{j_3}(w_s)$, and $OuterNodes_{j_4}(w_{s+1}) = OuterNodes_{j_4}(w_s)$. So again, $z \in OuterNodes_{j_3}(w_{s+1})$ or $z \in OuterNodes_{j_4}(w_{s+1})$.

Finally, we are in the case that w_{s+1} is of level i and there exist level $i-1$ nodes z' that close a clique with the nodes in G corresponding to w_{s+1}. We show in Claim 2.8.2 below that (a) there are at most two level $i-1$ nodes z_1 and z_2 in G that close a clique with the nodes in G corresponding to w_{s+1} and (b) that it cannot be the case that both z_1 and z_2 differ from z. This implies that either $z_1 = z$, or $z_2 = z$, or there is only a single node in G that closes a clique with the nodes in G corresponding to w_{s+1}. In the first and second case one of $OuterNodes_{j_3}(w_{s+1})$ or $OuterNodes_{j_4}(w_{s+1})$ will be set by definition to $\{z\}$. In the third case, assume w.l.o.g. that $z_1 \neq z$ and $OuterNodes_{j_3}(w_s)$ contains z (by induction), we set $OuterNodes_{j_3}(w_{s+1}) = OuterNodes_{j_3}(w_s)$ and $OuterNodes_{j_4}(w_{s+1}) = \{z_1\}$. In all three cases we will have $z \in OuterNodes_{j_3}(w_{s+1})$ or $z \in OuterNodes_{j_4}(w_{s+1})$.

- $z_1, z_2 \in C_j,\ j = i-1$ (z_1, z_2 in the component which encloses the face represented by (x, y)). z_1, z_2 close a clique with (x, y). In the previous case we proved that $z_\ell \in OuterNodes_{j_3}(u)$ or $z_\ell \in OuterNodes_{j_4}(u)$, $\ell \in \{0, 1\}$. There exist two paths $w_{1\ell}, w_{2\ell}, \ldots, w_{r\ell} = w$ in G_{alg} such that (a) $w_{1\ell}$ is a vertex that represents an edge incident to z_ℓ (i.e, (z_ℓ, a) or (a, z_ℓ)) in G_{alg} ($w_{1\ell}$ is a level $i-1$ leaf or a level $i-1$ face vertex), (b) no vertex above $w_{1\ell}$ in G_{alg} is incident to z_ℓ, (c) for all vertices $w_{s\ell}$ on the path it holds that $z_\ell \in Slice(w_{s\ell})$, (d) the table for $w_{s\ell}$ is computed before the table for $w_{(s+1)\ell}$.

  We show in the proof of Claim 2.8.2 that for each level i component C_i inside a level $i-1$ component C_{i-1} the following statement holds: Let (x, y) be an edge on C_i. Let z_1 and z_2 be nodes on C_{i-1} that close a triangle with (x, y). See Figure 2.3. Every other node on C_i must be enclosed by one of the triangles (x, y, z_1) or (x, y, z_2). This follows directly from planarity properties. Thus, all the edges represented by level i vertices descendant of v in G_{alg} are enclosed by the triangle x, y, z_1 or x, y, z_2. Assume w.l.o.g that they are enclosed by x, y, z_1. It follows that all the level i vertices descendant of v in G_{alg} in the paths $w_{1\ell}, w_{2\ell}, \ldots, w_{r\ell} = w$ can close a clique only with z_1. According to the inductive proof above, for each level i vertex w_{s+1} in one of the paths, if we perform the adjust operation, then one of the sets $OuterNodes_{j_3}(w_{s+1})$ or $OuterNodes_{j_4}(w_{s+1})$ is set by definition to $\{z_1\}$, and the other set is not changed. It follows that if $z_1 \in OuterNodes_{j_3}(u)$ then $z_2 \in OuterNodes_{j_4}(u)$, and if $z_1 \in OuterNodes_{j_4}(u)$ then $z_2 \in OuterNodes_{j_3}(u)$.

$\square$

We now state and prove the assertion needed in the third case of the proof of Claim 2.8.1.

Claim 2.8.2. *(a) Let v be a level i face vertex of the execution tree G_{alg}. There are at most two level $i-1$ nodes z_1 and z_2 in G that close a clique with the nodes in G corresponding to v. (b) Let z be a level $i-1$ node in G that closes a clique with the nodes in G corresponding to v. Let w_1 be a vertex in G_{alg} that represents an edge in G with one end point which is z. Let w be any level i descendant of v in G_{alg} such that w is in the path $w_1, w_2, \ldots, v$ ($z \in Slice(w)$). There are at most two level $i-1$ nodes z'_1 and z'_2 in G that close a clique with the nodes in G corresponding to w, and it cannot be the case that both of them differ from z.*

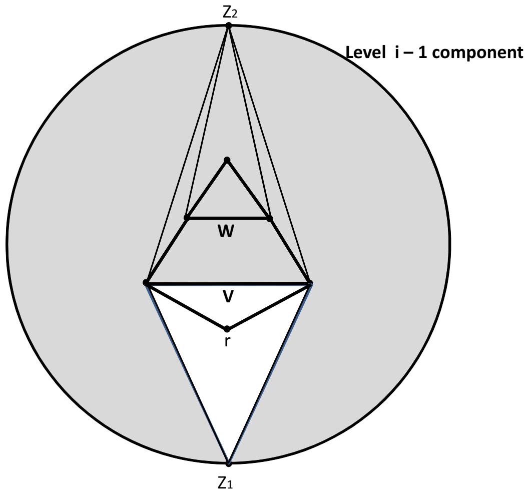

Figure 2.3: An illustration of the proof for Claim 2.8.2. Here, $v = (x, y)$.

Proof. (a) Each face vertex v in the execution tree G_{alg} is in fact an edge (x, y) in the graph G. Consider the face F that contains v (F is a level $i-1$ face). According to planarity constraint, there are at most 2 nodes on F that can close a triangle with x, y. Otherwise, there must be edges that intersect each other. For example, in Figure 2.3, v is a level i face vertex and z'_1, z'_2 are nodes in a level $i-1$ component. We can see that if v closes a triangle with z'_1 and z'_2, there are no other nodes on the level $i-1$ component that can close a triangle with v. (b) Let w_1 be a vertex in G_{alg} that represents an edge in G with one end point which is z, and let z close a clique with the nodes in G corresponding to v. It follows that w_1 is a level $i-1$ vertex representing an edge which is on the face that encloses v. According to Property 2.2.5, it holds that any other level i vertex on the path $w_1, w_2, \ldots, v$ is in the same component as v. Thus, v and w are face vertices on the same level i component. For each level i component C_i inside a level $i-1$ component C_{i-1} the following statement holds: Let (a, b) be an edge on C_i. Let c_1 and c_2 be nodes on C_{i-1} that close a triangle with (a, b). Every other node on C_i must be enclosed by one of the triangles (a, b, c_1) or (a, b, c_2). This follows directly from planarity properties.

Now, consider the nodes z'_1 and z'_2 in G that can potentially close a clique with the nodes in G corresponding to w. If the nodes in G corresponding to w close a clique with only one node the assertion holds. If the nodes in G corresponding to w close a clique with both nodes z'_1 and z'_2, then since v and w are on the same level i component, it holds that the nodes in G corresponding to v are enclosed by the triangle created by nodes in G corresponding to w and z'_1 or enclosed by the triangle created by nodes in G corresponding to w and z'_2. Assume, w.l.o.g, that the nodes in G corresponding to v are enclosed by the triangle created by nodes in G corresponding to w and z'_2. Thus, according to planarity constraint, if v closes a clique with a level $i-1$ node z, and the edges from v to z can not intersect the edges from w to z'_2, it must be the case that $z = z'_2$.

For example, in Figure 2.3, v is a level i face vertex that closes a clique with z'_1 and z'_2, and w is a level i face vertex enclosed by the triangle created by v and z'_2. We can see that the only level $i-1$ node that can close a clique with w is z'_2.

□

Using the definitions above we now prove that v can indeed compute its table from the information present in the tables computed so far.

Claim 2.8.3. *Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$. Let $Slice(u)$ be the subgraph of the component enclosed by v (represented by u) in the complex adjust case, and the subgraph which is obtained by merging the children of v in the simple adjust case. Let $G_v = Slice(v)$. Let $G_u = Slice(u) = Slice(v) \setminus (x, y)$ be an induced subgraph of G, that does not include the edge $e = (x, y)$. Let B_u, S_u be the boundary nodes and outer subsets of G_u respectively. Let B_v, S_v be the boundary nodes and outer subsets of G_v respectively. If for each $S'_u \in S_u$, for each $B'_u \subseteq B_u$, the values $\texttt{CC}(G_u \setminus B'_u \setminus S'_u)$ are known, then one can compute $\texttt{CC}(G_v \setminus B'_v \setminus S'_v) = \texttt{CC}(G_u + (x, y) \setminus B'_v \setminus S'_v)$ for each $S'_v \in S_v$, for each $B'_v \subseteq B_v$.*

Proof. In what follows, in a slight change of notation, let S_v and S_u be outer subsets of G_v and G_u respectively, such that for each $S'_v \in S_v$ it holds that $S'_v \cap \{x, y\} = \phi$, and for each $S'_u \in S_u$ it holds that $S'_u \cap \{x, y\} = \phi$. In other words, we will consider only outer subsets that don't contain the nodes $\{x, y\}$. We will show later that these outer subsets are sufficient for our proof. Let $G'_v = G_v \setminus B'_v \setminus S'_v$. Let $G'_u = G_u \setminus B'_v \setminus S'_v$. That is, we have to show that one can compute $\texttt{CC}(G_v \setminus B'_v \setminus S'_v) = \texttt{CC}(G'_v) = \texttt{CC}(G'_u + (x, y))$. For that purpose, we will use Claims 3.2.1 and 3.2.2 (of the Preliminaries). According to Claims 3.2.1 and 3.2.2, in order to calculate $\texttt{CC}(G'_v)$ we have to know the following values: $\texttt{CC}(G'_u \setminus \{x\})$, $\texttt{CC}(G'_u \setminus \{y\})$,and $\texttt{CC}(G'_u \setminus \{x, y\})$. We show in Claim 2.8.4 below that (a) $B_v = B_u$, and (b) $S_v \subset S_u$. Thus, for each $S'_v \in S_v$, there is a corresponding subset $S'_u \in S_u$, such that $S'_v = S'_u$. Similarly, for each $B'_v \in B_v$, there is a corresponding subset $B'_u \in B_u$, such that $B'_v = B'_u$. Thus, we can compute $\texttt{CC}(G'_u \setminus \{x\})$, $\texttt{CC}(G'_u \setminus \{y\})$,and $\texttt{CC}(G'_u \setminus \{x, y\})$ from the table of u. In order to prove that it is sufficient to consider outer subsets that don't contain the nodes $\{x, y\}$, recall that we have to compute the value of $\texttt{CC}(G_v \setminus B'_v \setminus S'_v)$. Notice that for each $S'_v \in S_v$, if $x \in S'_v$, then $\texttt{CC}(G_v \setminus B'_v \setminus S'_v) = \texttt{CC}(G_v \setminus (B'_v \cup x) \setminus (S'_v \setminus x))$ (The same holds for $y \in S'_v$). In other words, since $x, y \in Boundary(v) = Boundary(u)$, we can "move" the nodes x, y from S'_v to B'_v and thus we can compute the value of $\texttt{CC}(G_v \setminus B'_v \setminus S'_v)$ for outer subsets S'_v that don't contain the nodes x, y.

The main difference is regarding Claim 3.2.2: In order to find the clique cover after the "adjust" operation, we must know if there is a node z such that x, y, z close a triangle. There may be many nodes that can close a triangle with x, y, and we have to know the clique cover value of G_v without any of these nodes. Here we use the values we saved for $Table(u)$. That is, for each $S'_v \in S_v$ and $B'_v \subseteq B_v$, we have to find $\texttt{CC}(G'_u + (x, y)) = \texttt{CC}(G'_v)$. In order to add the edge (x, y), using Claim 3.2.2, we have to know for each $S'_v \in S_v$ and for each $B'_v \subseteq B_v$ the value of $\texttt{CC}(G'_u \setminus \{x, y, z\})$, for each z that closes a clique with (x, y). According to Claim 3.2.2, if there is no node z such that x, y, z close a clique, then $\texttt{CC}(G'_u + (x, y)) = \texttt{CC}(G'_u)$. If there is a node z such that x, y, z close a clique, and $\texttt{CC}(G'_u \setminus \{x, y, z\}) = \texttt{CC}(G'_u) - 2$ then $\texttt{CC}(G'_u + (x, y)) = \texttt{CC}(G'_u) - 1$, otherwise, $\texttt{CC}(G'_u + (x, y)) = \texttt{CC}(G'_u)$. Now, the only challenge is to prove that the value of $\texttt{CC}(G'_u \setminus \{x, y, z\})$ for each z that closes a clique with (x, y) is known. We consider three kinds of nodes that can close a clique with (x, y) (the first one is applicable only in the "complex adjust" case):

- $z \in C_j, j = i + 1$ (z in the component which is enclosed in the face represented by (x, y)). According to Claim 2.8.1, $z \in OuterNodes_{j_1}(u)$. In addition, according to definition of $OuterNodes$, $z \notin OuterNodes_{j_1}(v)$ since $OuterNodes_{j_1}(v) = \phi$. We show in Claim 2.8.4 below that (a) $B_v = B_u$, (b) $S_v \subset S_u$, and (c) Let $OuterNodes_{j_\ell}(v)$, $OuterNodes_{j_\ell}(u)$ be specific sets from $OuterNodes(v)$ and $OuterNodes(u)$ correspondingly. For each $S'_v \in S_v$, if $S'_v \cap OuterNodes_{j_\ell}(v) = \phi$ and $z \in OuterNodes_{j_\ell}(u)$, then $(S'_v \cup z)$ is an outer subset of u. In our case, $S'_v \cap OuterNodes_{j_1}(v) = \phi$ ($OuterNodes_{j_1}(v) = \phi$), and $z \in OuterNodes_{j_1}(u)$. According to Claim 2.8.4 it holds that $(S'_v \cup z)$ is an outer subset of u. Recall that for each $S'_u \in S_u$, for each $B'_u \subseteq B_u$, the values $\texttt{CC}(G_u \setminus B'_u \setminus S'_u)$ are known. Since $B_v = B_u$ and $(S'_v \cup z)$ is an outer subset of u, it holds that the value of $\texttt{CC}(G_u \setminus B'_v \setminus (S'_v \cup z))$ = $\texttt{CC}(G'_u \setminus \{z\})$ is known. Moreover, $\{x, y\} \subseteq Boundary(v)$ and we are only interested in B'_v that do not include $\{x, y\}$. Otherwise, G'_u is a subgraph that doesn't contain the edge (x, y) and thus there is no clique (x, y, z). Thus, the value of $\texttt{CC}(G_u \setminus (B'_v \cup \{x, y\}) \setminus (S'_v \cup z))$ = $\texttt{CC}(G'_u \setminus \{x, y, z\})$ is known, since $\texttt{CC}(G'_u \setminus \{z\}) = \texttt{CC}(G_u \setminus B'_v \setminus S'_v)$ is known for any subset of B_v, especially for $B'_v \cup \{x, y\}$.

- $z \in C_i$ (z in the same component of (x, y)). Similar to the previous case, $z \in OuterNodes_{j_2}(u)$, and $OuterNodes_{j_2}(v) = \{x, y\}$. As above by Claim 2.8.4, for each $S'_v \in S_v$, it holds that $(S'_v \cup z)$ is an outer subset of u, and $\texttt{CC}(G'_u \setminus \{z\})$ is known. Moreover, as before $\{x, y\} \subseteq Boundary(v)$ and we are studying B'_v that do not include $\{x, y\}$. Thus, the value of $\texttt{CC}(G_u \setminus (B'_v \cup \{x, y\}) \setminus (S'_v \cup z))$ = $\texttt{CC}(G'_u \setminus \{x, y, z\})$ is known. It follows that $\texttt{CC}(G'_v \setminus \{x, y, z\})$ is known.

- $z \in C_j, j = i-1$ (z in the component which encloses the face represented by (x, y)). $z \in OuterNodes_{j_3}(u)$ or $z \in OuterNodes_{j_4}(u)$. According to our definitions, one of the following holds: (a) $z \in OuterNodes_{j_3}(u)$ and $OuterNodes_{j_3}(v) = \{z\}$, (b) $z \in OuterNodes_{j_4}(u)$ and $OuterNodes_{j_4}(v) = \{z\}$. Assume w.l.o.g that $z \in OuterNodes_{j_3}(u)$ and $OuterNodes_{j_3}(v) = \{z\}$. For each $S'_v \in S_v$, it holds that $(S'_v \cap OuterNodes_{j_3}(v)) = \phi$ or $(S'_v \cap OuterNodes_{j_3}(v)) = \{z\}$. If $(S'_v \cap OuterNodes_{j_3}(v)) = \phi$, then $(S'_v \cup z)$ is an outer subset of u, and like the previous sections $\texttt{CC}(G'_u \setminus \{z\})$ is known. If $(S'_v \cap OuterNodes_{j_3}(v)) = \{z\}$, then $(S'_v \cup z) = S'_v$ and it is an outer subset of u. Thus, like the previous sections $\texttt{CC}(G'_u \setminus \{z\})$ is known. Moreover, as before $\{x, y\} \subseteq Boundary(v)$ and we are studying B'_v that do not include $\{x, y\}$. Thus, the value of $\texttt{CC}(G_u \setminus (B'_v \cup \{x, y\}) \setminus (S'_v \cup z)) = \texttt{CC}(G'_u \setminus \{x, y, z\})$ is known. It follows that $\texttt{CC}(G'_v \setminus \{x, y, z\})$ is known.

□

Claim 2.8.4. *Let $v = (x, y)$ be a vertex on a level i component $\bar{C}_i$. Let $Slice(u)$ be the subgraph of the component enclosed by v (represented by u) in the complex adjust case, and the subgraph which is obtained by merging the children of v in the simple adjust case. Let $G_v = Slice(v)$. Let $G_u = Slice(u) = Slice(v) \setminus (x, y)$ be an induced subgraph of G, that does not include the edge $e = (x, y)$. Let B_u, S_u be the boundary nodes and outer subsets of G_u respectively, such that for each $S'_u \in S_u$ it holds that $\{x, y\} \notin S'_u$. Let B_v, S_v be the boundary nodes and outer subsets of G_v respectively, such that for each $S'_v \in S_v$ it holds that $\{x, y\} \notin S'_v$. The following statements hold: (a) $B_v = B_u$ (b) $S_v \subset S_u$ (c) Let $OuterNodes_{j_\ell}(v)$, $OuterNodes_{j_\ell}(u)$ be specific sets from $OuterNodes(v)$ and $OuterNodes(u)$ correspondingly. For each $S'_v \in S_v$, if $S'_v \cap OuterNodes_{j_\ell}(v) = \phi$ and $z \in OuterNodes_{j_\ell}(u)$, then $(S'_v \cup z)$ is an outer subset of u.*

Proof. (a) $B_v = B_u$ This statement holds according to our definitions. $Slice(u) = Slice(v) \setminus (x, y)$. In the adjust operation we only add to $Slice(u)$ the edge (x, y), and there are no changes in the boundary nodes. (b) $S_v \subset S_u$. The sets $OuterNodes_{j_\ell}(v)$ are obtained from the sets $OuterNodes_{j_\ell}(u)$ as defined above. Following are the possibilities for $OuterNodes_{j_\ell}(v)$:

- $OuterNodes_{j_\ell}(v) = OuterNodes_{j_\ell}(u)$.
- $OuterNodes_{j_\ell}(v) = \phi$.
- $OuterNodes_{j_\ell}(v) = \{z_1\}$, where z_1 is a level $j-1$ node that closes a clique with (x, y). According to Claim 2.8.2, $z_1 \in OuterNodes_{j_\ell}(u)$.
- $OuterNodes_{j_\ell}(v) = (x, y)$.

From all the possibilities above we can conclude that $\{OuterNodes_{j_\ell}(v) \setminus \{x, y\}\} \subset \{OuterNodes_{j_\ell}(u) \setminus \{x, y\}\}$. As outer subsets are obtained from outer nodes directly, it holds that $S_v \subset S_u$.

(c) By definition, $S'_v \in OuterSubsets(v)$ is a subset which contain at most 1 node from each $OuterNodes_{j_\ell}(v)$, $j \leq i, \ell \leq 4$. Moreover, for each $OuterNodes_{j_\ell}(v)$ it holds that $\{OuterNodes_{j_\ell}(v) \setminus \{x, y\}\} \subset \{OuterNodes_{j_\ell}(u) \setminus \{x, y\}\}$. Thus, there is an outer subset $S'_u \in OuterSubsets(u)$ that contain the same nodes as the nodes in S'_v plus one node from $OuterNodes_{j_\ell}(u)$, especially z.

□

Chapter 3

Proof of Theorem 2.1.2

We now prove Theorem 2.1.2 given in the Introduction. Namely, that the variant of Baker's algorithm specified in Figure 1.2 successfully computes the clique cover size of any k-outerplanar graph in running time $n^{O(k)}$. Building the trees and computing boundaries of slices requires linear time (as shown in Baker [2]). In addition, [2] proves that the algorithm is called recursively at most once for each tree vertex. Since each leaf in a tree represents an oriented exterior edge, the number of vertices in trees is at most the number of edges in the k-outerplanar graph. For a planar graph, the number of edges is linear in the number of nodes. Therefore, the number of calls on the main procedure is linear in the number of nodes. Each recursive call computes the table for a certain vertex in the tree. Each table includes at most $n^{4k}2^{2k}$ entries (corresponding to the subsets we defined). To compute the clique cover for each entry, one must access (via Claim 2.8.3) several entries from other tables. The number of such entries needed is bounded by n^{4k} (specified by the size of our $OuterSubsets$ set system). All in all, this gives a total running time of $n^{O(k)}$.

Chapter 4

Conclusions

The results of this work focus on information graphs G which are undirected and on the case in which the encoding functions are (scalar) linear. The problem of efficiently computing the index coding round complexity for encoding functions which are not scalar linear but rather *vector linear* or non-linear is left open in this work. The connection between the clique cover of G and the scalar linear index coding round complexity holds also for directed side information graphs as well. However, in the directed case it is not hard to find examples in which these two differ (e.g., any directed cycle). Finally, the side information graph model does not suffice to represent the index coding problem in which clients c_i require not a single message but multiple ones. In this case one should introduce a hypergraph side information model (e.g., [1]), which does not fit into the framework discussed in this work.

Bibliography

[1] N. Alon, E. Lubetzky, U. Stav, A. Weinstein, and A. Hassidim. Broadcasting with side information. *In Proceedings of 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 823–832, 2008.

[2] B. S. Baker. Approximation Algorithms for NP-complete Problems on Planar Graphs. *J. ACM*, 41(1):153–180, 1994.

[3] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol. Index Coding with Side Information. *In Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 197–206, 2006.

[4] Yitzhak Birk and Tomer Kol. Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients. *IEEE Trans. Inform. Theory*, 52(6):2825–2830, 2006. An earlier version appeared in INFOCOM 1998.

[5] E. D. Demaine and M. Hajiaghayi. Encyclopedia of Algorithms, Kao, Ming-Yang (Ed.), Chapter : Approximation Schemes for Planar Graph Problems. *Springer*, pages 59–61, 2008.

[6] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. *In Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 637–646, 2005.

[7] Erik D. Demaine, MohammadTaghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences*, 69(2):166–195, 2004.

[8] M. E. Dyer and A. M. Frieze. Planar 3DM is NP-Complete. *Journal of Algorithms*, 7:174–184, 1986.

[9] S. El Rouayheb, A. Sprintson, and C. Georghiades. On the relation between the index coding and the network coding problems. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1823–1827, July 2008.

[10] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3,4):275–291, 2000.

[11] Earkus Frick and Martin Grohe. Deciding first-order properties of locally treedecomposable structures. *Journal of the ACM*, 48(6):1184–1206, 2001.

[12] W. H. Haemers. On Some Problems of Lovász Concerning the Shannon Capacity of a Graph. *IEEE Transactions on Information Theory,*, 25(2):231–232, 1979.

[13] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27:1142–1167, August 1998.

[14] M. Langberg and A. Sprintson. On the hardness of approximating the network coding capacity. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 315–319, July 2008.

[15] A. Rasala Lehman. *Network Coding*. Ph.d. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2005.

[16] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177189, 1979.

[17] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615627, 1980.

[18] E. Lubetzky and U. Stav. Non-linear Index Coding Outperforming the Linear Optimum. *In Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 161–168, 2007.

[19] R. Peeters. Orthogonal Representations Over Finite Fields and the Chromatic Number of Graphs. *Combinatorica*, 16(3):417–431, 1996.

[20] D. Zuckerman. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. *In Proceedings of the 38th annual ACM symposium on Theory of Computing*, pages 681–690, 2006.