Part II

Above Guarantee Parameterizations of Vertex Cover
4.1 The Maximum Matching Problem

The classical notions of *matchings* and *vertex covers* have been at the center of serious study for several decades in the area of Combinatorial Optimization [70]. In 1931, König and Egerváry independently proved the following result of fundamental importance

**Theorem 4.1.1.** ([70]) *In a bipartite graph, the size of the maximum matching is equal to the size of the minimum vertex cover.*

This led to a polynomial-time algorithm for finding a minimum vertex cover in bipartite graphs. Interestingly, this min-max relationship holds for a larger class of graphs known as König-Egerváry graphs (or König graphs) and it includes bipartite graphs as a proper subclass (see Fig. 4.1). In this section, we recall some classical theorems on the subject of matchings and the definitions of the combinatorial objects used in the majority of these results.

**Definition 4.1.2.** Given a graph $G = (V, E)$ and a matching $M$, we call a path $P =$
Figure 4.1: A non-bipartite König graph where the size of the minimum vertex cover (green vertices) is equal to the size of the maximum matching (red edges).

In the graph, an $M$-alternating path (or alternating path if $M$ is clear from the context) if for every $2 \leq i \leq t - 1$ in $P$, either $(v_{i-1}, v_i) \in M$ or $(v_i, v_{i+1}) \in M$ but not both.

The following theorem of Berge is a fundamental result relating maximum matchings and alternating paths.

**Theorem 4.1.3.** ([4]) A matching $M$ in a graph $G$ is not of maximum cardinality if and only if there is an alternating path between two vertices left unsaturated by $M$.

Theorem 4.1.3 led to a polynomial time algorithm to compute a maximum matching in a bipartite graph.

**Theorem 4.1.4.** Maximum matching in a bipartite graph can be computed in polynomial time.

**Definition 4.1.5.** Given a graph $G = (V, E)$ and a matching $M$, we call an alternating path $P = v_1, \ldots, v_t$ in the graph, an odd $M$-path from $v_1$ to $v_t$ (or from $v_1$ to $v_1$) if the edges $(v_1, v_2)$ and $(v_{t-1}, v_t)$ are in $M$ (see Fig 4.2) and an even $M$-path from $v_1$ to $v_t$ if
Figure 4.2: Illustrations of the two types of $M$-alternating paths and an $M$-flower. The non matched edges are represented by the dashed lines. (a) An odd $M$-path $v_1, v_2, v_3, v_4$. (b) An even $M$-path $v_1, v_2, v_3, v_4, v_5, v_6, v_7$. (c) An $M$-flower with root $v_1$, base $v_4$, stem $v_1, v_2, v_3, v_4$, blossom $v_4, v_5, v_6, v_7, v_8, v_4$, and a blossom path $v_5, v_6, v_7, v_8$.

the edge $(v_1, v_2)$ is in $M$, but the edge $(v_{t-1}, v_t)$ is not. If $v_1 \in X$ and $v_t \in Y$, then $P$ is also called an odd (respectively even) $M$-path from $X$ to $Y$.

Note that, by our definition, a single matched edge is an odd $M$-path. In addition, we consider a path consisting of a single vertex to be an even $M$-path.

**Definition 4.1.6.** An odd cycle $C = v_1, v_2, \ldots, v_t, v_1$ is called an $M$-blossom (or blossom if $M$ is clear from the context) if the path $v_2, \ldots, v_t$ is an odd $M$-path. The vertex $v_1$ is called the base of the blossom. The odd $M$-path $v_2, \ldots, v_t$ is called a blossom $M$-path (or blossom path if $M$ is clear from the context).

**Definition 4.1.7.** The union of a blossom $C$ and an odd $M$-path $P$ is called an $M$-flower (or flower if $M$ is clear from the context) if the base of $C$ is an endpoint of $P$ and $C$ and $P$ do not intersect in any other vertex. The base of $C$ is also called the base of the flower, $P$ is called the stem and the endpoint of $P$ disjoint from $C$ is called the root of the flower. If the root of the blossom is in a set $X$, then the flower is referred to as an $X$-flower.

**Definition 4.1.8.** Given a graph $G = (V, E)$ and a matching $M$, an $M$-alternating tree (or alternating tree if $M$ is clear from the context) rooted at a saturated vertex $v$
is defined as the output of the following procedure. Label \( v \) even, and keep all the other vertices unlabeled at this point.

1. For each even vertex \( u \), label its matching partner \( u' \) odd and make \( u' \) a child of \( u \).
2. For each odd vertex \( u \), make its unlabeled neighbors the children of \( u \) and label them even.
3. If neither of the above operations can label an unlabeled vertex, then return the tree constructed.

The following observations are a consequence of the definition of alternating trees.

**Observation 4.1.9.** Given a graph \( G = (V, E) \) and a matching \( M \) in \( G \), consider the \( M \)-alternating tree rooted at a saturated vertex \( v \in V \).

(a) There is an odd (even) \( M \)-path from \( v \) to a vertex \( u \) if and only if \( u \) is labeled odd (respectively even) in this tree.

(b) \( G \) has an \( M \)-flower rooted in \( v \) if and only if there is an edge between two vertices labeled odd in this tree.

**Observation 4.1.10.** Given a graph \( G = (V, E) \), a matching \( M \), and a vertex \( u \), we can test in time \( O(m) \) if there is an \( M \)-flower in \( G \) with \( u \) as the root. Furthermore, given a vertex \( v \), we can test in time \( O(m) \) if there is an odd (or even) \( M \)-path from \( u \) to \( v \) in \( G \).

**Proof.** This follows from the fact that the alternating tree rooted at \( u \) can be computed in time \( O(m) \) assuming that the matching has been given in the form of an appropriate data structure. Since this is straightforward, we do not give a detailed proof. \( \square \)

### 4.2 König Graphs with Extendable Vertex Covers

In this section we obtain an extension of the result by König and Egerváry relating maximum matchings and minimum vertex covers. More precisely, we address the following
question.

*When does a König graph have a minimum vertex cover containing a specified subset of vertices of the graph?*

While the question can be considered an extension of that studied by König and Egerváry, the answer we obtain revolves around the structures of alternating paths and flowers (see previous section) used by Berge [4] and Edmonds [29] in their classical results.

We begin by giving a sketch of our characterization. Consider the path $P_\ell = v_1, \ldots, v_\ell$, where $\ell$ is even. It is clearly a König graph since the size of the maximum matching is $\frac{\ell}{2}$ and we have a vertex cover of size $\frac{\ell}{2}$ comprising the vertices $v_2, v_4, \ldots, v_\ell$. However, observe that there is no minimum vertex cover which contains $v_1$ and $v_\ell$ since any minimum vertex cover containing $v_1$ is forced to exclude the set $\{v_2, v_4, \ldots, v_\ell\}$. In other words, if there is an odd $M$-path between two vertices of the König graph, then we cannot have a minimum vertex cover containing both end points of the path. Similarly, consider a graph which is an $M$-flower. By an argument similar to the previous one (for an odd length path), observe that there cannot be a minimum vertex cover for this graph containing the root.

The above arguments give us an idea as to what type of structures must necessarily be excluded for the given annotated König graph to have a minimum vertex cover containing the annotated vertices. Our main contribution in this regard is a proof that it is also sufficient for the above structures to be absent from the given annotated graph for it to have a minimum vertex cover containing the annotated vertices. Before we present a formal proof of our characterization, we revisit a different characterization of König graphs, followed by some structural observations regarding alternating paths and flowers in König graphs.
Lemma 4.2.1 (see, for example, [81]). A graph $G = (V, E)$ is König if and only if there exists a bipartition of $V$ as $V_1 \uplus V_2$, with $V_1$ a vertex cover of $G$ such that there exists a matching across the cut $(V_1, V_2)$ saturating every vertex of $V_1$.

Due to Lemma 4.2.1, we define a König graph $G$ as a triple $(A, B, E)$ where the vertex set of the graph is $A \cup B$ with $A$ being a minimum vertex cover of $G$, $B$ the corresponding independent set and $E$ is the edge set of the graph. We also have the following simple observation about König graphs.

**Observation 4.2.2.** Given a König graph $G = (A, B, E)$, let $M$ be a maximum matching of $G$ and let $U$ be the set of vertices left unsaturated by $M$. Then, $M$ saturates $A$ across the cut $(A, B)$ and no vertex of $U$ is present in any minimum vertex cover of $G$.

**Proof.** Since every minimum vertex cover of a König graph contains exactly one vertex of each edge in any maximum matching, it must be the case that $M$ saturates $A$ across the cut $(A, B)$. Furthermore, if a vertex in $U$ occurs in some minimum vertex cover of $G$, say $S$, then since any vertex cover is forced to contain at least one vertex from each edge of $M$, $S$ has size at least $|M| + 1$, which is a contradiction. This completes the proof.

The following consequences follow from the definitions in the previous section.

**Lemma 4.2.3.** Let $G = (A, B, E)$ be a König graph with a maximum matching $M$ and let $L \subseteq A$ and $R \subseteq B$ be vertex subsets.

(a) There is no odd $M$-path from $A$ to $A$.

(b) Any odd $M$-path from $B$ to $B$ contains exactly one edge between two vertices in $A$.

(c) There is no blossom with the base in $B$. 

46
(d) In any blossom in $G$, one of the two neighbors of the base in the blossom is in $B$.

(e) A minimum vertex cover of $G$ contains at most 1 endpoint of an odd $M$-path. Furthermore, if a minimum vertex cover of $G$ does not contain a vertex $u$, then it does not contain any vertex which has an even $M$-path to $u$.

(f) Let $P = v_1, \ldots, v_t$ be an odd $M$-path from $B$ to $B$. Then there is an edge $(u, v)$ such that $u, v \in A$ and there is an odd $M$-path $P_1$ from $v_1$ to $u$ and an odd $M$-path $P_2$ from $v_t$ to $v$ such that every matched edge in $P$ is present in either $P_1$ or $P_2$.

(g) Consider a vertex $v$ and an even $M$-path $P$ from some vertex $u$ to $v$. Then $P$ does not contain the matching partner of $v$.

Proof. (a) Consider the bipartite graph obtained from $G$ by deleting edges with both endpoints in $A$. If there were an odd $M$-path from $A$ to $A$ in $G$, then this path also exists in the bipartite graph which we constructed, implying an odd path between the same partition of the bipartite graph, a contradiction.

(b) We know that any odd $M$-path from $B$ to $B$ which does not contain an edge between two vertices in $A$ is also an odd path from $B$ to $B$ in the bipartite graph constructed as described in (a), a contradiction. Hence any such path must contain at least one edge between two vertices of $A$. We now show that there is exactly one such edge. Let $P = v_1, \ldots, v_t$ be the odd $M$-path under consideration and suppose that there are two edges $e_1 = (v_i, v_{i+1})$ and $e_2 = (v_j, v_{j+1})$ in $P$ such that $v_i, v_{i+1}, v_j, v_{j+1} \in A$. Assume without loss of generality that $i + 1 < j$. Then, the edges $(v_{i+1}, v_{i+2}), (v_{j-1}, v_j)$ must be in $M$. They cannot be the same edge since that would mean a matched edge between vertices of $A$. However, the subpath $v_{i+1}, v_{i+2}, \ldots, v_j$ is an odd $M$-path from $A$ to $A$, which contradicts (a).
(c) If this were not the case, then there is a blossom with some vertex \( b \in B \) as its base. Let \( u_1 \) and \( u_2 \) be the neighbors of \( b \) in the blossom. Since \( B \) is independent, \( u_1, u_2 \in A \). However, by definition, the blossom path between \( u_1 \) and \( u_2 \) is an odd \( M \)-path, a contradiction to (a).

(d) Let \( u_1 \) and \( u_2 \) be the neighbors of the base of the blossom within the blossom. Suppose \( u_1, u_2 \in A \). Then the blossom path from \( u_1 \) to \( u_2 \) is an odd \( M \)-path from \( A \) to \( A \) which contradicts (a).

(e) Let \( P = v_1, \ldots, v_t \) be an odd \( M \)-path and let \( S \) be a minimum vertex cover for \( G \). Note that since \( S \) is a minimum vertex cover for \( G \), \( S \) must contain exactly one end point of each matched edge in \( M \). We prove by induction on \( t \) that if \( v_1 \in S \), then \( v_t \notin S \). In the base case, when \( t = 2 \), the claim is clearly true. Therefore, let \( t > 2 \) and assume that our claim is true \( \forall t' < t \). Consider the path \( v_1, \ldots, v_{t-2} \). It is clearly an odd \( M \)-path. By the induction hypothesis, \( v_1 \in S \) implies that \( v_{t-2} \notin S \). Since \( S \) is a vertex cover, it must be the case that \( v_{t-1} \in S \) in order to cover the edge \( (v_{t-2}, v_{t-1}) \). However, for \( S \) to be a minimum vertex cover, it must contain exactly one vertex from the matched edge \( (v_{t-1}, v_t) \), implying that \( v_t \notin S \). For the proof of the second statement, suppose that \( G \) contains a minimum vertex cover \( S \) excluding \( u \) and let \( v \) a vertex such that there is an even \( M \)-path \( P \) from \( v \) to \( u \). The statement is clearly true if \( u = v \). Therefore, let \( P = v_1, \ldots, v_t \) be this path where \( v = v_1 \) and \( u = v_t \) and \( t > 1 \). Since \( v_t \) is not in \( S \), it must be the case that \( v_{t-1} \in S \). Since \( P \) is an even \( M \)-path, the subpath of \( P \) from \( v_1 \) to \( v_{t-1} \) is an odd \( M \)-path and by the first statement, \( v_{t-1} \in S \) implies that \( v_1 \notin S \). Therefore, we have that \( v \) is also disjoint from \( S \).

(f) Let \( P = v_1, \ldots, v_t \) be an odd \( M \)-path from \( B \) to \( B \). Let \( v_i \) and \( v_{i+1} \) be the vertices
occurring in $P$ such that $v_i, v_{i+1} \in A$ given by (b). Let $P_1$ be the subpath of $P$ from $v_1$ to $v_i$ and let $P_2$ be the subpath of $P$ from $v_{i+1}$ to $v_t$. We claim that $P_1$ and $P_2$ are both odd $M$-paths. Note that $v_i, v_{i+1} \in A$ implies that $(v_i, v_{i+1}) \notin M$. Since $P$ is an $M$-alternating path, the edges $(v_{i-1}, v_i)$ and $(v_{i+1}, v_{i+2})$ are in $M$. Therefore, $P_1$ is an $M$-alternating path where the first and last edges are matched edges, that is, $P_1$ is an odd $M$-path. The same is the case for $P_2$ and since $P_1$ and $P_2$ together contain every matched edge in $P$, the claim follows.

(g) Let $v'$ be the matching partner of $v$ and suppose $P$ contains $v'$. Observe that $u \neq v'$ since that would imply that the first edge of $P$ is a matched edge, implying that the first edge of $P$ is $(v', v)$, which is a contradiction to our assumption that $P$ is an even $M$-path from $u$ to $v$. Since $v'$ is not an end point of $P$, $v'$ has two edges of $P$ incident on it. Since $P$ is an alternating path, at least one of these two edges is that edge $(v', v)$. Since $v$ is an endpoint of $P$, it must be the case that the last edge of $P$ is the edge $(v', v)$. Since $(v', v)$ is a matched edge, it contradicts our assumption that $P$ is an even $M$-path from $u$ to $v$.

Using the above observations, we prove our main structural result.

**Lemma 4.2.4.** Let $G = (A, B, E)$ be a König graph with a maximum matching $M$, let $L \subseteq A$ and $R \subseteq B$ be saturated vertex subsets, and let $U \subseteq B$ be the set of vertices of $G$ left unsaturated by $M$. Then, $G$ has a minimum vertex cover containing $L \cup R$ if and only if $G$ contains none of the following structures.

1. an odd $M$-path from $L \cup R$ to $L \cup R$
2. an even M-path from $L \cup R$ to $U$.

3. an R flower

**Proof.** $(\Rightarrow)$ Let $A = a_1, \ldots, a_r$ and $M = \{m_1, \ldots, m_r\}$ where $m_i = (a_i, b_i)$ and let $U = \{b_{r+1}, \ldots, b_{|R|}\}$. Suppose $G$ has a minimum vertex cover $S$ containing $L$ and $R$. Consider an odd $M$-path in $G$. By Lemma 4.2.3(e), it cannot be the case that both endpoints of this path are in $S$. Hence, any odd $M$-path in $G$ has at least one endpoint disjoint from $L \cup R$.

Suppose that $G$ has an even $M$-path $P$ from $L \cup R$ to a vertex $u \in U$. Since no vertex in $U$ is part of any minimum vertex cover (by Observation 4.2.2), $U$ is disjoint from $S$. Therefore, by Lemma 4.2.3(e), any vertex which has an even $M$-path to $U$ must also be disjoint from $S$. Since $L \cup R \subseteq S$, it cannot be the case that a vertex in $L \cup R$ has an even $M$-path to a vertex in $U$.

Finally, consider the case when $G$ has an R-flower. Then, the root of the flower is in $S$ since $R \subseteq S$. By the definition of flowers, the stem is an odd $M$-path from the root to the base and by Lemma 4.2.3(e) the base is not in $S$. This implies that $S$ contains both neighbors of the base in the blossom, a contradiction to Lemma 4.2.3(e) since the blossom path between these two vertices is an odd $M$-path.

$(\Leftarrow)$ Suppose $G$ has no odd $M$-paths from $L \cup R$ to $L \cup R$, no even $M$-paths from $L \cup R$ to $U$ and no R-flowers. We define $S_1$ as the set of all vertices to which there is an even $M$-path from $R$. We define the set $S_2$ as the set of all vertices $a_i$ such that $m_i$ is not covered by $S_1$. We let $S$ denote the union of $S_1$ and $S_2$. Formally,

$$S_1 = \{v \mid \text{there is an even } M\text{-path from } R \text{ to } v\}.$$
\[ S_2 = \{ a_i | V(m_i) \cap S_1 = \phi \}. \]

\[ S = S_1 \cup S_2. \]

We claim that \( S \) is a minimum vertex cover of \( G \), containing \( L \cup R \). We first prove that \( S \) contains \( L \cup R \). Recall that by definition, a single vertex is an even \( M \)-path and hence \( S_1 \) contains \( R \). Suppose there is a vertex \( a_i \) in \( L \) which is not in \( S \). By the definition of \( S \), \( b_i \in S_1 \), implying that there is an even \( M \)-path \( P \) from \( R \) to \( b_i \). By Lemma 4.2.3(g) \( P \) does not contain \( a_i \). Hence, \( P + (b_i, a_i) \) is an odd \( M \)-path from \( R \) to \( L \), a contradiction to our assumption that there are no odd \( M \)-paths from \( L \cup R \) to \( L \cup R \). Hence, we conclude that \( L \cup R \subseteq S \).

We now prove that \( S \) is indeed a vertex cover. Suppose that \( S \) is not a vertex cover and let \( e = (a_i, b_j) \) be an edge left uncovered by \( S \). Note that the definition of \( S \) implies that \( e \) cannot be a matched edge since if \( S_1 \) picked neither endpoint of a matched edge, then we will have added the vertex of that edge lying in \( A \), into \( S_2 \) and hence in \( S \) as well. Therefore, \( i \neq j \). Since \( a_i \notin S \), it must be the case that \( b_i \in S_1 \). Hence there is an even \( M \)-path \( P = v_1, \ldots, v_t \) from \( R \) to \( b_i \) where \( v_t = b_i \). By Lemma 4.2.3(g) \( P \) does not contain \( a_i \).

If \( P \) did not contain \( b_j \), then \( P + (b_i, a_i) + (a_i, b_j) \) is an even \( M \)-path from \( R \) to \( b_j \). But this implies that \( b_j \in S_1 \subseteq S \), a contradiction to our assumption that \( e \) was not covered by \( S \). Therefore, we assume that \( P \) contains \( b_j \). Furthermore, since \( b_j \notin S \), we have that \( b_j \notin R \), which implies that \( b_j \) is not an endpoint of \( P \) and hence there are two edges incident on \( b_j \) in \( P \). Since one of the edges incident on \( b_j \) in \( P \) is a matched edge, it must be the case that \( b_j \notin U \) and \( b_j \) has a matching partner \( a_j \). Furthermore, \( a_j \) is adjacent to \( b_j \) in \( P \). If \( a_j \) occurs immediately after \( b_j \), then the subpath of \( P \) from
$v_1$ to $b_j$ is an even $M$-path, in which case $b_j$ would have been added to $S_1$, covering the edge $e$. Hence, $a_j$ must occur immediately before $b_j$ in $P$. Let $P'$ be the subpath of $P$ from $v_1$ to $b_j$. Now, $P' + (b_j, a_i)$ is an even $M$-path from $R$ to $a_i$, which implies that $a_i \in S_1 \subseteq S$, a contradiction. We have thus established that $S$ is indeed a vertex cover of $G$.

We now prove that $S$ is a minimum vertex cover of $G$. In particular we will prove that $S$ contains exactly one vertex from every edge of $M$. Since we have already shown that $S$ is a vertex cover, it contains at least one vertex from every matched edge. Furthermore, due to the absence of even $M$-paths from $L \cup R$ to $U$, no vertex of $U$ is in $S$. Therefore it is enough for us to prove that $S$ does not contain both end points of any matched edge. Suppose there is a matched edge $m_j$ such that both $a_j$ and $b_j$ are in $S$. Then it must be the case that both $a_j$ and $b_j$ are in $S_1$ as well. This could only be possible if there were even $M$-paths $P_1 = x_1, \ldots, x_s$ and $P_2 = y_1, \ldots, y_t$ from $R$ to $a_j$ and $b_j$ respectively. For each matched edge $m_i$ such that $S$ contains both end points, let $P_{i1}$ and $P_{i2}$ be even $M$-paths of least length from $R$ to $a_i$ and $b_i$ respectively. Among all such matched edges, let $m_d$ be one which minimizes the sum $|P_{i1}| + |P_{i2}|$, that is $|P_{1d}| + |P_{2d}| = \min_i \{ |P_{i1}| + |P_{i2}| \}$.

For the sake of convenience, in the rest of the proof of this lemma, we refer to the paths $P_{1d}$ and $P_{2d}$ as $P_1$ and $P_2$. By Lemma 4.2.3(g), $P_1$ does not contain $b_d$ and $P_2$ does not contain $a_d$. We now consider the following two cases.

1. If $P_1$ and $P_2$ do not intersect at all, then $P_1 + (a_j, b_j) + \text{Rev}(P_2)$ is an odd $M$-path from $R$ to $R$, a contradiction.

2. Suppose $P_1$ and $P_2$ do intersect and let $y_q = x_p$ be the last vertex along $P_2$ when traversing from $y_1$, which is also present in $P_1$. Since one of the two edges incident on $y_q$ in $P_2$ is a matched edge, it must be the case that the edge $(y_{q-1}, y_q)$ is a matched edge. For the same reason, it must be the case that one of the two edges $(x_p, x_{p+1})$
or \((x_{p-1}, x_p)\) is the same as the edge \((y_{q-1}, y_q)\), that is, either \(y_{q-1} = x_{p+1}\) or \(y_{q-1} = x_{p-1}\). Since \(P_1\) and \(P_2\) are both disjoint from either \(a_d\) or \(b_d\), the edge \((y_{q-1}, y_q)\) cannot be the edge \(m_d\) since it occurs in both paths. We now consider the following two cases.

(a) \(x_{p-1} = y_{q-1}\). Let \(P'_1\) be the subpath of \(P_1\) from \(x_p\) to \(x_s\) and let \(P'_2\) be the subpath of \(P_2\) from \(y_q\) to \(y_t\). Then, the paths \(P'_1\) and \(P'_2\) along with the edge \(m_d\) form a blossom \(C\). Let \(P''_1\) be the subpath of \(P_1\) from \(x_1\) to \(x_p\). By our assumption regarding \(y_q\), \(P''_1\) is disjoint from \(P'_2\). Since \(P''_1\) is an odd \(M\)-path from \(x_1\) to \(x_p\) disjoint from the blossom \(C\), we have a flower with \(x_1\) as the root. However, \(x_1 \in R\), which implies that there is an \(R\) flower in \(G\), a contradiction to our assumption.

(b) \(x_{p+1} = y_{q-1}\). Let \(P'_1\) be the subpath of \(P_1\) from \(x_1\) to \(x_p\) and let \(P'_2\) be the subpath of \(P_2\) from \(y_1\) to \(y_{q-1}\). Then, the paths \(P'_1\) and \(P'_2\) are even \(M\)-paths from \(R\) to the two endpoints of the matched edge \((y_q, y_{q-1})\). Since we know that the edge \((y_{q-1}, y_q)\) is distinct from the edge \(m_d\), it must be the case that \(|P'_1| + |P'_2| < |P_1| + |P_2|\), a contradiction to our choice of \(m_d\).

This concludes the proof that \(S\) is a minimum vertex cover of \(G\) containing \(L \cup R\) and hence the proof of the lemma.

Since a perfect matching does not leave any vertex unsaturated, we have the following special case of the above lemma which will be a crucial component of the next chapter.

**Lemma 4.2.5.** Let \(G = (A, B, E)\) be a König graph with a perfect matching \(M\), let \(L \subseteq A\) and \(R \subseteq B\), be vertex subsets. Then, \(G\) has a minimum vertex cover containing \(L \cup R\) if and only if \(G\) contains neither an odd \(M\)-path from \(L \cup R\) to \(L \cup R\) nor an \(R\)-flower.
4.3 Computing extendable vertex covers

In this section, we show that we can compute a minimum vertex cover containing the annotated vertices of a König graph (if one exists) in polynomial time. We begin with the following definition.

**Definition 4.3.1.** Consider a König graph $G = (A, B, E)$ a maximum matching $M$ of $G$ and let $U$ be the set of vertices left unsaturated by $M$. Let $R(U)$ be the set of vertices in $V \setminus U$ which to which there is an alternating path from a vertex in $U$. We denote by $G/U$ the graph induced on the set $V \setminus R(U)$.

**Observation 4.3.2.** Consider a König graph $G = (A, B, E)$ a maximum matching $M$ and let $U$ be the set of vertices left unsaturated by $M$. Then, the graph $G/U$ is a König graph with a perfect matching.

**Proof.** Consider the set of alternating trees rooted at the vertices of $L$ where $L$ is the set of neighbors of $U$. Observe that the union of the vertices in these trees is precisely the set $R(U) \setminus U$. Since by definition, any vertex occurs along with its matching partner in an alternating tree, every vertex in the graph $G/U$ also occurs with its matching partner. Therefore, $G/U$ has a perfect matching. Furthermore, since $A$ is a vertex cover for $G$, the set $A \setminus R(U)$ is a vertex cover for $G/U$. Since $G/U$ has a matching saturating $A \setminus R(U)$ across the cut $(A \setminus R(U), B \setminus R(U))$ in $G/U$, it must be a König graph. □

**Lemma 4.3.3.** Consider a König graph $G = (A, B, E)$. Let $M$ be a maximum matching of $G$ and $U$ be the set of vertices left unsaturated by $M$. Let $X$ be the set of vertices of $G$ to which there is an odd length alternating path from $U$. Then, for any minimum vertex cover $S$ of $G$, $X = S \cap R(U)$ and $S \setminus X$ is a minimum vertex cover for $G/U$.  

54
Proof. Observe that $X \subseteq A$ since $U \subseteq B$. Let $Y$ be the set of vertices in $B$ whose matching partners are in $X$. By Observation 4.1.9, for any vertex $y \in Y$, there is a vertex of $U$ labeled even in the alternating tree rooted at $y$. Therefore, since $U$ is not part of any minimum vertex cover of $G$, neither is $Y$ (Lemma 4.2.3(e)), implying that $X$ is part of every minimum vertex cover of $G$.

Lemma 4.3.4. Let $G = (A, B, E)$ be a König graph with a perfect matching $M$, let $L \subseteq A, R \subseteq B$ be saturated vertex subsets. If $G$ has a minimum vertex cover containing $L \cup R$, then one such minimum vertex cover can be computed in time $O(mn)$.

Proof. Observe that the sets $S_1$ and $S_2$ defined in the proof of Lemma 4.2.4 can be computed in time $O(mn)$. Since we have already shown that $S = S_1 \cup S_2$ is a minimum vertex cover of $G$, this completes the proof of the lemma.

Lemma 4.3.5. Let $G = (A, B, E)$ be a König graph with a maximum matching $M$, let $L \subseteq A, R \subseteq B$ be vertex subsets and let $U \subseteq B$ be the set of vertices left unsaturated by $M$. If $G$ has a minimum vertex cover containing $L \cup R$, then one such minimum vertex cover can be computed in time $O(mn)$.

Proof. We begin by first computing the intersection of the minimum vertex cover with the set $R(U)$ which is part of every minimum vertex cover (Lemma 4.3.3). This can be done in time $O(mn)$ by constructing alternating trees for every vertex in $B$. Now, by Lemma 4.3.3, it only remains for us to compute a minimum vertex cover containing $L' = L \setminus R(U)$ and $R' = R \setminus R(U)$ in the graph $G' = G/U$, which is a König graph with a perfect matching. This can be done in time $O(mn)$ by Lemma 4.3.4, which completes the proof of the lemma.
4.4 Characterizing König graphs with a unique minimum vertex cover

In this section, we give a complete characterization of König graphs based on the number of minimum vertex covers which they contain.

The following observation follows from Lemma 4.3.3 and implies that it is sufficient for us to restrict our attention to König graphs with perfect matchings.

Observation 4.4.1. Consider a König graph $G = (A, B, E)$ with a maximum matching $M$ where $U$ is the set of vertices left unsaturated by $M$. Then, $G$ has a unique minimum vertex cover if and only if the graph $G/U$, which is a König graph with a perfect matching, has a unique minimum vertex cover.

Lemma 4.4.2. Consider a König graph $G = (V, E)$ with a perfect matching $M$. Then, $G$ has a unique minimum vertex cover if and only if every vertex of $G$ is in the stem of a flower.

Proof. ($\Rightarrow$) Suppose that $G$ has a unique minimum vertex cover $S$ and there is a vertex $v \in V$ which is not in the stem of a flower. We first consider the case when $v \notin S$. Consider the alternating tree rooted at $v$, call it $T$. We now construct a set $S'$ as follows. Initially, set $S' = S \cap (V \setminus V(T))$. Now, for every vertex labeled even in $T$, we add this vertex to $S'$. Clearly, $|S'| = |S|$ since every matched edge in $G$ contributes exactly 1 vertex to $S'$. By our assumption, there is a unique minimum vertex cover and hence it must be the case that $S'$ is not a vertex cover. This implies that there is an edge $(u, w)$ not covered by $S'$. If $u$ and $w$ were both in $T$, then they must both be labeled odd, which implies the presence of a flower with the stem containing $v$ (by Observation 4.1.9). Similarly, if neither $u$ nor $w$ were in $T$, then the set $S \cap (V \setminus V(T))$ will contain either
$u$ or $w$ since $S$ is a vertex cover of $G$. Hence, we are left with the case when exactly
one of $u$ or $w$ is in $T$. Assume without loss of generality that $u$ is in $T$ and $w$ is not.
Since $w \notin S'$, it must be the case that $w \notin S$, implying that $u \in S$. However, $u \notin S'$ implies that $u$ is labeled odd in $T$. This also implies that the edge $(u, w)$ is not a matched edge and hence the path in $T$ from $v$ to $u$, along with the edge $(u, w)$ implies that there is an even $M$-path from $v$ to $w$, a contradiction to Observation 4.1.9. The case when $v \in S$ can be argued similarly by considering the alternating tree rooted at the matching partner of $v$ which is not in $S$.

$(\Leftarrow)$ Suppose that every vertex of $G$ is part of the stem of a flower. Consider an arbitrary vertex $v$ and a flower whose stem contains $v$. We claim that if the parity of the shortest path along the stem from the base to $v$ is even, then $v$ is in every minimum vertex cover of $G$ and if it is odd, then $v$ is in no minimum vertex cover of $G$. Suppose that $v$ is the base of the flower and not part of some minimum vertex cover, then the two neighbors of $v$ in the blossom are both in the minimum vertex cover, contradicting Lemma 4.2.3(e) since there is an odd $M$-path between them. Suppose that $v$ is not the base of this flower. By Lemma 4.2.3(a), if a vertex $u$ has an odd $M$-path to $v$, then since $v$ is part of every minimum vertex cover, $u$ is not in any minimum vertex cover. Since no vertex at an odd distance along the stem from the base is in any minimum vertex cover, every vertex at an even distance along the stem from the base is part of every minimum vertex cover. This proves our claim and in fact also shows that if every vertex is part of the stem of a flower, then no vertex will be at an even distance (along the stem) from the base of one flower and at an odd distance (along the stem) from the base of another flower. Hence, the presence or absence of a vertex in any minimum vertex cover of $G$ is solely determined by its position in the stem of any flower. Since every vertex in $G$ is in the stem of a flower, the lemma follows.

\[ \square \]
Observation 4.4.1 along with Lemma 4.4.2 gives a complete characterization of König graphs with unique a minimum vertex cover.

4.5 Notes

If we restrict ourselves to bipartite graphs with a perfect matching, it is clear that either partition forms a vertex cover. We note that Lemma 4.4.2 is a generalization of this trivial observation. In other words, since the presence of a flower requires the presence of an odd cycle, by Lemma 4.4.2, a bipartite graph with a perfect matching has at least 2 minimum vertex covers.

We conclude with the following corollary of Observation 4.4.1 applied to bipartite graphs, which is very well known.

Corollary 4.5.1. A bipartite graph $G = (V, E)$ with a maximum matching $M$ has a unique vertex cover if and only if $U = V$ where $U$ is the set of vertices of $G$ reachable from $U$ via alternating paths.
5.1 Introduction

The standard version of VERTEX COVER, where we are interested in finding a vertex cover of size at most \( k \) for the given parameter \( k \) was one of the earliest problems that was shown to be FPT [27]. After a long race, the current best algorithm for VERTEX COVER runs in time \( \mathcal{O}(1.2738^k + kn) \) [15]. However, when \( k < m \), the size of the maximum matching, the standard version of VERTEX COVER is not interesting, as the answer is trivially NO. And if \( m \) is comparable to the size of the vertex set itself, for example, when the graph has a perfect matching, then for the cases the problem is interesting, the running time of the standard version is not practical, as \( k \), is quite large in these cases. This motivates the parameterization of VERTEX COVER above the size of the maximum matching.
**Above Guarantee Vertex Cover (AGVC)**

**Parameter:** $k$

**Input:** Graph $G$, a maximum matching $M$ for $G$, positive integer $k$.

**Question:** Does $G$ have a vertex cover of size at most $|M| + k$?

Prior to the work presented in this chapter, the only known parameterized algorithm for *Above Guarantee Vertex Cover* was the consequence of a parameter preserving reduction from AGVC to the *Almost 2-SAT* problem. In *Almost 2-SAT*, we are given a 2-SAT formula $\phi$, a positive integer $k$ and the objective is to check whether there exists a set of at most $k$ clauses whose deletion from $\phi$ makes the resulting formula satisfiable. The *Almost 2-SAT* problem was introduced in [72] and a decade later it was shown by Razgon and Barry O’Sullivan [90] to have an $O^*(15^k)$ time algorithm, thereby proving fixed-parameter tractability of the problem when $k$ is the parameter. The *Almost 2-SAT* problem is a fundamental problem in the context of designing parameterized algorithms. This is evident from the fact that there is a polynomial time parameter preserving reduction from problems like *Odd Cycle Transversal* [62] and *Above Guarantee Vertex Cover* [81] to it. An FPT algorithm for *Almost 2-SAT* led to FPT algorithms for several problems, including AGVC and *König Vertex Deletion* [81]. In recent times this has been used as a subroutine in obtaining a parameterized approximation as well as an FPT algorithm for *Multicut* [78, 79]. Later chapters in the thesis will also demonstrate the utility of having an FPT algorithm for *Almost 2-SAT*. Therefore, a strong motivation for obtaining a faster FPT algorithm for *Above Guarantee Vertex Cover* is that it also implies a faster FPT algorithm for *Almost 2-SAT* and a number of other problems which are summarized in Fig. 5.1.

In this chapter, we present an improved FPT algorithm for AGVC by using our structural results described in the previous chapter and the important separator template.
Figure 5.1: Some of the many problems which have a parameter preserving reduction to AGVC. Out of these reductions, the reduction from KVD \(_{pm}\) takes the parameter from \(k\) to \(\frac{k}{2}\) while the rest are all parameter preserving reductions.

5.2 Algorithm for AGVC

We first apply the following lemma that allows us to assume that the input graph has a perfect matching.

**Lemma 5.2.1.** [81, Lemma 5] If \((G = (V, E), M, k)\) is an instance of Above Guarantee Vertex Cover and \(G\) is a graph without a perfect matching, then in time \(O(m\sqrt{n})\), we can obtain an instance \((G', M', k)\) such that \(G'\) has a perfect matching \(M'\) and \((G, M, k)\) is a Yes instance of Above Guarantee Vertex Cover if and only if \((G', M, k)\) is a Yes instance of Above Guarantee Vertex Cover where
Due to Lemma 5.2.1, we assume that in our input instance \((G, M, k)\), the matching \(M\) is a perfect matching of \(G\). We now describe the iterative compression step, which is central to our algorithm, in detail.

**Iterative Compression for AGVC.** Given an instance \((G = (V, E), M, k)\) of \textsc{Above Guarantee Vertex Cover} let \(M = \{m_1, \ldots, m_{n/2}\}\) be a perfect matching for \(G\) where \(n = |V|\). Define \(M_i = \{m_1, \ldots, m_i\}\), \(m_i = (a_i, b_i)\), and \(G_i = G[V(M_i)]\), \(1 \leq i \leq n/2\). We iterate through the instances \((G_i, M_i, k)\) starting from \(i = k + 1\) and for the \(i^{th}\) instance, with the help of a known solution \(S_i\) of size at most \(|M_i| + k + 1\) we try to find a solution \(\hat{S}_i\) of size at most \(|M_i| + k\). Formally, the compression problem we address is the following.

<table>
<thead>
<tr>
<th><strong>AGVC Compression (AGVCC)</strong></th>
<th><strong>Parameter:</strong> (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ((G, M, S, k)) where ((G = (V, E), M, k)) is an instance of AGVC and (S) a vertex cover of (G) of size at most (</td>
<td>M</td>
</tr>
</tbody>
</table>

We reduce the AGVC problem to \(n/2 - k\) instances of the AGVCC problem as follows. Let \(I_i = (G_i, M_i, S_i, k)\) be the \(i^{th}\) instance. Clearly, the set \(V(M_{k+1})\) is a vertex cover of size at most \(|M_{k+1}| + k + 1\) for the instance \(I_{k+1}\). It is also easy to see that if \(\hat{S}_{i-1}\) is a vertex cover of size at most \(|M_{i-1}| + k\) for instance \(I_{i-1}\), then the set \(\hat{S}_{i-1} \cup V(m_i)\) is a vertex cover of size at most \(|M_i| + k + 1\) for the instance \(I_i\). We use these two observations to start off the iteration with the instance \((G_{k+1}, M_{k+1}, S_{k+1}, k)\) where \(S_{k+1} = V(M_{k+1})\) and look for a vertex cover of size at most \(|M_{k+1}| + k\) for this instance. If we find such a vertex cover \(\hat{S}_{k+1}\), we set \(S_{k+2} = \hat{S}_{k+1} \cup V(m_{k+2})\) and look for a vertex cover of size at most \(|M_{k+2}| + k\) for the instance \(I_{k+2}\) and so on. If, during any iteration, the
corresponding instance does not have a vertex cover of the required size, it implies that
the original instance is also a NO instance. Finally the solution for the original input
instance will be \( \hat{S}_2 \). Since there can be at most \( \frac{n}{2} \) iterations, the total time taken is
bounded by \( \frac{n}{2} \) times the time required to solve the AGVCC problem.

Our algorithm for AGVCC is as follows. Let the input instance be \( I = (G = (V,E), S, M, k) \). Let \( M' \) be the edges in \( M \) which have both vertices in \( S \). Note that
\( |M'| \leq k + 1 \). Then, \( G \setminus V(M') \) is a König graph and by Lemma 4.2.1 has a partition \( (A, B) \) such that \( A \) is a minimum vertex cover and there is a matching saturating
\( A \) across the cut \( (A, B) \), which in this case is \( M \setminus M' \). We guess a subset \( Y \subseteq M' \)
with the intention of picking both vertices of these edges in our solution (see Fig. 5.2).
For the remaining edges of \( M' \), exactly one vertex from each edge will be part of our
eventual solution. For each edge of \( M' \setminus Y \), we guess the vertex which is not part of our
eventual solution. Let \( T \) be the set of vertices guessed in this way to be disjoint from the
solution. Define \( L = A \cap N_G(T) \) and \( R = B \cap N_G(T) \). Clearly our guess forces \( L \cup R \)
to be in the solution (see Fig. 5.3). We have thus reduced this problem to the problem
of checking if the instance \( (G[V(M \setminus M')], A, M \setminus M', k - |M'|) \) has a vertex cover
of size at most \( |M \setminus M'| + k - |M'| \) which contains \( L \) and \( R \). We formally define this
annotated variant as follows.

<table>
<thead>
<tr>
<th>ANNOTATED AGVC (A-AGVC)</th>
<th>Parameter: ( k )</th>
</tr>
</thead>
</table>
| **Input:** \( (G = (A, B, E), M, L, R, k) \), where \( (G = (A, B, E), M, k) \) is an instance of
AGVC, \( L \subseteq A \) and \( R \subseteq B \). |
| **Question:** Does \( G \) have a vertex cover of size at most \( |M| + k \) containing \( L \cup R \)? |

Our main result is the following lemma.

**Lemma 5.2.2.** There is an algorithm for A-AGVC which runs in time \( \mathcal{O}(4^k kmn) \) on
an input \( (G, M, L, R, k) \), where \( n \) is the number of vertices in \( G \) and \( m \) is the number

63
of edges in $G$.

Given Lemma 5.2.2 the running time of our algorithm for AGVCC is bounded as follows. For every $0 \leq i \leq k$, for every $i$ sized subset $Y$, for every guess of $T$, we run the algorithm for A-AGVC with parameter $k - i$. For each $i$, there are $\binom{k+1}{i}$ subsets of $M'$ of size $i$, and for every choice of $Y$ of size $i$, there are $2^{k+1-i}$ choices for $T$ and for every choice of $T$, running the algorithm for A-AGVC given by Lemma 5.2.2 takes time $\mathcal{O}(4^{k-i}kmn)$. Hence, the running time of our algorithm for AGVCC is bounded by $\mathcal{O} \left( \sum_{i=0}^{k} \binom{k+1}{i} 2^{k+1-i} 4^{k-i} kmn \right) = \mathcal{O}(9^k kmn)$ and hence our algorithm for ABOVE GUARANTEE VERTEX COVER runs in time $\mathcal{O}(9^k kmn^2)$. Thus we have the following theorem.

**Theorem 5.2.3.** AGVC can be solved in time $\mathcal{O}(9^k kmn^2)$.

The rest of the chapter is devoted to presenting a proof of Lemma 5.2.2.
5.3 Phase 1

In this section, we describe how to apply Phase 1 of the important separator template. In order to do so, we use Lemma 4.2.5 to model the A-AGVC problem as a problem of eliminating certain paths in a directed graph.

Note that, given an instance \((G = (A, B, E), M, L, R, k)\) of A-AGVC, in order to find a minimum vertex cover containing \(L \cup R\), it is sufficient to find the set \(M'\) of matched edges which have both end points in this minimum vertex cover. This follows from the fact that the graph \(G \setminus V(M')\) is then a König graph and has a minimum vertex cover that contains \((L \cup R) \setminus V(M')\). Thus, by Lemma 4.3.5, a minimum vertex cover of \(G \setminus V(M')\) containing \((L \cup R) \setminus V(M')\) can be computed in polynomial time. Hence, in the rest of the chapter, whenever we talk about a solution \(S\) for an instance of A-AGVC, we mean the set of edges of \(M\) which have both endpoints in the minimum vertex cover under consideration.

**Definition 5.3.1.** Given an instance \((G = (A, B, E), M, L, R, k)\) of A-AGVC we construct a directed graph \(D(G)\) corresponding to this instance as follows. Remove all the edges in \(G[A]\), orient all the edges of \(M\) from \(A\) to \(B\) and the remaining edges from \(B\) to \(A\).

An immediate observation to this is the following.

**Observation 5.3.2.** There is a path from \(u \in A\) to \(v \in B\) in \(D(G)\) if and only if there is an odd \(M\)-path from \(u\) to \(v\) in \(G\). Furthermore, for any set \(S\) of arcs in \(D(G)\) which correspond to edges of \(M\) in \(G\), there is a path from \(u \in A\) to \(v \in B\) in \(D(G) \setminus S\) if and only if there is an odd \(M\)-path from \(u\) to \(v\) in \(G \setminus V(S)\).

**Proof.** The first statement of the observation follows from the definition of \(D(G)\). For the proof of the second statement, suppose that there is a path from \(u \in A\) to \(v \in B\) in
D(G) \ S. Since the only out-neighbor of \( u \) in D(G) is the matching partner of \( u \) and the only in-neighbor of \( v \) in D(G) is the matching partner of \( v \), neither \( u \) nor \( v \) is in the set \( V(S) \). Therefore, the \( u \) to \( v \) path in D(G) \ S clearly corresponds to a \( u \) to \( v \) odd \( M \)-path in \( G \setminus V(S) \). The converse direction of the second statement simply follows from the definition of D(G).

Even though the edges of D(G) are directed (and henceforth will be called arcs), they originate in \( G \) and have a fixed direction. Hence we will occasionally use the same set of edges/arcs in both the undirected and directed sense. For example we may say that a set \( S \) of edges of \( G \) is both a solution for the corresponding instance (undirected) and an arc separator in the graph D(G) (directed).

The following lemma gives the first part of Phase 1, that is we show that there is a solution which corresponds to an \( X \)-\( Y \) separator for some sets \( X \) and \( Y \).

**Lemma 5.3.3.** Given an instance \( (G = (A, B, E), M, L, R, k) \) of A-AGVC, suppose that there is an L-R path in D(G) and let \( S \) be a solution to this instance. Then, \( S \) is an L-R arc separator in D(G).

**Proof.** Since \( S \) is a solution, the graph \( G \setminus V(S) \) is a König graph with a minimum vertex cover containing \((L \cup R) \setminus V(S)\). By Observation 5.3.2, we know that if there is an L-R path in D(G) \ S, then there is an odd \( M \)-path from \( L \) to \( R \) in \( G \setminus V(S) \) and by Lemma 4.2.5, this implies that there is no minimum vertex cover for \( G \setminus V(S) \) containing \((L \cup R) \setminus V(S)\), a contradiction.

The next lemma is used to handle the case when the instance does not have odd \( M \)-paths from \( L \) to \( R \).

**Lemma 5.3.4.** Let \( (G = (A, B, E), M, L, R, k) \) be an instance of A-AGVC such that there are no odd \( M \)-paths from \( L \) to \( R \) in \( G \). If there is an odd \( M \)-path \( P \) from \( R \) to \( R \)
(or an $R$-flower $\mathcal{P}$), then there is an edge $(u, v)$ such that $u, v \in A \setminus L$ and there is an odd $M$-path from $u$ to $R$ and an odd $M$-path from $v$ to $R$ such that any edge of $M$ which occurs in $P$ (respectively in $\mathcal{P}$) occurs in one of these two odd $M$-paths. Moreover, this edge can be found in time $O(mn)$.

**Proof.** Suppose $P = v_1, \ldots, v_t$ is an $R$ to $R$ odd $M$-path. Since $v_1, v_t \in R$, $v_1, v_t \in B$ and by Lemma 4.2.3(f) there is an edge $(u, v)$ such that $u, v \in A$ and there are odd $M$-paths from $u$ and $v$ to $v_1$ and $v_t$ respectively, which are odd $M$-paths from $u$ to $R$ and $v$ to $R$ respectively. Furthermore, any edge of $M$ present in $P$ occurs in one of these two odd $M$-paths.

Suppose that $\mathcal{P}$ is a flower with root $v_1 \in R$ and base $b$. Let $u_1$ and $u_2$ be the neighbors of $b$ in the blossom. We know by Lemma 4.2.3(c) that $b \in A$ and by Lemma 4.2.3(d) that at least one of $u_1$ and $u_2$ is in $B$. We first consider the case when $u_1, u_2 \in B$. Applying Lemma 4.2.3(f) on the blossom path from $u_1$ to $u_2$ (see Fig. 5.4) we know that there is an edge $(u, v)$ such that $u, v \in A$ and there are odd $M$-paths $P_1$ and $P_2$ from

Figure 5.4: An illustration of the two sub cases for the flower in Lemma 5.3.4. (a) $u_1, u_2 \in B$. (b) $u_1 \in A$, $u_2 \in B$. 

![Figure 5.4]
$u_1$ to $u$ and $u_2$ to $v$ respectively which lie inside the blossom path. Since $P_1$ and $P_2$ lie entirely within this blossom path, they do not intersect the stem of the flower. We also know by the definition of flowers that there are even $M$-paths $P_3$ and $P_4$ from the root to $u_1$ and $u_2$ respectively. Hence, $Rev(P_1 + P_3)$ and $Rev(P_2 + P_4)$ are odd $M$-paths from $u$ and $v$ respectively to $R$ and every edge of $M$ present in the stem of $\mathcal{P}$ occurs in $P_3$ and $P_4$ and every edge of $M$ present in the blossom of $\mathcal{P}$ occurs in $P_1$ or $P_2$.

We now consider the case when exactly one of $u_1$ and $u_2$, is in $A$. Suppose that $u_1 \in A$. We claim that there is an odd $M$-path from $u_1$ to $R$ and one from $b$ to $R$. By the definition of a flower, the stem, say $P$ is an odd $M$-path from $b$ to $R$. Let $P'$ be the blossom path from $u_1$ to $u_2$. Observe that $P' + (u_2, b) + Rev(P)$ is indeed an odd $M$-path from $u_1$ to $R$ and it contains every edge of $M$ which is present in $\mathcal{P}$.

In either of these cases, we have found an edge $(u, v)$ such that $u, v \in A$ and there are odd $M$-paths from both these vertices to $R$. Since we have assumed that there are no $L$ to $R$ odd $M$-paths, $u, v \in A \setminus L$. That we can find this edge in time $O(mn)$ follows from Observation 4.1.10.

Finally we come to the last part of Phase 1, where we show that the problem is solvable in polynomial time if none of the forbidden structures exist. This follows from Lemma 4.2.5, which states that if neither of the forbidden structures exist, then there is indeed a minimum vertex cover for $G$ containing $L \cup R$ and by Lemma 4.3.5, which gives a polynomial time algorithm to compute such a vertex cover.

Summarizing this section, we have shown that either there is a solution for the given instance which is an $L$-$R$ arc separator in $\text{D}(G)$ (Lemma 5.3.3), or there is an edge in $G$ on which we can branch and make progress (Lemma 5.3.4), or the problem is solvable in polynomial time (Lemma 4.3.5). This completes Phase 1 of the important separator.
template.

5.4 Phase 2

In this section, we demonstrate that Phase 2 of the important separator template is applicable by showing the existence of a dominating set of bounded size for the set of L-R arc separators in D(G).

Lemma 5.4.1. Given an instance \((G = (A, B, E), M, L, R, k)\) of A-AGVC, any important L-R arc separator in D(G) comprises precisely arcs corresponding to some subset of M.

Proof. Let \(X\) be an important L-R arc separator in D(G). Suppose there is an arc \(e = (b_j, a_i) \in X\) such that \(e \notin M\). The minimality of \(X\) implies that \(b_j\) and \(a_i\) are reachable from \(L\) in D(G) but only \(b_j\) is reachable from \(L\) in D(G) \(\setminus X\). Now, consider the set \(X' = (X \setminus e) \cup (a_i, b_i)\). Clearly \(|X'| \leq |X|\). Now, \(X'\) is also a minimal L-R arc separator in D(G) since any L-R path containing the arc \(e\) also contains the arc \((a_i, b_i)\). Now, \(a_i\) is reachable from \(L\) in D(G) \(\setminus X'\) and all vertices reachable from \(L\) in D(G) \(\setminus X\) are also reachable from \(L\) in D(G) \(\setminus X'\). Hence the set of vertices reachable from \(L\) in D(G) \(\setminus X'\) is a strict superset of the set of vertices reachable from \(L\) in D(G) \(\setminus X\). This contradicts our assumption that \(X\) was an important L-R arc separator. \(\square\)

Lemma 5.4.2. Let \((G = (A, B, E), M, L, R, k)\) be an instance of A-AGVC. If \((G = (A, B, E), M, L, R, k)\) is a YES instance, then it has a solution which contains an important L-R arc separator in D(G).

Proof. Let \(S\) be a solution for the given instance. By Lemma 7.4.3, we have that \(S\) is an L-R arc separator in D(G). Let \(S_{LR}\) be a minimal subset of \(S\) such that the graph D(G) \(\setminus S_{LR}\) does not contain a path from L to R.
Let \( K \) be the set of vertices reachable from \( L \) in \( D(G) \setminus S_{LR} \). If \( S_{LR} \) is an important \( L-R \) arc separator, we are done since \( S \) itself is a solution which satisfies our claim. If this were not the case, then there is an important \( L-R \) arc separator \( S'_{LR} \) such that \( |S'_{LR}| \leq |S_{LR}| \) and \( K' \supseteq K \) where \( K' \) is the set of vertices reachable from \( L \) in \( D(G) \setminus S'_{LR} \).

Consider the set \( \hat{S} = (S \setminus S_{LR}) \cup S'_{LR} \). We claim that \( \hat{S} \) is a solution which satisfies our claim. Clearly, \( |\hat{S}| \leq |S| \) and \( \hat{S} \) contains an important \( L-R \) arc separator in \( D(G) \).

By Lemma 5.4.1, \( S'_{LR} \subseteq M \) and hence \( \hat{S} \subseteq M \). Therefore, it remains to prove that \( \hat{S} \) is indeed a solution to the given instance.

If \( \hat{S} \) were not a solution, then \( \hat{G} = G \setminus V(\hat{S}) \) is a König graph that does not have a minimum vertex cover containing \((L \cup R) \setminus V(\hat{S})\). Lemma 4.2.5 implies that there is either an odd \( M \)-path from \( L \cup R \) to \( L \cup R \) or an \( R \)-flower in \( \hat{G} \). Since \( \hat{S} \) is an \( L-R \) arc separator in \( D(G) \), \( \hat{G} \) does not have odd \( M \)-paths from \( L \) to \( R \). Therefore there must be either an \( M \)-path from \( R \) to \( R \) or an \( R \)-flower in \( \hat{G} \). Let this odd \( M \)-path (or \( R \)-flower) be \( P \). Note that in either case, \( P \) must contain an edge in \( S'' = S_{LR} \setminus S'_{LR} \).

By Lemma 5.3.4, we know that there is an edge \((u, v)\) such that \( u, v \in A \) and there is an odd \( M \)-path from \( u \) to \( R \) and from \( v \) to \( R \) in \( \hat{G} \), where \( u, v \in A \). By Observation 5.3.2, this implies a path from \( u \) to \( R \), say \( P_1 \) and a path from \( v \) to \( R \), say \( P_2 \) in \( D(\hat{G}) \). By Lemma 5.3.4, we also know that every edge of \( M \) in \( P \) is present in \( P_1 \) or \( P_2 \). Suppose the edge \( m_j = (a_j, b_j) \) be an edge which is present in \( S_{LR} \setminus S'_{LR} \) and \( P \). Therefore, \( m_j \) is present in either \( P_1 \) or \( P_2 \). We assume without loss of generality that \( m_j \) is present in \( P_1 \). Since \( S'_{LR} \) dominates \( S_{LR} \) with respect to \( L \), it must be the case that \( a_j \) is reachable from \( L \) in \( D(G) \setminus S'_{LR} \) via a path, say \( P_3 \). Therefore, combining the path \( P_3 \) with the path \( P_1 \), we have a path from \( L \) to \( R \) in \( D(G) \) which is disjoint from \( S'_{LR} \), a contradiction. This concludes the proof of the lemma.

\[ \square \]

**Lemma 5.4.3.** Let \((G = (A, B, E), M, L, R, k)\) be an instance of A-AGVC and let
$D = D(G) = (V, A)$ be defined as above. Then the number of important $L$-$R$ arc separators of size at most $k$ in the graph $D$ is bounded by $4^k$.

**Proof.** Given $D, L, R, k \geq 0$ we define a measure $\mu_a(D, L, R, k) = 2k - \lambda_D(L, R)$. We prove by induction on $\mu_a(D, L, R, K)$ that there are at most $2^\mu_a(D, L, R, k)$ important $L$-$R$ arc separators of size at most $k$. For the base case, if $2k - \lambda_D(L, R) < k$ then $\lambda_D(L, R) > k$ and hence the number of important $L$-$R$ arc separators of size at most $k$ is 0. If $\lambda_D(L, R) = 0$, it means that there is no path from $L$ to $R$ and hence the empty set is the only important $L$-$R$ arc separator. Consider $D, L, R, k \geq 0$ such that $\mu_a(D, L, R, k) \geq k, \lambda_D(L, R) > 0$ and assume that the statement of the Lemma holds for all $D', L', R', k'$ where $\mu_a(D', L', R', k') < \mu_a$.

By Lemma 3.2.26, there is a unique important $L$-$R$ arc separator $S^*$ of size $\lambda_D(L, R)$. Since we have assumed $\lambda_D(L, R)$ to be positive, $S^*$ is non empty. Consider an arc $e = (u, v) \in S^*$. By Lemma 5.4.1, there is some $i$ such that $u = a_i$ and $v = b_i$. Any important $L$-$R$ arc separator $S$ either contains $e$ or does not contain $e$. For any important $L$-$R$ arc separator $S$ which contains $e$, $S \setminus \{e\}$ is an important $L$-$R$ arc separator in $D \setminus \{e\}$ by Lemma 3.2.30(2). Hence the number of important $L$-$R$ arc separators of size at most $k$ in $D$ which contain $e$, is at most the number of important $L$-$R$ arc separators of size at most $k - 1$ in $D \setminus \{e\}$. Observe that $\lambda_{D \setminus \{e\}}(L, R) = \lambda_D(L, R) - 1$ which implies that $\mu_a(D \setminus \{e\}, L, R, k - 1) < \mu_a$ and by the induction hypothesis, the number of important $L$-$R$ arc separators of size at most $k - 1$ in $D \setminus \{e\}$ is bounded by $2^{\mu_a - 1}$ which is also a bound on the number of important $L$-$R$ arc separators of size at most $k$ in $D$ which contain $e$.

Now let $S$ be an important $L$-$R$ arc separator of size at most $k$ which does not contain $e$. By Lemma 3.2.29 we know that $K_S \supseteq K_{S^*}$ and by the minimality of $S^*$, $a_i \in K_{S^*}$ and since $K_S \supseteq K_{S^*}, a_i$ is in $K_S$. But $e \notin S$, which implies that $b_i$ is in $K_S$ which
implies that \( K_S \supseteq K_{S'} \cup \{b_i\} \). But by Lemma 5.4.1, no other edge incident on \( b_i \) can be in \( S \). Hence the vertices in \( \delta^+(b_i) \) are also reachable from \( L \) in \( D \setminus S \). We now set \( X = K_{S'} \cup \delta^+(b_i) \) and by Lemma 3.2.30(3) we know that \( S \) is also an important \( X-R \) arc separator. We set \( L' = A \cap X \). Since there cannot be paths from \( R \) to \( R \) in \( D(G) \), any \( X-R \) separator is also an \( L'-R \) separator. Thus a bound on the number of important \( L'-R \) arc separators of size at most \( k \) is also a bound on the number of important \( L-R \) arc separators of size at most \( k \) which do not contain the arc \( e \). First note that \( \lambda_D(L', R) > \lambda_D(L, R) \) since otherwise we would have an \( L-R \) arc separator \( S' \) of size at most \( S' \) such that \( K_{S'} \supseteq K_{S'} \). Now, \( \mu_a(D, L', R, k) < \mu_a \) and by the induction hypothesis, the number of important \( L'-R \) arc separators of size at most \( k \) is bounded by \( 2^{2^{\mu_a-1}} \).

Summing up the upper bounds in both cases, the number of important \( L-R \) arc separators of size at most \( k \) is bounded by \( 2 \cdot 2^{\mu_a-1} = 2^{\mu_a} \leq 2^{2k} \).

We have thus shown that the set of important \( L-R \) arc separators of size at most \( k \) is indeed a dominating set for the set of \( L-R \) arc separators of size at most \( k \) with respect to this problem (Lemma 5.4.2) and that the size of the dominating set is bounded by \( 4^k \) (Lemma 5.4.3). Therefore, there is a solution which intersects the set of \( 4^k \) vertices in the union of the vertices in all important separators of size at most \( k \). This completes Phase 2 of the important separator template.

5.5 Phase 3

In this section, we show that the dominating set, that is, the set of important \( L-R \) arc separators defined in the previous section can be computed in \( \text{FPT} \) time. The proof is the same as that seen in Chapter 3. However, since the analysis of our final algorithm
will require us to take a closer look at how this particular algorithm is interlinked with the rest of the steps, we have presented the algorithm to compute the set of important arc separators in terms of the problem at hand.

**Lemma 5.5.1.** The set of important L-R arc separators of size at most \( k \) can be enumerated in time \( O(4^k k(m + n)) \).

**Proof.** The algorithm for enumerating the important L-R arc separators of size at most \( k \) follows from the proof of Lemma 5.4.3. The algorithm first computes the unique smallest important L-R separator \( S^* \) using the algorithm described in Lemma 3.2.28, selects an arc \( e \in S^* \) and recursively enumerates all important L-R arc separators which contain \( e \) and those which do not. It follows from the proof of Lemma 5.4.3 that this algorithm runs in time \( O(4^k k(m + n)) \).

Thus, Lemma 5.5.1 gives an algorithm running in time \( O(4^k k(m + n)) \) to compute the dominating set. This completes Phase 3.

### 5.6 Phase 4

Finally, we combine the first three phases (summarized in Fig. 9.7) to obtain our algorithm for A-AGVC.

We are now ready to prove Lemma 5.2.2 by describing an algorithm (Algorithm. 5.6.1) for A-AGVC. In order to make the analysis of our algorithm simpler (and stronger), we embed the algorithm for enumerating important separators (Lemma 5.4.3) into our algorithm for A-AGVC.

**Correctness.** The Correctness of Step 1 is obvious. In Steps 6 and 8 we are merely guessing the vertex which covers the edge \((u, v)\), while Step 11 is correct due to Lemma 4.2.5.
Algorithm 5.6.1: Algorithm $Solve - AAGVC$ for A-AGVC

- **Input**: An instance $(G, M, L, R, k)$ of A-AGVC
- **Output**: A solution of size at most $k$ for the instance $(G, M, L, R, k)$ if it exists and No otherwise

1. If $k < 0$ then return No

2. Compute a minimum size $L-R$ arc separator $S$ in the directed graph $D(G)$

3. If $|S| = 0$ then
   - If there an odd $M$-path from $R$ to $R$ or an $R$-flower then
     - Compute the edge $e = (u, v)$ given by Lemma 5.3.4
     - $S_1 \leftarrow Solve - AAGVC(G, M, L \cup \{u\}, R, k)$
     - If $S_1$ is not No then return $S_1$
     - $S_2 \leftarrow Solve - AAGVC(G, L \cup \{v\}, R, k)$
     - Return $S_2$
   - Else return $\phi$

4. End

5. If $|S| > k$ then return No

6. Else
   - Compute the unique minimum size important $L-R$ separator $S^*$ in $D(G)$ (Lemma 3.2.28) and select an arc $e = (w, z) \in S^*$
   - $S_3 \leftarrow Solve - AAGVC(G \setminus \{e\}, M \setminus \{e\}, L, R, k - 1)$
   - If $S_3$ is not No then return $S_3 \cup \{e\}$
   - $S_4 \leftarrow Solve - AAGVC(G, M, A \cap (\delta^+_D(G)(z) \cup K_{S^*}), R, k)$
   - Return $S_4$

7. End
Phase 1.

(a) If there is an odd $M$-path from $L$ to $R$ in $G$, then the solution contains an $L$-$R$ arc separator in $D(G)$ (Lemma 5.3.3).

(b) If there are no odd $M$-paths from $L$ to $R$, but there is either an odd $M$-path from $R$ to $R$ or an $R$-flower, then there exists a branch-able edge $(u, v)$ between two vertices in $A$ which can be computed in time $O(mn)$ (Lemma 5.3.4).

(c) If neither of these two cases occur, then the graph already has a minimum vertex cover containing $L \cup R$ and this can be computed in time $O(mn)$ (Lemma 4.2.5).

Phase 2. If there is a solution containing an $L$-$R$ arc separator in $D(G)$, there is one which contains an important $L$-$R$ arc separator in $D(G)$ (Lemma 5.4.2). The number of important $L$-$R$ arc separators of size at most $k$ is at most $4^k$ (Lemma 5.4.3) and hence the number of vertices in their union is at most $4^k k$.

Phase 3. The set of important $L$-$R$ arc separators of size at most $k$ in $D(G)$ can be enumerated in time $O(4^k kmn)$ (Lemma 5.5.1).

Step 13 is correct because the size of the minimum $L$-$R$ separator in $D(G)$ is a lower bound on the solution size. Steps 15 and 17 are part of enumerating the important $L$-$R$ arc separators as seen in Lemma 5.4.3. Since we have shown in Lemma 5.4.2 that if there is a solution, there is one which contains an important $L$-$R$ separator in $D(G)$, these steps are also correct.

Running Time. In order to analyze the algorithm, we define the search tree $T(G, M, L, R, k)$ resulting from a call to $Solve - AAGVC(G, M, L, R, k)$ inductively as follows. The tree $T(G, M, L, R, k)$ is a rooted tree whose root node corresponds to the instance $(G, M, L, R, k)$. If $Solve - AAGVC(G, M, L, R, k)$ does not make a recursive call, then $(G, M, L, R, k)$ is said to be the only node of this tree. If it does make recursive calls, then the children of $(G, M, L, R, k)$ correspond to the instances given as input to the recursive calls made inside the current procedure call. The subtree rooted at a child node $(G', M', L', R', k')$ is the search tree $T(G', M', L', R', k')$. 

75
Given an instance $I = (G, M, L, R, k)$, we prove by induction on $\mu(I) = 2k - \lambda_{D(G)}(L, R)$ that the number of leaves of the tree $T(I)$ is bounded by $\max\{2^{\mu(I)}, 1\}$. In the base case, if $\mu(I) < k$, then $\lambda(L, R) > k$ in which case the number of leaves is 1. Assume that $\mu(I) \geq k$ and our claim holds for all instances $I'$ such that $\mu(I') < \mu(I)$.

Suppose $\lambda(L, R) = 0$. In this case, the children $I_1$ and $I_2$ of this node correspond to the recursive calls made in Steps 6 and 8. By Lemma 5.3.4 there are odd $M$-paths from $u$ to $R$ and from $v$ to $R$. Hence, $\lambda(L \cup \{u\}, R) > 0$ and $\lambda(L \cup \{v\}, R) > 0$. This implies that $\mu(I_1), \mu(I_2) < \mu(I)$. By the induction hypothesis, the number of leaves in the search trees rooted at $I_1$ and $I_2$ are at most $2^{\mu(I_1)}$ and $2^{\mu(I_2)}$ respectively. Hence the number of leaves in the search tree rooted at $I$ is at most $2^{2^{\mu(I)}-1} = 2^{2^{\mu(I)}}$.

Suppose $\lambda(L, R) > 0$. In this case, the children $I_1$ and $I_2$ of this node correspond to the recursive calls made in Steps 15 and 17. But in these two cases, as seen in the proof of Lemma 5.4.3, $\mu(I_1), \mu(I_2) < \mu(I)$ and hence by applying the induction hypothesis on the two child nodes and summing up the number of leaves in the subtrees rooted at each, we can bound the number of leaves in the subtree of $I$ by $2^{\mu(I)}$.

Therefore, the number of leaves of the search tree $T$ rooted at the input instance $I = (G, M, L, R, k)$ is $2^{\mu(I)} \leq 2^{2k}$. The time spent at any internal node of the search tree is bounded by $O(k(m + n) + mn)$ since at every node, we either find an $M$-path from $R$ to $R$ or an $R$-flower or find a minimum $L-R$ arc separator if one of size at most $k$ exists. Therefore the running time of the algorithm is $O(4^k kmn)$. This completes the proof of Lemma 5.2.2.

### 5.7 Corollaries

The following corollaries are obtained using known FPT reductions [88, 81, 44].
**Lemma 5.7.1.** Almost 2-SAT can be solved in time $O(9^knm^2)$.

**Lemma 5.7.2.** König Vertex Deletion on graphs with a perfect matching can be solved in time $O(3^knm^2)$.

**Lemma 5.7.3.** RHorn Backdoor Detection can be solved in time $O(9^knm^2)$.
6

Vertex Cover Parameterized Above LP Optimum

6.1 Introduction

The well known integer linear programming formulation (ILP) for VERTEX COVER is as follows.

**ILP FORMULATION OF MINIMUM VERTEX COVER – ILPVC**

*Instance:* A graph $G = (V, E)$.

*Feasible Solution:* A function $x : V \rightarrow \{0, 1\}$ satisfying edge constraints

$$x(u) + x(v) \geq 1 \text{ for each edge } (u, v) \in E.$$ 

*Goal:* To minimize $w(x) = \Sigma_{u \in V} x(u)$ over all feasible solutions $x$.

In the standard linear programming relaxation of the above ILP, the constraint $x(v) \in \{0, 1\}$ is replaced with $x(v) \geq 0$, for all $v \in V$. For a graph $G$, we call this relaxation LPVC($G$). Clearly, every integer feasible solution is also a feasible solution to LPVC($G$). Clearly the size of a minimum vertex cover of a graph is at least the min-
Table 6.1: The table gives the previous $f(k)$ bound in the running time of various problems and the ones obtained in this paper.

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Previous $f(k)$/Reference</th>
<th>New $f(k)$ in this paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGVC</td>
<td>$4^k$ [23]</td>
<td>$2.3146^k$</td>
</tr>
<tr>
<td>ALMOST 2-SAT</td>
<td>$4^k$ [23]</td>
<td>$2.3146^k$</td>
</tr>
<tr>
<td>RHORN-BACKDOOR SET DETECTION</td>
<td>$4^k$ [23, 44]</td>
<td>$2.3146^k$</td>
</tr>
<tr>
<td>KÖNIG VERTEX DELETION</td>
<td>$4^k$ [23, 81]</td>
<td>$1.5214^k$</td>
</tr>
<tr>
<td>SPLIT VERTEX DELETION</td>
<td>$5^k$ [12]</td>
<td>$2.3146^k$</td>
</tr>
<tr>
<td>ODD CYCLE TRANSVERSAL</td>
<td>$3^k$ [91]</td>
<td>$2.3146^k$</td>
</tr>
<tr>
<td>VERTEX COVER PARAM BY OCT</td>
<td>$2^k$ (folklore)</td>
<td>$1.5214^k$</td>
</tr>
<tr>
<td>VERTEX COVER PARAM BY KVD</td>
<td>–</td>
<td>$1.5214^k$</td>
</tr>
</tbody>
</table>

minimum value of LPVC for the graph. This allows us to parameterize VERTEX COVER above the minimum value of LPVC for the input graph. Prior to our study, a similar parameterization had recently been studied by Cygan et al. [23] in the context of the MULTIWAY CUT problem, as a consequence of which they obtained an $O(4^k n^{O(1)})$ algorithm for VERTEX COVER parameterized above the size of the maximum matching.

In this chapter, we develop a $O(2.3146^{(k−vc^∗(G))} n^{O(1)})$ time branching algorithm for VERTEX COVER above LP. In an effort to present the key ideas of our algorithm in as clear a way as possible, we first present a simpler and slightly slower algorithm in Section 6.3. This algorithm exhaustively applies a collection of previously known preprocessing steps. If no further preprocessing is possible the algorithm simply selects an arbitrary vertex $v$ and recursively tries to find a vertex cover of size at most $k$ by considering whether $v$ is in the solution or not. While the algorithm is simple, the analysis is more involved as it is not obvious that the measure $k−vc^∗(G)$ actually drops in the recursive calls. In order to prove that the measure does drop we string together several known results about the linear programming relaxation of VERTEX COVER, such as the classical Nemhauser-Trotter theorem and properties of “minimum surplus sets”. We find it
intriguing that, as our analysis shows, combining well-known reduction rules with naive branching yields fast FPT algorithms for all problems in Figure 10.1. We then show in Section 6.4 that adding several more involved branching rules to our algorithm yields an improved running time of $O(2.3146^{k-vc^*(G)})n^{O(1)}$. Using this algorithm we obtain even faster algorithms for the problems in Figure 10.1.

We give a list of problems with their previous best running time and the ones obtained in this paper in Table 6.1. The most notable among them is the new algorithm for ODD CYCLE TRANSVERSAL, the problem of deleting at most $k$ vertices to obtain a bipartite graph. The parameterized complexity of ODD CYCLE TRANSVERSAL was a long standing open problem in the area, and only in 2003 Reed et al. [91] developed an algorithm for the problem running in time $O^*(3^k)$. However, there has been no further improvement over this algorithm in the last 9 years; though reinterpretations of the algorithm have been published [49, 69].

We also find the algorithm for KÖNIG VERTEX DELETION, the problem of deleting at most $k$ vertices to obtain a König graph very interesting. KÖNIG VERTEX DELETION is a natural variant of the odd cycle transversal problem. In [81] it was shown that given a minimum vertex cover one can solve KÖNIG VERTEX DELETION in polynomial time. In this article we show a relationship between the measure $k - vc^*(G)$ and the minimum number of vertices needed to delete to obtain a König graph. This relationship together with a reduction rule for KÖNIG VERTEX DELETION based on the Nemhauser-Trotter theorem gives an algorithm for the problem with running time $O^*(1.5214^k)$.

We also note that using our algorithm, we obtain a polynomial time algorithm for VERTEX COVER that, given an input $(G, k)$ returns an equivalent instance $(G' = (V', E'), k')$ such that $k' \leq k$ and $|V(G')| \leq 2k - c \log k$ for any fixed constant $c$. This is known as a kernel for VERTEX COVER in the literature. We note that this kernel is simpler than
another kernel with the same size bound [65].

**Organization of the chapter.** In Section 6.3, we give a simple branching algorithm and introduce the nature of analysis which will be required for such algorithms. In Section 6.4, we present our main algorithm using much more involved branching rules. In Section 10.4, we give the applications of our result to numerous other parameterized problems.

### 6.2 Preliminaries

The surplus of an independent set $X \subseteq V$ is defined as $\text{surplus}(X) = |N(X)| - |X|$. For a set $\mathcal{A}$ of independent sets of a graph, $\text{surplus}(\mathcal{A}) = \min_{X \in \mathcal{A}} \text{surplus}(X)$.

The surplus of a graph $G$, $\text{surplus}(G)$, is defined to be the minimum surplus over all independent sets in the graph.

By the phrase “an optimum solution to LPVC($G$)”, we mean a feasible solution with $x(v) \geq 0$ for all $v \in V$ minimizing the objective function $w(x) = \sum_{u \in V} x(u)$. It is well known that for any graph $G$, there exists an optimum solution to LPVC($G$), such that $x(u) \in \{0, \frac{1}{2}, 1\}$ for all $u \in V$ [83]. Such a feasible optimum solution to LPVC($G$) is called a half integral solution and can be found in polynomial time [83]. In this chapter we always deal with half integral optimum solutions to LPVC($G$). Thus, by default whenever we refer to an optimum solution to LPVC($G$) we will be referring to a half integral optimum solution to LPVC($G$). Furthermore, it is also known that the modified LP resulting from forcing certain variables to a value in $\{0, \frac{1}{2}, 1\}$ also has a half integral optimum solution. Let $VC(G)$ be the set of all minimum vertex covers of $G$ and $vc(G)$ denote the size of a minimum vertex cover of $G$. Let $VC^\ast(G)$ be the set of all optimal solutions (including non half integral optimal solution) to LPVC($G$). By $vc^\ast(G)$ we
denote the value of an optimum solution to LPVC($G$). We define $V_i^x = \{u \in V : x(u) = i\}$ for each $i \in \{0, \frac{1}{2}, 1\}$ and define $x \equiv i$, $i \in \{0, \frac{1}{2}, 1\}$, if $x(u) = i$ for every $u \in V$. Clearly, $vc(G) \geq vc^*(G)$ and $vc^*(G) \leq \frac{|V|}{2}$ since $x \equiv \frac{1}{2}$ is always a feasible solution to LPVC($G$). We also refer to the $x \equiv \frac{1}{2}$ solution simply as the all $\frac{1}{2}$ solution.

In branching algorithms, we say that a branching step results in a drop of $(p_1, p_2, ..., p_l)$ where $p_i, 1 \leq i \leq l$ is an integer, if the measure we use in the analysis drops respectively by $p_1, p_2, ..., p_l$ in the corresponding branches. We also call the vector $(p_1, p_2, \ldots, p_l)$ the branching vector of the step.

### 6.3 A Simple Algorithm for VERTEX COVER ABOVE LP

In this section, we give a simpler algorithm for VERTEX COVER ABOVE LP. The algorithm has two phases, a preprocessing phase and a branching phase. We first describe the preprocessing steps used in the algorithm and then give a simple description of the algorithm. Finally, we prove its correctness and the bound on the running time of the algorithm.

#### 6.3.1 Preprocessing

We describe three standard preprocessing rules to simplify the input instance. We first state the (known) results which allow for their correctness, and then describe the rules.

**Lemma 6.3.1.** [84, 87] For a graph $G$, in time $O(m\sqrt{n})$, we can compute an optimal solution $x$ to LPVC($G$) where $n$ is the number of vertices in $G$ and $m$ is the number of edges. Furthermore, in time $O(mn\sqrt{n})$, we can compute an optimal solution $x$ to LPVC($G$) such that all $\frac{1}{2}$ is the unique optimal solution to LPVC($G[V_{1/2}]$). Furthermore, $\text{surplus}(G[V_{1/2}]) > 0$.

83
Lemma 6.3.2. [84] Let $G$ be a graph and $x$ be an optimal solution to LPVC($G$). There is a minimum vertex cover for $G$ which contains all the vertices in $V^x_1$ and none of the vertices in $V^x_0$.

Preprocessing Rule 1. Apply Lemma 6.3.1 to compute an optimal solution $x$ to LPVC($G$) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V^x_1/2]$). Delete the vertices in $V^x_0 \cup V^x_1$ from the graph after including $V^x_1$ in the vertex cover we develop, and reduce $k$ by $|V^x_1|$.

In the discussions in the rest of the chapter, we say that Preprocessing Rule 1 applies (or is applicable) if all $\frac{1}{2}$ is not the unique solution to LPVC($G$) and that it doesn’t apply (or is not applicable) if all $\frac{1}{2}$ is the unique solution to LPVC($G$).

The soundness/correctness of Preprocessing Rule 1 follows from Lemma 6.3.2. The time required to check if it is applicable and to apply it is $O(mn\sqrt{n})$. After the application of Preprocessing Rule 1, we know that $x \equiv \frac{1}{2}$ is the unique optimal solution to LPVC of the resulting graph and the graph has a surplus of at least 1.

Lemma 6.3.3. [15, 84] Let $G(V, E)$ be a graph, and let $S \subseteq V$ be an independent subset such that $\text{surplus}(Y) \geq \text{surplus}(S)$ for every set $Y \subseteq S$. Then there exists a minimum vertex cover for $G$ that contains either all of $S$ or none of $S$. In particular, if $S$ is an independent set with the minimum surplus, then there exists a minimum vertex cover for $G$, that contains all of $S$ or none of $S$.

The following lemma, which handles without branching, the case when the minimum surplus of the graph is 1, follows from the above lemma.

Lemma 6.3.4. [15, 84] Let $G$ be a graph, and let $Z \subseteq V(G)$ be an independent set such that $\text{surplus}(Z) = 1$ and for every $Y \subseteq Z$, $\text{surplus}(Y) \geq \text{surplus}(Z)$. Then,
1. If the graph induced by $N(Z)$ is not an independent set, then there exists a minimum vertex cover in $G$ that includes all of $N(Z)$ and excludes all of $Z$.

2. If the graph induced by $N(Z)$ is an independent set, let $G'$ be the graph obtained from $G$ by removing $Z \cup N(Z)$ and adding a vertex $z$, followed by making $z$ adjacent to every vertex $v \in G \setminus (Z \cup N(Z))$ which was adjacent to a vertex in $N(Z)$ (also called identifying the vertices of $N(Z)$). Then, $G$ has a vertex cover of size at most $k$ if and only if $G'$ has a vertex cover of size at most $k - |Z|$.

We now give two preprocessing rules to handle the case when the surplus of the graph is 1.

**Preprocessing Rule 2.** If there is a set $Z$ such that $\text{surplus}(Z) = 1$ and $N(Z)$ is not an independent set, then apply Lemma 6.3.4 to reduce the instance as follows. Include $N(Z)$ in the vertex cover, delete $Z \cup N(Z)$ from the graph, and decrease $k$ by $|N(Z)|$.

**Preprocessing Rule 3.** If there is a set $Z$ such that $\text{surplus}(Z) = 1$ and $N(Z)$ is an independent set, then apply Lemma 6.3.4 to reduce the instance as follows. Remove $Z$ from the graph, identify the vertices of $N(Z)$, and decrease $k$ by $|Z|$.

The correctness of Preprocessing Rules 2 and 3 follows from Lemma 6.3.4. The entire preprocessing phase of the algorithm is summarized in Figure 6.1. Recall that each preprocessing rule can be applied only when none of the preceding rules are applicable, and that Preprocessing Rule 1 is applicable if and only if there is a solution to LPVC($G$) which does not assign $\frac{1}{2}$ to every vertex. Hence, when Preprocessing Rule 1 does not apply, all $\frac{1}{2}$ is the unique solution for LPVC($G$). We now show that we can test whether Preprocessing Rules 2 and 3 are applicable on the current instance in polynomial time.
Lemma 6.3.5. Given an instance \((G, k)\) of VERTEX COVER ABOVE LP on which Preprocessing Rule 1 does not apply, we can test if Preprocessing Rule 2 applies on this instance in time \(O(m^2 \sqrt{n})\).

Proof. We first prove the following claim.

Claim 1. The graph \(G\) (in the statement of the lemma) contains a set \(Z\) such that \(\text{surplus}(Z) = 1\) and \(N(Z)\) is not independent if and only if there is an edge \((u, v) \in E\) such that solving \(LPVC(G)\) with \(x(u) = x(v) = 1\) results in a solution with value exactly \(\frac{1}{2}\) greater than the value of the original \(LPVC(G)\).

Proof. Suppose there is an edge \((u, v)\) such that \(w(x') = w(x) + \frac{1}{2}\) where \(x\) is the solution to the original \(LPVC(G)\) and \(x'\) is the solution to \(LPVC(G)\) with \(x'(u) = x'(v) = 1\) and let \(Z = V_0 x'\). We claim that the set \(Z\) is a set with surplus 1 and that \(N(Z)\) contains vertices \(u\) and \(v\), \(N(Z)\) is not an independent set. Since \(N(Z)\) contains the vertices \(u\) and \(v\), \(N(Z)\) is not an independent set. Now, since \(x \equiv \frac{1}{2}\) (Preprocessing Rule 1 does not apply), \(w(x') = w(x) - \frac{1}{2}|Z| + \frac{1}{2}|N(Z)| = w(x) + \frac{1}{2}\). Hence, \(|N(Z)| - |Z| = \text{surplus}(Z) = 1\).

Conversely, suppose that there is a set \(Z\) such that \(\text{surplus}(Z) = 1\) and \(N(Z)\) contains vertices \(u\) and \(v\) such that \((u, v) \in E\). Let \(x'\) be the assignment which assigns 0 to all vertices in \(Z\), 1 to all vertices in \(N(Z)\) and \(\frac{1}{2}\) to the rest of the vertices. Clearly, \(x'\) is a feasible assignment and \(w(x') = |N(Z)| + \frac{1}{2}|V \setminus (Z \cup N(Z))|\). Since Preprocessing Rule 1 does not apply, \(w(x') - w(x) = |N(Z)| - \frac{1}{2}(|Z| + |N(Z)|) = \frac{1}{2}(|N(Z)| - |Z|) = \frac{1}{2}\), which proves the converse part of the claim.

Given the above claim, we check if Preprocessing Rule 2 applies by doing the following for every edge \((u, v)\) in the graph. Set \(x(u) = x(v) = 1\) and solve the resulting LP looking for a solution whose optimum value is exactly \(\frac{1}{2}\) more than the optimum value of \(LPVC(G)\). The time required to check for applicability and to apply the rule is
bounded by \( m \) times the time to compute an optimum solution to LPVC\((G)\), which is \( O(m^2 \sqrt{n}) \).

**Lemma 6.3.6.** Given an instance \((G, k)\) of \textsc{Vertex Cover Above LP} on which Preprocessing Rules 1 and 2 do not apply, we can test if Preprocessing Rule 3 applies on this instance in time \( O(mn \sqrt{n}) \).

**Proof.** We first prove a claim analogous to that proved in the previous lemma.

**Claim 2.** The graph \( G \) (in the statement of the lemma) contains a set \( Z \) such that \( \text{surplus}(Z) = 1 \) and \( N(Z) \) is independent if and only if there is a vertex \( u \in V \) such that solving LPVC\((G)\) with \( x(u) = 0 \) results in a solution with value exactly \( \frac{1}{2} \) greater than the value of the original LPVC\((G)\).

**Proof.** Suppose there is a vertex \( u \) such that \( w(x') = w(x) + \frac{1}{2} \) where \( x \) is the solution to the original LPVC\((G)\) and \( x' \) is the solution to LPVC\((G)\) with \( x'(u) = 0 \) and let \( Z = V_0^{x'} \). We claim that the set \( Z \) is a set with surplus 1 and that \( N(Z) \) is independent.

Since \( x \equiv \frac{1}{2} \) (Preprocessing Rule 1 does not apply), \( w(x') = w(x) - \frac{1}{2} |Z| + \frac{1}{2} |N(Z)| = w(x) + \frac{1}{2} \). Hence, \( |N(Z)| - |Z| = \text{surplus}(Z) = 1 \). Since Preprocessing Rule 2 does not apply, it must be the case that \( N(Z) \) is independent.

Conversely, suppose that there is a set \( Z \) such that \( \text{surplus}(Z) = 1 \) and \( N(Z) \) is independent. Let \( x' \) be the assignment which assigns 0 to all vertices of \( Z \) and 1 to all vertices of \( N(Z) \) and \( \frac{1}{2} \) to the rest of the vertices. Clearly, \( x' \) is a feasible assignment and \( w(x') = |N(Z)| + \frac{1}{2} |V \setminus (Z \cup N(Z))| \). Since Preprocessing Rule 1 does not apply, \( w(x') - w(x) = |N(Z)| - \frac{1}{2} (|Z| + |N(Z)|) = \frac{1}{2} (|N(Z)| - |Z|) = \frac{1}{2} \). This proves the converse part of the claim with \( u \) being any vertex of \( Z \).

Given the above claim, we check if Preprocessing Rule 3 applies by doing the following for every vertex \( u \) in the graph. Set \( x(u) = 0 \), solve the resulting LP and look for
The rules are applied in the order in which they are presented, that is, any rule is applied only when none of the earlier rules are applicable.

**Preprocessing rule 1:** Apply Lemma 6.3.1 to compute an optimal solution $x$ to $\text{LPVC}(G)$ such that all $\frac{1}{2}$ is the unique optimum solution to $\text{LPVC}(G[V^x_{1/2}])$. Delete the vertices in $V^x_0 \cup V^x_1$ from the graph after including $V^x_1$ in the vertex cover we develop, and reduce $k$ by $|V^x_1|$.

**Preprocessing rule 2:** Apply Lemma 6.3.5 to test if there is a set $Z$ such that $\text{surplus}(Z) = 1$ and $N(Z)$ is not an independent set. If such a set does exist, then we apply Lemma 6.3.4 to reduce the instance as follows. Include $N(Z)$ in the vertex cover, delete $Z \cup N(Z)$ from the graph, and decrease $k$ by $|N(Z)|$.

**Preprocessing rule 3:** Apply Lemma 6.3.6 to test if there is a set $Z$ such that $\text{surplus}(Z) = 1$ and $N(Z)$ is an independent set. If there is such a set $Z$ then apply Lemma 6.3.4 to reduce the instance as follows. Remove $Z$ from the graph, identify the vertices of $N(Z)$, and decrease $k$ by $|Z|$.

Figure 6.1: Preprocessing Steps

a solution whose optimum value exactly $\frac{1}{2}$ more than the optimum value of $\text{LPVC}(G)$. The time required to check for applicability and to apply the rule is bounded by $n$ times the time to compute an optimum solution to $\text{LPVC}(G)$, which is $O(mn\sqrt{n})$. \hfill $\Box$

**Definition 6.3.7.** For a graph $G$, we denote by $\mathcal{R}(G)$ the graph obtained after applying Preprocessing Rules 1, 2 and 3 exhaustively in this order.

Strictly speaking $\mathcal{R}(G)$ is not a well defined function since the reduced graph could depend on which sets the reduction rules are applied on, and these sets, in turn, depend on the solution to the LP. To overcome this technicality we let $\mathcal{R}$ be a function not only of the graph $G$ but also of the representation of $G$ in memory. Since our reduction rules are deterministic (and the LP solver we use as a black box is deterministic as well), running the reduction rules on (a specific representation of) $G$ will always result in the same graph, making the function $\mathcal{R}(G)$ well defined. Finally, observe that for any $G$
the all $\frac{1}{2}$ solution is the unique optimum solution to the LPVC($\mathcal{R}(G)$) and $\mathcal{R}(G)$ has a surplus of at least 2.

### 6.3.2 Branching

After the preprocessing rules are applied exhaustively, we pick an arbitrary vertex $u$ in the graph and branch on it. In other words, in one branch, we add $u$ into the vertex cover, decrease $k$ by 1, and delete $u$ from the graph, and in the other branch, we add $N(u)$ into the vertex cover, decrease $k$ by $|N(u)|$, and delete $\{u\} \cup N(u)$ from the graph. The correctness of this algorithm follows from the soundness of the preprocessing rules and the fact that the branching is exhaustive.

### 6.3.3 Analysis

In order to analyze the running time of our algorithm, we define a measure $\mu = \mu(G, k) = k - \text{vc}^*(G)$. We first show that our preprocessing rules do not increase this measure. Following this, we will prove a lower bound on the decrease in the measure occurring as a result of the branching, thus allowing us to bound the running time of the algorithm in terms of the measure $\mu$. For each case, we let $(G', k')$ be the instance resulting by the application of the preprocessing rule or branch, and let $x'$ be an optimum solution to LPVC($G'$).

1. Consider the application of Preprocessing Rule 1. We know that $k' = k - |V_{1/2}^x|$. Since $x' \equiv \frac{1}{2}$ is the unique optimum solution to LPVC($G'$), and $G'$ comprises precisely the vertices of $V_{1/2}^x$, the value of the optimum solution to LPVC($G'$) is exactly $|V_{1/2}^x|$ less than that of $G$. Hence, $\mu(G, k) = \mu(G', k')$.

2. We now consider the application of Preprocessing Rule 2 and let $V'$ be the set of vertices in the graph resulting from the application of the rule. We know that
We now consider the application of Preprocessing Rule 3. We know that $N(Z)$ was not independent. In this case, $k' = k - |N(Z)|$. We also know that $w(x') = \sum_{u \in V} x'(u) = w(x) - \frac{1}{2}(|Z| + |N(Z)|) + \frac{1}{2}(|\mathcal{V}_1 - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(Z)|)$, we get $w(x') = w(x) - |N(Z)| + \frac{1}{2}(|N(Z)| - |Z|) + \frac{1}{2}(|\mathcal{V}_1 - |V_0^{x'}|)$. But, $Z \cup V_0^{x'}$ is an independent set in $G$, and $N(Z \cup Z_0^{x'}) = N(Z) \cup V_1^{x'}$ in $G$. Since surplus$(G) \geq 1$, $|N(Z \cup Z_0^{x'})| - |Z \cup V_0^{x'}| \geq 1$. Hence, $w(x') = w(x) - |N(Z)| + \frac{1}{2}(|N(Z \cup V_0^{x'})| - |Z \cup V_0^{x'}|) \geq w(x) - |N(Z)| + \frac{1}{2}$.

Thus, $\mu(G', k') \leq \mu(G, k) - \frac{1}{2}$.

3. We now consider the application of Preprocessing Rule 3. We know that $N(Z)$ was independent. In this case, $k' = k - |Z|$. We claim that $w(x') \geq w(x) - |Z|$. Suppose that this is not true. Then, it must be the case that $w(x') \leq w(x) - |Z| - \frac{1}{2}$.

We will now consider three cases depending on the value $x'(z)$ where $z$ is the vertex in $G'$ resulting from the identification of $N(Z)$.

**Case 1:** $x'(z) = 1$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex $v$ in $G' \setminus \{z\}$, retain the value assigned by $x'$, that is $x''(v) = x'(v)$. For every vertex in $N(Z)$, assign 1 and for every vertex in $Z$, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') - 1 + |N(Z)| = w(x') - 1 + (|Z| + 1) \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

**Case 2:** $x'(z) = 0$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex $v$ in $G' \setminus \{z\}$, retain the value assigned by $x'$, that is $x''(v) = x'(v)$. For every vertex in $Z$, assign 1 and for every vertex in $N(Z)$, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') + |Z| \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

**Case 3:** $x'(z) = \frac{1}{2}$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex $v$ in $G' \setminus \{z\}$, retain the value assigned by $x'$, that is $x''(v) = x'(v)$. For every vertex in $\{z\}$, retain the value assigned by $x'$, that is $x''(v) = x'(v)$. For every vertex in $N(Z)$, assign 1 and for every vertex in $Z$, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x) - |Z| \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.
For every vertex in $Z \cup N(Z)$, assign $\frac{1}{2}$. Clearly this is a feasible solution. But now,

$$w(x'') = w(x') - \frac{1}{2} + \frac{1}{2}(|Z| + |N(Z)|) = w(x') - \frac{1}{2} + \frac{1}{2}(|Z| + |Z| + 1) \leq w(x) - \frac{1}{2}.$$  

Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

Hence, $w(x') \geq w(x) - |Z|$, which implies that $\mu(G', k') \leq \mu(G, k)$.

4. We now consider the branching step.

(a) Consider the case when we pick $u$ in the vertex cover. In this case, $k' = k - 1$.

We claim that $w(x') \geq w(x) - \frac{1}{2}$. Suppose that this is not the case. Then, it must be the case that $w(x') \leq w(x) - 1$. Consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex $v \in V \setminus \{u\}$, $x''(v) = x'(v)$ and $x''(u) = 1$. Now, $x''$ is clearly a feasible solution for LPVC($G$) and has a value at most that of $x$. But this contradicts our assumption that $x \equiv \frac{1}{2}$ is the unique optimum solution to LPVC($G$). Hence, $w(x') \geq w(x) - \frac{1}{2}$, which implies that $\mu(G', k') \leq \mu(G, k) - \frac{1}{2}$.

(b) Consider the case when we don’t pick $u$ in the vertex cover. In this case, $k' = k - |N(u)|$. We know that $w(x') = w(x) - \frac{1}{2}(|\{u\}| + |N(u)|) + \frac{1}{2}(|V_1'| - |V_0'\cup \{u\}|).$ Adding and subtracting $\frac{1}{2}(|N(u)|)$, we get $w(x') = w(x) - |N(u)| - \frac{1}{2}(|\{u\}| - |N(u)|) + \frac{1}{2}(|V_1'| - |V_0'|)$. But, $\{u\} \cup V_0'$ is an independent set in $G$, and $N(\{u\} \cup V_0') = N(u) \cup V_1'$ in $G$. Since surplus($G$) $\geq 2,$ $|N(\{u\} \cup V_0')| - |\{u\} \cup V_0'| \geq 2$. Hence, $w(x') = w(x) - |N(u)| + \frac{1}{2}(|N(\{u\} \cup V_0')| - |\{u\} \cup V_0'|) \geq w(x) - |N(u)| + 1.$

Hence, $\mu(G', k') \leq \mu(G, k) - 1.$

We have thus shown that the preprocessing rules do not increase the measure $\mu = \mu(G, k)$ and the branching step results in a $(\frac{1}{2}, 1)$ branching vector, resulting in the re-
currence $T(\mu) \leq T(\mu - \frac{1}{2}) + T(\mu - 1)$ which solves to $(2.6181)^{\mu} = (2.6181)^{k-vc^*(G)}$. Thus, we get a $O^*(2.6181^{(k-vc^*(G))})$ algorithm for VERTEX COVER ABOVE LP.

**Theorem 6.3.8.** VERTEX COVER ABOVE LP can be solved in time $O^*(2.6181^{(k-vc^*(G))})$.

By applying the above theorem iteratively for increasing values of $k$, we can compute a minimum vertex cover of $G$ and hence we have the following corollary.

**Corollary 6.3.9.** There is an algorithm that, given a graph $G$, runs in time $O^*(2.6181^{(vc(G)-vc^*(G))})$ and computes a minimum vertex cover of $G$.

### 6.4 Improved Algorithm for VERTEX COVER ABOVE LP

In this section we give an improved algorithm for VERTEX COVER ABOVE LP using some more branching steps based on the structure of the neighborhood of the vertex (set) on which we branch. The goal is to achieve branching vectors better that $(\frac{1}{2}, 1)$.

#### 6.4.1 Some general claims to measure the drops

First, we capture the drop in the measure in the branching steps, including when we branch on a larger sized sets. In particular, when we branch on a set $S$ of vertices, in one branch we set all vertices of $S$ to 1, and in the other, we set all vertices of $S$ to 0. Note, however that such a branching on $S$ may not be exhaustive (as the branching doesn’t explore the possibility that some vertices of $S$ are set to 0 and some are set to 1) unless the set $S$ satisfies the premise of Lemma 6.3.3. Let $\mu = \mu(G, k)$ be the measure as defined in the previous section.

**Lemma 6.4.1.** Let $G$ be a graph with surplus$(G) = p$, and let $S$ be an independent set. Let $\mathcal{H}_S$ be the collection of all independent sets of $G$ that contain $S$ (including $S$). Then,
including \( S \) in the vertex cover while branching leads to a decrease of \( \min\{\frac{|S|}{2}, \frac{p}{2}\} \) in \( \mu \); and the branching excluding \( S \) from the vertex cover leads to a drop of \( \frac{\text{surplus}(G_S)}{2} \geq \frac{p}{2} \) in \( \mu \).

**Proof.** Let \((G', k')\) be the instance resulting from the branching, and let \( x' \) be an optimum solution to \( \text{LPVC}(G') \). Consider the case when we pick \( S \) in the vertex cover. In this case, \( k' = k - |S| \). We know that \( w(x') = w(x) - \frac{|S|}{2} + \frac{1}{2}(|V_1'| - |V_0'|) \). If \( V_0' = \emptyset \), then we know that \( V_1' = \emptyset \), and hence we have that \( w(x') = w(x) - \frac{|S|}{2} \). Else, by adding and subtracting \( \frac{1}{2}(|S|) \), we get \( w(x') = w(x) - |S| + \frac{|S|}{2} + \frac{1}{2}(|V_1'| - |V_0'|) \). However, \( N(V_0') \subseteq S \cup V_1' \) in \( G \). Thus, \( w(x') \geq w(x) - |S| + \frac{1}{2}(|N(V_0')| - |V_0'|) \). We also know that \( V_0' \) is an independent set in \( G \), and thus, \( |N(V_0')| - |V_0'| \geq \text{surplus}(G) = p \).

Hence, in the first case \( \mu(G', k') \leq \mu(G, k) - \frac{|S|}{2} \) and in the second case \( \mu(G', k') \leq \mu(G, k) - \frac{p}{2} \). Thus, the drop in the measure when \( S \) is included in the vertex cover is at least \( \min\{\frac{|S|}{2}, \frac{p}{2}\} \).

Consider the case when we don’t pick \( S \) in the vertex cover. In this case, \( k' = k - |N(S)| \). We know that \( w(x') = w(x) - \frac{1}{2}(|S| + |N(S)|) + \frac{1}{2}(|V_1'| - |V_0'|) \). Adding and subtracting \( \frac{1}{2}(|N(S)|) \), we get \( w(x') = w(x) - |N(S)| + \frac{1}{2}(|N(S)| - |S|) + \frac{1}{2}(|V_1'| - |V_0'|) \). But, \( S \cup V_0' \) is an independent set in \( G \), and \( N(S \cup V_0') = N(S) \cup V_1' \) in \( G \). Thus, \( |N(S \cup V_0')| - |S \cup V_0'| \geq \text{surplus}(G_S) \). Hence, \( w(x') = w(x) - |N(S)| + \frac{1}{2}(|N(S \cup V_0')| - |S \cup V_0'|) \geq w(x) - |N(S)| + \frac{\text{surplus}(G_S)}{2} \). Hence, \( \mu(G', k') \leq \mu(G, k) - \frac{\text{surplus}(G_S)}{2} \).

Thus, after the preprocessing steps (when the surplus of the graph is at least 2), suppose we manage to find (in polynomial time) a set \( S \subseteq V \) such that

- \( \text{surplus}(G) = \text{surplus}(S) = \text{surplus}(G_S) \),
- \( |S| \geq 2 \), and
that the branching that sets all of $S$ to 0 or all of $S$ to 1 is exhaustive.

Then, Lemma 6.4.1 guarantees that branching on this set right away leads to a $(1, 1)$ branching vector. We now explore the cases in which such sets do exist. Note that the first condition above implies the third from the Lemma 6.3.3. First, we show that if there exists a set $S$ such that $|S| \geq 2$ and \text{surplus}(G) = \text{surplus}(S)$, then we can find such a set in polynomial time.

**Lemma 6.4.2.** Let $G$ be a graph on which Preprocessing Rule 1 does not apply (i.e. all $\frac{1}{2}$ is the unique solution to LPVC($G$)). If $G$ has an independent set $S'$ such that $|S'| \geq 2$ and \text{surplus}(S') = \text{surplus}(G)$, then in time $O(mn^2 \sqrt{n})$ we can find an independent set $S$ such that $|S| \geq 2$ and \text{surplus}(S) = \text{surplus}(G)$.

**Proof.** By our assumption we know that $G$ has an independent set $S'$ such that $|S'| \geq 2$ and \text{surplus}(S') = \text{surplus}(G)$. Let $u, v \in S'$. Let $\mathcal{H}$ be the collection of all independent sets of $G$ containing $u$ and $v$. Let $x$ be an optimal solution to LPVC($G$) obtained after setting $x(u) = 0$ and $x(v) = 0$. Take $S = V^x_0$, clearly, we have that $\{u, v\} \subseteq V^x_0$. We now have the following claim.

**Claim 3.** $\text{surplus}(S) = \text{surplus}(G)$.

**Proof.** We know that the objective value of LPVC($G$) after setting $x(u) = x(v) = 0$, $w(x) = |V|/2 + (|N(S)| - |S|)/2 = |V|/2 + \text{surplus}(S)/2$, as all $\frac{1}{2}$ is the unique solution to LPVC($G$).

Another solution $x'$, for LPVC($G$) that sets $u$ and $v$ to 0, is obtained by setting $x'(a) = 0$ for every $a \in S'$, $x'(a) = 1$ for every $a \in N(S')$ and by setting all other variables to $1/2$. It is easy to see that such a solution is a feasible solution of the required kind and $w(x') = |V|/2 + (|N(S')| - |S'|)/2 = |V|/2 + \text{surplus}(S')/2$. However, as $x$ is also an optimum solution, $w(x) = w(x')$, and hence we have that $\text{surplus}(S) \leq$
surplus(S′). But as S′ is a set of minimum surplus in G, we have that surplus(S) = surplus(S′) = surplus(G) proving the claim.

Thus, we can find a such a set S in polynomial time by solving LPVC(G) after setting x(u) = 0 and x(v) = 0 for every pair of vertices u, v such that (u, v) \notin E and picking that set V_0υ which has the minimum surplus among all x′s among all pairs u, v. Since any V_0υ contains at least 2 vertices, we have that |S| ≥ 2. The bound on the time required to find this set follows from Lemma 6.3.1.

6.4.2 (1,1) drops in the measure

Lemma 6.4.1 and Lemma 6.4.2 together imply that, if there is a minimum surplus set of size at least 2 in the graph, then we can find and branch on that set to get a (1, 1) drop in the measure.

Suppose that there is no minimum surplus set of size more than 1. Note that, by Lemma 6.4.1, when surplus(G) ≥ 2, we get a drop of (surplus(G))/2 ≥ 1 in the branch where we exclude a vertex or a set. Hence, if we find a vertex (set) to exclude in either branch of a two way branching, we get a (1, 1) branching vector. We now identify another such case.

Lemma 6.4.3. Let v be a vertex such that G[N(v) \ {u}] is a clique for some neighbor u of v. Then, there exists a minimum vertex cover that excludes either v or u.

Proof. Towards the proof we first show the following well known observation.

Claim 4. Let G be a graph and v be a vertex. Then there exists a minimum vertex cover for G containing N(v) or at most |N(v)| − 2 vertices from N(v).

Proof. If a minimum vertex cover of G, say C, contains exactly |N(v)| − 1 vertices of N(v), then we know that C must contain v. Observe that C′ = C \ {v} ∪ N(v)
is also a vertex cover of $G$ of the same size as $C$. However, in this case, we have a minimum vertex cover containing $N(v)$. Thus, there exists a minimum vertex cover of $G$ containing $N(v)$ or at most $|N(v)| − 2$ vertices from $N(v)$. □

Let $v$ be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique. Consider a minimum vertex cover and suppose that $v$ is in the vertex cover. Clearly, $N(v)$ is not contained in this vertex cover. Since $G[N(v) \setminus \{u\}]$ is a clique this vertex cover contains at least $|N(v)| − 2$ vertices from $G[N(v) \setminus \{u\}]$. Hence, by Claim 4, the vertex $v$ is not part of the vertex cover. This completes the proof. □

Next, in order to identify another case where we might obtain a $(1, 1)$ branching vector, we first observe and capture the fact that when Preprocessing Rule 2 is applied, the measure $k − ve^*(G)$ actually drops by at least $\frac{1}{2}$ (as proved in item 2 of the analysis of the simple algorithm in Section 6.3.3).

**Lemma 6.4.4.** Let $(G, k)$ be the input instance and $(G', k')$ be the instance obtained after applying Preprocessing Rule 2. Then, $\mu(G', k') \leq \mu(G, k) − \frac{1}{2}$.

Thus, after we branch on an arbitrary vertex, if we are able to apply Preprocessing Rule 2 in the branch where we include that vertex, we get a $(1, 1)$ drop. This is because, in the branch where we exclude the vertex, we get a drop of 1 by Lemma 6.4.1, and in the branch where we include the vertex, we get a drop of $\frac{1}{2}$ by Lemma 6.4.1, which is then followed by a drop of $\frac{1}{2}$ due to Lemma 6.4.4.

Thus, after preprocessing, the algorithm performs the following steps (see Figure 6.2) each of which results in a $(1, 1)$ drop as argued before. Note that Preprocessing Rule 1 cannot apply in the graph $G \setminus \{v\}$ since the surplus of $G$ can drop by at most 1 by deleting a vertex. Hence, checking if rule B3 applies is equivalent to checking if, for some vertex $v$, Preprocessing Rule 2 applies in the graph $G \setminus \{v\}$. Recall that, by Lemma 6.3.5
we can check this in polynomial time and hence we can check if $B_3$ applies on the graph in polynomial time.

**Branching Rules.**

These branching rules are applied in this order.

**B 1.** Apply Lemma 6.4.2 to test if there is a set $S$ such that $\text{surplus}(S) = \text{surplus}(G)$ and $|S| \geq 2$. If so, then branch on $S$.

**B 2.** Let $v$ be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique for some vertex $u$ in $N(v)$. Then in one branch add $N(v)$ into the vertex cover, decrease $k$ by $|N(v)|$, and delete $N[v]$ from the graph. In the other branch add $N(u)$ into the vertex cover, decrease $k$ by $|N(u)|$, and delete $N[u]$ from the graph.

**B 3.** Apply Lemma 6.3.5 to test if there is a vertex $v$ such that Preprocessing Rule 2 applies in $G \setminus \{v\}$. If there is such a vertex, then branch on $v$.

Figure 6.2: Outline of the branching steps yielding $(1, 1)$ drop.

### 6.4.3 A Branching step yielding $(1/2, 3/2)$ drop

Now, suppose none of the preprocessing and branching rules presented thus far apply. Let $v$ be a vertex with degree at least 4. Let $S = \{v\}$ and recall that $\mathcal{H}_S$ is the collection of all independent sets containing $S$, and $\text{surplus}(\mathcal{H}_S)$ is the surplus of an independent set with minimum surplus in $\mathcal{H}_S$. We claim that $\text{surplus}(\mathcal{H}_S) \geq 3$.

As the preprocessing rules don’t apply, clearly $\text{surplus}(\mathcal{H}_S) \geq \text{surplus}(G) \geq 2$. If $\text{surplus}(\mathcal{H}_S) = 2$, then the set that realizes $\text{surplus}(\mathcal{H}_S)$ is not $S$ (as the $\text{surplus}(S) = \text{degree}(v) - 1 = 3$), but a superset of $S$, which is of cardinality at least 2. Then, the Branching Rule B1 would have applied which is a contradiction. This proves the claim. Hence, by Lemma 6.4.1, we get a drop of at least $3/2$ in the branch that excludes the vertex $v$ resulting in a $(1/2, 3/2)$ drop. This branching step is illustrated in Figure 6.3.
4. If there exists a vertex $v$ of degree at least 4 then branch on $v$.

Figure 6.3: The branching step yielding a $(1/2, 3/2)$ drop.

6.4.4 A Branching step yielding $(1, 3/2, 3/2)$ drop

Next, we observe that when branching on a vertex, if in the branch that includes the vertex in the vertex cover (which guarantees a drop of $1/2$), any of the Branching Rules B1 or B2 or B3 applies, then combining the subsequent branching with this branch of the current branching step results in a net drop of $(1, 3/2, 3/2)$ (which is $(1, 1/2+1, 1/2+1)$) (see Figure 6.10 (a)). Thus, we add the following branching rule to the algorithm (Figure 6.5).

5. Let $v$ be a vertex. If B1 applies in $\mathcal{R}(G \setminus \{v\})$ or there exists a vertex $w$ in $\mathcal{R}(G \setminus \{v\})$ on which either B2 or B3 applies then branch on $v$.

Figure 6.5: The branching step yielding a $(1, 3/2, 3/2)$ drop.
6.4.5 The Final branching step

Finally, if the preprocessing and branching rules presented thus far do not apply, then note that we are left with a 3-regular graph. In this case, we simply pick a vertex \( v \) and branch. However, we execute the branching step carefully in order to simplify the analysis of the drop. More precisely, we execute the following step at the end.

**B 6.** Pick an arbitrary degree 3 vertex \( v \) in \( G \) and let \( x, y \) and \( z \) be the neighbors of \( v \). Then in one branch add \( v \) into the vertex cover, decrease \( k \) by 1, and delete \( v \) from the graph. The other branch that excludes \( v \) from the vertex cover, is performed as follows. Delete \( x \) from the graph, decrease \( k \) by 1, and obtain \( \mathcal{R}(G \setminus \{x\}) \). During the process of obtaining \( \mathcal{R}(G \setminus \{x\}) \), Preprocessing Rule 3 would have been applied on vertices \( y \) and \( z \) to obtain a ‘merged’ vertex \( v_{yz} \) (see proof of correctness of this rule). Now delete \( v_{yz} \) from the graph \( \mathcal{R}(G \setminus \{x\}) \), and decrease \( k \) by 1.

Figure 6.6: Outline of the last step.

6.4.6 Complete Algorithm and Correctness

A detailed outline of the algorithm is given in Figure 6.7. Note that we have already argued the correctness and analyzed the drops of all steps except the last step, B6.

The correctness of this branching rule will follow from the fact that \( \mathcal{R}(G \setminus \{x\}) \) is obtained by applying Preprocessing Rule 3 alone and that too only on the neighbors of \( x \), that is, on the degree 2 vertices of \( G \setminus \{x\} \) (Lemma 6.4.10). Lemma 6.4.15 (to appear later) shows the correctness of deleting \( v_{yz} \) from the graph \( \mathcal{R}(G \setminus \{x\}) \) without branching. Thus, the correctness of this algorithm follows from the soundness of the preprocessing rules and the fact that the branching is exhaustive.

The running time will be dominated by the way B6 and the subsequent branching
**Preprocessing Step.** Apply Preprocessing Rules 1, 2 and 3 in this order exhaustively on $G$.

**Connected Components.** Apply the algorithm on connected components of $G$ separately. Furthermore, if a connected component has size at most 10, then solve the problem optimally in $O(1)$ time.

**Branching Rules.**
These branching rules are applied in this order.

**B1** If there is a set $S$ such that $\text{surplus}(S) = \text{surplus}(G)$ and $|S| \geq 2$, then branch on $S$.

**B2** Let $v$ be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique for some vertex $u$ in $N(v)$. Then in one branch add $N(v)$ into the vertex cover, decrease $k$ by $|N(v)|$, and delete $N[v]$ from the graph. In the other branch add $N(u)$ into the vertex cover, decrease $k$ by $|N(u)|$, and delete $N[u]$ from the graph.

**B3** Let $v$ be a vertex. If Preprocessing Rule 2 can be applied to obtain $R(G \setminus \{v\})$ from $G \setminus \{v\}$, then branch on $v$.

**B4** If there exists a vertex $v$ of degree at least 4 then branch on $v$.

**B5** Let $v$ be a vertex. If **B1** applies in $R(G \setminus \{v\})$ or if there exists a vertex $w$ in $R(G \setminus \{v\})$ on which $\textbf{B2}$ or $\textbf{B3}$ applies then branch on $v$.

**B6** Pick an arbitrary degree 3 vertex $v$ in $G$ and let $x$, $y$ and $z$ be the neighbors of $v$. Then in one branch add $v$ into the vertex cover, decrease $k$ by 1, and delete $v$ from the graph. The other branch, the one that excludes $v$ from the vertex cover, is performed as follows. Delete $x$ from the graph, decrease $k$ by 1, and obtain $R(G \setminus \{x\})$. Now, delete $v_{yz}$ from the graph $R(G \setminus \{x\})$, the vertex that has been created by the application of Preprocessing Rule 3 on $v$ while obtaining $R(G \setminus \{x\})$ and decrease $k$ by 1.

Figure 6.7: Outline of the Complete algorithm.

apply. We will see that **B6** is our most expensive branching rule. In fact, this step dominates the running time of the algorithm of $O^*(2.3146^{\mu(G,k)})$ due to a branching vector of $(3/2, 3/2, 5/2, 5/2, 2)$. We will argue that when we apply **B6** on a vertex, say $v$, then on either side of the branch we will be able to branch using rules **B1**, or **B2**, or
### Rule and Branching Vector

<table>
<thead>
<tr>
<th>Rule</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>B5</th>
<th>B6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Vector</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>($\frac{1}{2}, \frac{3}{2}$)</td>
<td>($\frac{3}{2}, \frac{1}{2}$)</td>
<td>($\frac{3}{2}, \frac{1}{2}$, 1)</td>
</tr>
</tbody>
</table>

| Running time | $2^\mu$ | $2^\mu$ | $2^\mu$ | 2.1479$^\mu$ | 2.3146$^\mu$ | 2.3146$^\mu$ |

Figure 6.8: A table giving the decrease in the measure due to each branching rule.

**B3** or **B4.** More precisely, we show that in the branch where we include $v$ in the vertex cover,

- there is a vertex of degree 4 in $\mathcal{R}(G \setminus \{v\})$. Thus, **B4** will apply on the graph $\mathcal{R}(G \setminus \{v\})$ (if any of the earlier branching rules applied in this graph, then rule **B5** would have applied on $G$).

- $\mathcal{R}(G \setminus \{v\})$ has a degree 4 vertex $w$ such that there is a vertex of degree 4 in the graph $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w\})$ and thus one of the Branching Rules **B1, B2, B3** or **B4** applies on the graph $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w\})$.

Similarly, in the branch where we exclude the vertex $v$ from the solution (and add the vertices $x$ and $v_{yz}$ into the vertex cover), we will show that a degree 4 vertex remains in the reduced graph. This yields the claimed branching vector (see Figure 6.4.6). The rest of the section is geared towards showing this.

We start with the following definition.

**Definition 6.4.5.** We say that a graph $G$ is irreducible if Preprocessing Rules 1, 2 and 3 and the Branching Rules **B1, B2, B3, B4** and **B5** do not apply on $G$.

Observe that when we apply **B6**, the current graph is 3-regular. Thus, after we delete a vertex $v$ from the graph $G$ and apply Preprocessing Rule 3 we will get a degree 4 vertex. Our goal is to identify conditions that ensure that the degree 4 vertices we obtain by...
applying Preprocessing Rule 3 survive in the graph \( R(G \setminus \{v\}) \). We prove the existence of degree 4 vertices in subsequent branches after applying B6 as follows.

- We do a closer study of the way Preprocessing Rules 1, 2 and 3 apply on \( G \setminus \{v\} \) if Preprocessing Rules 1, 2 and 3 and the Branching Rules B1, B2 and B3 do not apply on \( G \). Based on our observations, we prove some structural properties of the graph \( R(G \setminus \{v\}) \), This is achieved by Lemma 6.4.10.

- Next, we show that Lemma 6.4.10, along with the fact that the graph is irreducible implies a lower bound of 7 on the length of the shortest cycle in the graph (Lemma 6.4.13). This lemma allows us to argue that when the preprocessing rules are applied, their effect is local.

- Finally, Lemmas 6.4.10 and 6.4.13 together ensure the presence of the required number of degree 4 vertices in the subsequent branching (Lemma 6.4.14).

**Main Structural Lemmas: Lemmas 6.4.10 and 6.4.13**

We start with some simple well known observations that we use repeatedly in this section. These observations follow from results in [84]. We give proofs for completeness.

**Lemma 6.4.6.** Let \( G \) be an undirected graph, then the following are equivalent.

1. Preprocessing Rule 1 applies (i.e. All \( \frac{1}{2} \) is not the unique solution to the LPVC(G).)
2. There exists an independent set \( I \) of \( G \) such that surplus(\( I \)) \( \leq 0 \).
3. There exists an optimal solution \( x \) to LPVC(G) that assigns 0 to some vertex.

**Proof.** (1) \( \implies \) (3): As we know that the optimum solution is half-integral, there exists an optimum solution that assigns 0 or 1 to some vertex. Suppose no vertex is assigned 0. Then, for any vertex which is assigned 1, its value can be reduced to \( \frac{1}{2} \) maintaining
feasibility (as all its neighbors have been assigned value $\geq \frac{1}{2}$) which is a contradiction to the optimality of the given solution.

(3) $\implies$ (2): Let $I = V_0^x$, and suppose that $\text{surplus}(I) > 0$. Then consider the solution $x'$ that assigns $\frac{1}{2}$ to vertices in $I \cup N(I)$, retaining the value of $x$ for the other vertices. Then $x'$ is a feasible solution whose objective value $w(x')$ drops from $w(x)$ by $(|N(I)| - |I|)/2 = \text{surplus}(I)/2 > 0$ which is a contradiction to the optimality of $x$.

(2) $\implies$ (1): Setting all vertices in $I$ to 0, all vertices in $N(I)$ to 1 and setting the remaining vertices to $\frac{1}{2}$ gives a feasible solution whose objective value is at most $|V|/2$, and hence all $\frac{1}{2}$ is not the unique solution to LPVC($G$).

**Lemma 6.4.7.** Let $G$ be an undirected graph, then the following are equivalent.

(1) Preprocessing Rule 1 or 2 or 3 applies.

(2) There exists an independent set $I$ such that $\text{surplus}(I) \leq 1$.

(3) There exists a vertex $v$ such that an optimal solution $x$ to LPVC($G \setminus \{v\}$) assigns 0 to some vertex.

**Proof.** The fact that (1) and (2) are equivalent follows from the definition of the preprocessing rules and Lemma 6.4.6.

(3) $\implies$ (2). By Lemma 6.4.6, there exists an independent set $I$ in $G \setminus \{v\}$ whose surplus is at most 0. The same set will have surplus at most 1 in $G$.

(2) $\implies$ (3). Let $v \in N(I)$. Then $I$ is an independent set in $G \setminus \{v\}$ with surplus at most 0, and hence by Lemma 6.4.6, there exists an optimal solution to LPVC($G \setminus \{v\}$) that assigns 0 to some vertex.

We now prove an auxiliary lemma about the application of Preprocessing Rule 3 which will be useful in simplifying later proofs.
Lemma 6.4.8. Let $G$ be a graph and $G_R$ be the graph obtained from $G$ by applying Preprocessing Rule 3 on an independent set $Z$. Let $z$ denote the newly added vertex corresponding to $Z$ in $G_R$.

1. If $G_R$ has an independent set $I$ such that $\text{surplus}(I) = p$, then $G$ also has an independent set $I'$ such that $\text{surplus}(I') = p$ and $|I'| \geq |I|$.

2. Furthermore, if $z \in I \cup N(I)$ then $|I'| > |I|$.

Proof. Let $Z$ denote the minimum surplus independent set on which Preprocessing Rule 3 has been applied and $z$ denote the newly added vertex. Observe that since Preprocessing Rule 3 applies on $Z$, we have that $Z$ and $N(Z)$ are independent sets, $|N(Z)| = |Z| + 1$ and $|N(Z)| \geq 2$.

Let $I$ be an independent set of $G_R$ such that $\text{surplus}(I) = p$.

- If both $I$ and $N(I)$ do not contain $z$ then we have that $G$ has an independent set $I$ such that $\text{surplus}(I) = p$.

- Suppose $z \in I$. Then consider the following set: $I' := I \setminus \{z\} \cup N(Z)$. Notice that $z$ represents $N(Z)$ and thus $I$ do not have any neighbors of $N(Z)$. This implies that $I'$ is an independent set in $G$. Now we will show that $\text{surplus}(I') = p$. We know that $|N(Z)| = |Z| + 1$ and $N(I') = N(I) \cup Z$. Thus,

$$|N(I')| - |I'| = (|N(I)| + |Z|) - |I'|$$

$$= (|N(I)| + |Z|) - (|I| - 1 + |N(Z)|)$$

$$= (|N(I)| + |Z|) - (|I| + |Z|)$$

$$= |N(I)| - |I| = \text{surplus}(I) = p.$$

104
• Suppose \( z \in N(I) \). Then consider the following set: \( I' := I \cup Z \). Notice that \( z \) represents \( N(Z) \) and since \( z \notin I \) we have that \( I \) do not have any neighbors of \( Z \). This implies that \( I' \) is an independent set in \( G \). We show that \( \text{surplus}(I') = p \).

We know that \( |N(Z)| = |Z| + 1 \). Thus,

\[
|N(I')| - |I'| = (|N(I)| - 1 + |N(Z)|) - |I'|
\]
\[
= (|N(I)| - 1 + |N(Z)|) - (|I| + |Z|)
\]
\[
= (|N(I)| + |Z|) - (|I| + |Z|)
\]
\[
= |N(I)| - |I| = \text{surplus}(I) = p.
\]

From the construction of \( I' \), it is clear that \( |I'| \geq |I| \) and if \( z \in (I \cup N(I)) \) then \( |I'| > |I| \). This completes the proof.

We now give some definitions that will be useful in formulating the statement of the main structural lemma.

**Definition 6.4.9.** Let \( G \) be a graph and \( \mathcal{P} = P_1, P_2, \ldots, P_\ell \) be a sequence of exhaustive applications of Preprocessing Rules 1, 2 and 3 applied in this order on \( G \) to obtain \( G' \). Let \( \mathcal{P}_3 = P_a, P_b, \ldots, P_t \) be the subsequence of \( \mathcal{P} \) restricted to Preprocessing Rule 3. Furthermore let \( Z_j, \ j \in \{a, \ldots, t\} \) denote the minimum surplus independent set corresponding to \( P_i \) on which the Preprocessing Rule 3 has been applied and \( z_j \) denote the newly added vertex (See Lemma 6.3.4). Let \( Z^* = \{z_j \mid j \in \{a, \ldots, t\}\} \) be the set of these newly added vertices.

• We say that an application of Preprocessing Rule 3 is trivial if the minimum surplus independent set \( Z_j \) on which \( P_j \) is applied has size 1, that is, \( |Z_j| = 1 \).
• We say that all applications of Preprocessing Rule 3 are independent if for all \( j \in \{a, \ldots, t\} \), \( N[Z_j] \cap Z^* = \emptyset \).

Essentially, independent applications of Preprocessing Rule 3 mean that the set on which the rule is applied, as well as all its neighbors are vertices in the original graph.

Next, we state and prove one of the main structural lemmas of this section.

**Lemma 6.4.10.** Let \( G = (V, E) \) be a graph on which Preprocessing Rules 1, 2 and 3 and the Branching Rules \( B_1, B_2 \) and \( B_3 \) do not apply. Then for any vertex \( v \in V \),

1. Preprocessing Rules 1 and 2 have not been applied while obtaining \( R(G \setminus \{v\}) \) from \( G \setminus \{v\} \);

2. and all applications of the Preprocessing Rule 3 while obtaining \( R(G \setminus \{v\}) \) from \( G \setminus \{v\} \) are independent and trivial.

**Proof.** Fix a vertex \( v \). Let \( G_0 = G \setminus \{v\}, G_1, \ldots, G_t = R(G \setminus \{v\}) \) be a sequence of graphs obtained by applying Preprocessing Rules 1, 2 and 3 in this order to obtain the reduced graph \( R(G \setminus \{v\}) \).

We first observe that Preprocessing Rule 2 never applies in obtaining \( R(G \setminus \{v\}) \) from \( G \setminus \{v\} \) since otherwise, \( B_3 \) would have applied on \( G \). Next, we show that Preprocessing Rule 1 does not apply. Let \( q \) be the least integer such that Preprocessing Rule 1 applies on \( G_q \) and it does not apply to any graph \( G_{q'}, q' < q \). Suppose that \( q \geq 1 \). Then, only Preprocessing Rule 3 has been applied on \( G_0, \ldots, G_{q-1} \). This implies that \( G_q \) has an independent set \( I_q \) such that \( \text{surplus}(I_q) \leq 0 \). Then, by Lemma 6.4.8, \( G_{q-1} \) also has an independent set \( I'_q \) such that \( \text{surplus}(I'_q) \leq 0 \) and thus by Lemma 6.4.6 Preprocessing Rule 1 applies to \( G_{q-1} \). This contradicts the assumption that on \( G_{q-1} \) Preprocessing Rule 1 does not apply. Thus, we conclude that \( q \) must be zero. So, \( G \setminus \{v\} \) has an independent set \( I_0 \) such that \( \text{surplus}(I_0) \leq 0 \) in \( G \setminus \{v\} \) and thus \( I_0 \) is an independent...
set in $G$ such that $\text{surplus}(I_0) \leq 1$ in $G$. By Lemma 6.4.7 this implies that either of Preprocessing Rules 1, 2 or 3 is applicable on $G$, a contradiction to the given assumption.

Now we show the second part of the lemma. By the first part we know that the $G_i$’s have been obtained by applications of Preprocessing Rule 3 alone. Let $Z_i, 0 \leq i \leq t - 1$ be the sets in $G_i$ on which Preprocessing Rule 3 has been applied. Let the newly added vertex corresponding to $N(Z_i)$ in this process be $z'_i$. We now make the following claim.

**Claim 5.** For any $i \geq 0$, if $G_i$ has an independent set $I_i$ such that $\text{surplus}(I_i) = 1$, then $G$ has an independent set $I$ such that $|I| \geq |I_i|$ and $\text{surplus}(I) = 2$. Furthermore, if $(I_i \cup N(I_i)) \cap \{z_1, \ldots, z_{i-1}\} \neq \emptyset$, then $|I| > |I_i|$.

**Proof.** We prove the claim by induction on the length of the sequence of graphs. For the base case consider $q = 0$. Since Preprocessing Rules 1, 2, and 3 do not apply on $G$, we have that $\text{surplus}(G) \geq 2$. Since $I_0$ is an independent set in $G \setminus \{v\}$ we have that $I_0$ is an independent set in $G$ also. Furthermore since $\text{surplus}(I_0) = 1$ in $G \setminus \{v\}$, we have that $\text{surplus}(I_0) = 2$ in $G$, as $\text{surplus}(G) \geq 2$. This implies that $G$ has an independent set $I_0$ with $\text{surplus}(I_0) = 2 = \text{surplus}(G)$. Furthermore, since $G_0$ does not have any newly introduced vertices, the last assertion is vacuously true. Now let $q \geq 1$. Suppose that $G_q$ has a set $|I_q|$ and $\text{surplus}(I_q) = 1$. Thus, by Lemma 6.4.8, $G_{q-1}$ also has an independent set $I'_q$ such that $|I'_q| \geq |I_q|$ and $\text{surplus}(I'_q) = 1$. Now by the induction hypothesis, $G$ has an independent set $I$ such that $|I| \geq |I'_q| \geq |I_q|$ and $\text{surplus}(I) = 2 = \text{surplus}(G)$.

Next we consider the case when $(I_q \cup N(I_q)) \cap \{z'_1, \ldots, z'_{q-1}\} \neq \emptyset$. If $z'_{q-1} \notin I_q \cup N(I_q)$ then we have that $I_q$ is an independent set in $G_{q-1}$ such that $(I_q \cup N(I_q)) \cap \{z'_1, \ldots, z'_{q-2}\} \neq \emptyset$. Thus, by the induction hypothesis, we have that $G$ has an independent set $I$ such that $|I| > |I_q|$ and $\text{surplus}(I) = 2 = \text{surplus}(G)$. On the other hand, if $z'_{q-1} \in I_q \cup N(I_q)$ then by Lemma 6.4.8, we know that $G_{q-1}$ has an
independent set $I'_q$ such that $|I'_q| > |I_q|$ and $\text{surplus}(I'_q) = 1$. Now by induction hypothesis we know that $G$ has an independent set $I$ such that $|I| \geq |I'_q| > |I_q|$ and $\text{surplus}(I) = 2 = \text{surplus}(G)$. This concludes the proof of the claim.

We first show that all the applications of Preprocessing Rule 3 are trivial. Claim 5 implies that if we have a non-trivial application of Preprocessing Rule 3 then it implies that $G$ has an independent set $I$ such that $|I| \geq 2$ and $\text{surplus}(I) = 2 = \text{surplus}(G)$. Then, $B_1$ would apply on $G$, a contradiction.

Finally, we show that all the applications of Preprocessing Rule 3 are independent. Let $q$ be the least integer such that the application of Preprocessing Rule 3 on $G_q$ is not independent. That is, the application of Preprocessing Rule 3 on $G_{q'}$, $q' < q$, is trivial and independent. Observe that $q \geq 1$. We already know that every application of Preprocessing Rule 3 is trivial. This implies that the set $Z_q$ contains a single vertex. Let $Z_q = \{z_q\}$. Since the application of Preprocessing Rule 3 on $Z_q$ is not independent we have that $(Z_q \cup N(Z_q)) \cap \{z'_1, \ldots, z'_{q-1}\} \neq \emptyset$. We also know that $\text{surplus}(Z_q) = 1$ and thus by Claim 5 we have that $G$ has an independent set $I$ such that $|I| \geq 2 > |Z_q|$ and $\text{surplus}(I) = 2 = \text{surplus}(G)$. This implies that $B_1$ would apply on $G$, a contradiction. Hence, we conclude that all the applications of Preprocessing Rule 3 are independent. This proves the lemma.

Let $g(G)$ denote the girth of the graph, that is, the length of the smallest cycle in $G$. Our next goal of this section is to obtain a lower bound on the girth of an irreducible graph. Towards this, we first introduce the notion of an untouched vertex.

**Definition 6.4.11.** We say that a vertex $v$ is untouched by an application of Preprocessing Rule 2 or Preprocessing Rule 3, if $\{v\} \cap (Z \cup N(Z)) = \emptyset$, where $Z$ is the set on which the rule is applied.

Let $g(G)$ denote the girth of the graph, that is, the length of the smallest cycle in $G$. Our next goal of this section is to obtain a lower bound on the girth of an irreducible graph. Towards this, we first introduce the notion of an untouched vertex.

**Definition 6.4.11.** We say that a vertex $v$ is untouched by an application of Preprocessing Rule 2 or Preprocessing Rule 3, if $\{v\} \cap (Z \cup N(Z)) = \emptyset$, where $Z$ is the set on which the rule is applied.
We now prove an auxiliary lemma regarding the application of the preprocessing rules on graphs of a certain girth and following that, we will prove a lower bound on the girth of irreducible graphs.

**Lemma 6.4.12.** Let $G$ be a graph on which Preprocessing Rules 1, 2 and 3 and the Branching Rules $B1$, $B2$, $B3$ do not apply and suppose that $g(G) \geq 5$. Then for any vertex $v \in V$, any vertex $x \notin N_2[v]$ is untouched by the preprocessing rules applied to obtain the graph $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$ and has the same degree as it does in $G$.

**Proof.** Since the preprocessing rules do not apply in $G$, the minimum degree of $G$ is at least 3 and since the graph $G$ does not have cycles of length 3 or 4, for any vertex $v$, the neighbors of $v$ are independent and there are no edges between vertices in the first and second neighborhood of $v$.

We know by Lemma 6.4.10 that only Preprocessing Rule 3 applies on the graph $G \setminus \{v\}$ and it applies only in a trivial and independent way. Let $U = \{u_1, \ldots, u_t\}$ be the degree 3 neighbors of $v$ in $G$ and let $D$ represent the set of the remaining (high degree) neighbors of $v$. Let $P_1, \ldots, P_l$ be the sequence of applications of Preprocessing Rule 3 on the graph $G \setminus \{v\}$, let $Z_i$ be the minimum surplus set corresponding to the application of $P_i$ and let $z_i$ be the new vertex created during the application of $P_i$.

We prove by induction on $i$, that

- the application $P_i$ corresponds to a vertex $u_j \in U$,
- any vertex $x \notin N_2[v] \setminus D$ is untouched by this application, and
- after the application of $P_i$, the degree of $x \notin N_2[v]$ in the resulting graph is the same as that in $G$.

In the base case, $i = 1$. Clearly, the only vertices of degree 2 in the graph $G \setminus \{v\}$ are the degree 3 neighbors of $v$. Hence, the application $P_1$ corresponds to some $u_j \in U$. 

109
Since the graph $G$ has girth at least 5, no vertex in $D$ can lie in the set $\{u_j\} \cup N(u_j)$ and hence must be untouched by the application of $P_1$. Since $u_j$ is a neighbor of $v$, it is clear that the application of $P_1$ leaves any vertex disjoint from $N_2[v]$ untouched. Now, suppose that after the application of $P_1$, a vertex $w$ disjoint from $N_2[v] \setminus D$ has lost a degree. Then, it must be the case that the application of $P_1$ identified two of $w$’s neighbors, say $w_1$ and $w_2$ as the vertex $z_1$. But since $P_1$ is applied on the vertex $u_j$, this implies the existence of a 4 cycle $u_j, w_1, w, w_2$ in $G$, which is a contradiction.

We assume as the induction hypothesis that the claim holds for all $i'$ such that $1 \leq i' < i$ for some $i > 1$. Now, consider the application of $P_i$. By Lemma 6.4.10, this application cannot be on any of the vertices created by the application of $P_l$ ($l < i$), and by the induction hypothesis, after the application of $P_{i-1}$, any vertex disjoint from $N_2[v] \setminus D$ remains untouched and retains the degree (which is $\geq 3$) it had in the original graph. Hence, the application of $P_i$ must occur on some vertex $u_j \in U$. Now, suppose that a vertex $w$ disjoint from $N_2[v] \setminus D$ has lost a degree. Then, it must be the case that $P_i$ identified two of $w$’s neighbors say $w_1$ and $w_2$ as the vertex $z_i$. Since $P_i$ is applied on the vertex $u_j$, this implies the existence of a 4 cycle $u_j, w_1, w, w_2$ in $G$, which is a contradiction. Finally, after the application of $P_i$, since no vertex outside $N_2[v] \setminus D$ has ever lost degree and they all had degree at least 3 to begin with, we cannot apply Preprocessing Rule 3 any further. This completes the proof of the claim.

Hence, after applying Preprocessing Rule 3 exhaustively on $G \setminus \{v\}$, any vertex disjoint from $N_2[v]$ is untouched and has the same degree as in the graph $G$. This completes the proof of the lemma. \qed

Recall that the graph is irreducible if none of the preprocessing rules or Branching Rules $B_1$ through $B_5$ apply, i.e: the algorithm has reached $B_6$. 

110
Lemma 6.4.13. Let $G$ be a connected $3$-regular irreducible graph with at least 11 vertices. Then, $g(G) \geq 7$.

Proof.  
1. Suppose $G$ contains a triangle $v_1, v_2, v_3$. Let $v_4$ be the remaining neighbor of $v_1$. Now, $G[N(v_1) \setminus \{v_4\}]$ is a clique, which implies that Branching Rule B2 applies and hence contradicts the irreducibility of $G$. Hence, $g(G) \geq 4$.

2. Suppose $G$ contains a cycle $v_1, v_2, v_3, v_4$ of length 4. Since $G$ does not contain triangles, it must be the case that $v_1$ and $v_3$ are independent. Recall that $G$ has minimum surplus 2, and hence surplus of the set $\{v_1, v_3\}$ is at least 2. Since $v_2$ and $v_4$ account for two neighbors of both $v_1$ and $v_3$, the neighborhood of $\{v_1, v_3\}$ can contain at most 2 more vertices ($G$ is 3 regular). Since the minimum surplus of $G$ is 2, $|N(\{v_1, v_3\})| = 4$ and hence $\{v_1, v_3\}$ is a minimum surplus set of size 2, which implies that Branching Rule B1 applies and hence contradicts the irreducibility of $G$. Hence, $g(G) \geq 5$. 

Figure 6.9: Cases of Lemma 6.4.13 when there is a 5 cycle or a 6 cycle in the graph
3. Suppose that $G$ contains a 5 cycle $v_1, \ldots, v_5$. Since $g(G) \geq 5$, this cycle does not contain chords. Let $v_i'$ denote the remaining neighbor of the vertex $v_i$ in the graph $G$. Since there are no triangles or 4 cycles, $v_i' \neq v_j'$ for any $i \neq j$, and for any $i$ and $j$ such that $|i - j| = 1$, $v_i'$ and $v_j'$ are independent. Now, we consider the following 2 cases.

**Case 1:** Suppose that for every $i, j$ such that $|i - j| \neq 1$, $v_i'$ and $v_j'$ are adjacent. Then, since $G$ is a connected 3-regular graph, $G$ has size 10, which is a contradiction.

**Case 2:** Suppose that for some $i, j$ such that $|i - j| \neq 1$, $v_i'$ and $v_j'$ are independent (see Figure 6.9). Assume without loss of generality that $i = 1$ and $j = 3$. Consider the vertex $v_1'$ and let $x$ and $y$ be the remaining 2 neighbors of $v_1'$ (the first neighbor being $v_1$). Note that $x$ or $y$ cannot be incident to $v_3$, since otherwise $x$ or $y$ will coincide with $v_3'$. Hence, $v_3$ is disjoint from $N_2[v_1']$. By Lemma 6.4.10 and Lemma 6.4.12, only Preprocessing Rule 3 applies and the applications are only on the vertices $v_1, x$ and $y$ and leaves $v_3$ untouched and the degree of vertex $v_3$ unchanged. Now, let $\hat{v}_1$ be the vertex which is created as a result of applying Preprocessing Rule 3 on $v_1$. Let $\hat{v}_4$ be the vertex created when $v_4$ is identified with another vertex during some application of Preprocessing Rule 3. If $v_4$ is untouched, then we let $\hat{v}_4 = v_4$. Similarly, let $\hat{v}_3'$ be the vertex created when $v_3'$ is identified with another vertex during some application of Preprocessing Rule 3. If $v_3'$ is untouched, then we let $\hat{v}_3' = v_3'$. Since $v_3$ is untouched and its degree remains 3 in the graph $\mathcal{R}(G \setminus \{v\})$, the neighborhood of $v_3$ in this graph can be covered by a 2 clique $\hat{v}_1, \hat{v}_4$ and a vertex $\hat{v}_3'$, which implies that Branching Rule $B_2$ applies in this graph, implying that Branching Rule $B_5$ applies in the graph $G$, contradicting the irreducibility of $G$. Hence, $g(G) \geq 6$. 

112
4. Suppose that $G$ contains a 6 cycle $v_1, \ldots, v_6$. Since $g(G) \geq 6$, this cycle does not contain chords. Let $v'_i$ denote the remaining neighbor of each vertex $v_i$ in the graph $G$. Let $x$ and $y$ denote the remaining neighbors of $v'_1$ (see Figure 6.9). Note that both $v_3$ and $v_5$ are disjoint from $N_2[v'_1]$ (if this were not the case, then we would have cycles of length $\leq 5$). Hence, by Lemma 6.4.10 and Lemma 6.4.12, we know that only Preprocessing Rule 3 applies and the applications are only on the vertices $v_1, x, y$, vertices $v_3$ and $v_5$ are untouched, and the degree of $v_3$ and $v_5$ in the graph $\mathcal{R}(G \setminus \{v'_1\})$ is 3. Let $\hat{v}_1$ be the vertex which is created as a result of applying Preprocessing Rule 3 on $v_1$. Let $\hat{v}_4$ be the vertex created when $v_4$ is identified with another vertex during some application of Preprocessing Rule 3. If $v_4$ is untouched, then we let $\hat{v}_4 = v_4$. Now, in the graph $\mathcal{R}(G \setminus \{v'_1\})$, the vertices $v_3$ and $v_5$ are independent and share two neighbors $\hat{v}_1$ and $\hat{v}_4$. The fact that they have degree 3 each and the surplus of graph $\mathcal{R}(G \setminus \{v'_1\})$ is at least 2 (Lemma 6.4.10, Lemma 6.4.7) implies that $\{v_3, v_5\}$ is a minimum surplus set of size at least 2 in the graph $\mathcal{R}(G \setminus \{v'_1\})$, which implies that branching rule $B_2$ applies in this graph, implying that Branching Rule $B_5$ applies in the graph $G$, contradicting the irreducibility of $G$. Hence, $g(G) \geq 7$.

This completes the proof of the lemma. \hfill \Box

Correctness and Analysis of the last step

In this section we combine all the results proved above and show the existence of degree 4 vertices in subsequent branchings after $B_6$. Towards this we prove the following lemma.

**Lemma 6.4.14.** Let $G$ be a connected 3 regular irreducible graph on at least 11 vertices. Then, for any vertex $v \in V$,
1. \( \mathcal{R}(G \setminus \{v\}) \) contains three degree 4 vertices, say \( w_1, w_2, w_3 \); and

2. for any \( w_i, i \in \{1, 2, 3\}, \mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_i\}) \) contains \( w_j, i \neq j \) as a degree 4 vertex.

Proof. 

1. Let \( v_1, v_2, v_3 \) be the neighbors of \( v \). Since \( G \) was irreducible, \( B_1, B_2, B_3 \) do not apply on \( \mathcal{R}(G \setminus \{v\}) \) (else \( B_5 \) would have applied on \( G \)). By Lemma 6.4.10 and Lemma 6.4.12, we know that only Preprocessing Rule 3 would have been applied to get \( \mathcal{R}(G \setminus \{v\}) \) from \( G \setminus \{v\} \) and the applications are only on these three vertices \( v_1, v_2, v_3 \). Let \( w_1, w_2 \) and \( w_3 \) be the three vertices which are created as a result of applying Preprocessing Rule 3 on these three vertices respectively. We claim that the degree of each \( w_i \) in the resulting graph is 4. Suppose that the degree of \( w_j \) is at most 3 for some \( j \). But this can happen only if there was an edge between two vertices which are at a distance of 2 from \( v \), that is, a path of length 3 between \( w_i \) and \( w_j \) for some \( i \neq j \). This implies the existence of a cycle of length 5 in \( G \), which contradicts Lemma 6.4.13.

2. Note that, by Lemma 6.4.12, it is sufficient to show that \( w_i \) is disjoint from \( N_2[w_j] \) for any \( i \neq j \). Suppose that this is not the case and let \( w_i \) lie in \( N_2[w_j] \). First, suppose that \( w_i \) lies in \( N_2[w_j] \setminus N_1[w_j] \) and there is no \( w_k \) in \( N_1[w_i] \). Let \( x \) be a common neighbor of \( w_i \) and \( w_j \). This implies that, in \( G \), \( x \) has paths of length 3 to \( v \) via \( w_i \) and via \( w_j \), which implies the existence of a cycle of length at most 6, a contradiction. Now, suppose that \( w_i \) lies in \( N_1[w_j] \). But this can happen only if there was an edge between two vertices which are at a distance of 2 from \( v \). This implies the existence of a cycle of length 5 in \( G \), contradicting Lemma 6.4.13.

The next lemma shows the correctness of deleting \( v_{yz} \) from the graph \( \mathcal{R}(G \setminus \{x\}) \) without
Lemma 6.4.15. Let $G$ be a connected irreducible graph on at least 11 vertices, $v$ be a vertex of degree 3, and $x, y, z$ be the set of its neighbors. Then, $G \setminus \{x\}$ contains a vertex cover of size at most $k$ which excludes $v$ if and only if $\mathcal{R}(G \setminus \{x\})$ contains a vertex cover of size at most $k - 3$ which contains $v_{yz}$, where $v_{yz}$ is the vertex created in the graph $G \setminus \{x\}$ by the application of Preprocessing Rule 3 on the vertex $v$.

Proof. We know by Lemma 6.4.12 that there will be exactly 3 applications of Preprocessing Rule 3 in the graph $G \setminus \{x\}$, and they will be on the three neighbors of $x$. Let $G_1$, $G_2$, $G_3$ be the graphs which result after each such application, in that order. We assume without loss of generality that the third application of Preprocessing Rule 3 is on the vertex $v$.

By the correctness of Preprocessing Rule 3, if $G \setminus \{x\}$ has a vertex cover of size at most $k$ which excludes $v$, then $G_2$ has a vertex cover of size at most $k - 2$ which excludes $v$. Since this vertex cover must then contain $y$ and $z$, it is easy to see that $G_3$ contains a vertex cover of size at most $k - 3$ containing $v_{yz}$.

Conversely, if $G_3$ has a vertex cover of size at most $k - 3$ containing $v_{yz}$, then replacing $v_{yz}$ with the vertices $y$ and $z$ results in a vertex cover for $G_2$ of size at most $k - 2$ containing $y$ and $z$ (by the correctness of Preprocessing Rule 3). Again, by the correctness of Preprocessing Rule 3, it follows that $G \setminus \{x\}$ contains a vertex cover of size at most $k$ containing $y$ and $z$. Since $v$ is adjacent to only $y$ and $z$ in $G \setminus \{x\}$, we may assume that this vertex cover excludes $v$.

Thus, when Branching Rule B6 applies on the graph $G$, we know the following about the graph.

- $G$ is a 3 regular graph. This follows from the fact that Preprocessing Rules 1, 2 and 3 and the Branching Rule B4 do not apply.
\begin{itemize}
\item \( g(G) \geq 7 \). This follows from Lemma 6.4.13.
\end{itemize}

Let \( v \) be an arbitrary vertex and \( x, y \) and \( z \) be the neighbors of \( v \). Since \( G \) is irreducible, Lemma 6.4.14 implies that \( \mathcal{R}(G \setminus \{x\}) \) contains 3 degree 4 vertices, \( w_1, w_2 \) and \( w_3 \). We let \( v_{yz} \) be \( w_1 \). Lemma 6.4.14 also implies that for any \( i \), the graph \( \mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus \{w_i\}) \) contains 2 degree 4 vertices. Since the vertex \( v_{yz} \) is one of the three degree 4 vertices, in the graph \( \mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus v_{yz}) \), the vertices \( w_2 \) and \( w_3 \) have degree 4 and one of the Branching Rules \( B_1 \), or \( B_2 \), or \( B_3 \) or \( B_4 \) will apply in this graph. Hence, we combine the execution of the rule \( B_6 \) along with the subsequent execution of one of the rules \( B_1 \), \( B_2 \), \( B_3 \) or \( B_4 \) (see Fig. 6.10). To analyze the drops in the measure for the combined application of these rules, we consider each root to leaf path in the tree of Fig. 6.10 (b) and argue the drops in each path.

\begin{itemize}
\item Consider the subtree in which \( v \) is not picked in the vertex cover from \( G \), that is, \( x \) is picked in the vertex cover, following which we branch on some vertex \( w \) during the subsequent branching, from the graph \( \mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus v_{yz}). \)

Let the instances (corresponding to the nodes of the subtree) be \((G, k), (G_1, k_1), (G_2, k_2)\) and \((G'_2, k'_2)\). That is, \( G_1 = \mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus v_{yz}) \), \( G'_2 = \mathcal{R}(G_1 \setminus \{w\}) \) and \( G_2 = \mathcal{R}(G_1 \setminus N[w]) \).

By Lemma 6.4.1, we know that \( \mu(G \setminus \{x\}, k - 1) \leq \mu(G, k) - \frac{1}{2} \). This implies that \( \mu(\mathcal{R}(G \setminus \{x\}), k') \leq \mu(G, k) - \frac{1}{2} \) where \( (\mathcal{R}(G \setminus \{x\}), k') \) is the instance obtained by applying the preprocessing rules on \( G \setminus \{x\} \).

By Lemma 6.4.1, we also know that including \( v_{yz} \) into the vertex cover will give a further drop of \( \frac{1}{2} \). Hence, \( \mu(\mathcal{R}(G \setminus \{x\}) \setminus \{v_{yz}\}, k' - 1) \leq \mu(G, k) - 1 \). Applying further preprocessing will not increase the measure. Hence \( \mu(G_1, k_1) \leq \mu(G, k) - 1 \).
\end{itemize}
Now, when we branch on the vertex \( w \) in the next step, we know that we use one of the rules \( B_1, B_2, B_3 \) or \( B_4 \). Hence, \( \mu(G_2, k_2) \leq \mu(G_1, k_1) - \frac{3}{2} \) and \( \mu(G'_2, k'_2) \leq \mu(G_1, k_1) - \frac{1}{2} \) (since \( B_4 \) gives the worst branching vector). But this implies that \( \mu(G_2, k_2) \leq \mu(G, k) - \frac{5}{2} \) and \( \mu(G'_2, k'_2) \leq \mu(G, k) - \frac{3}{2} \).

This completes the analysis of the branch of rule \( B_6 \) where \( v \) is not included in the vertex cover.

- Consider the subtree in which \( v \) is included in the vertex cover, by Lemma 6.4.14 we have that \( \mathcal{R}(G \setminus \{v\}) \) has exactly three degree 4 vertices, say \( w_1, w_2, w_3 \) and furthermore for any \( w_i, i \in \{1, 2, 3\}, \mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_i\}) \) contains 2 degree 4 vertices. Since \( G \) is irreducible, we have that for any vertex \( v \) in \( G \), the Branching Rules \( B_1, B_2 \) and \( B_3 \) do not apply on the graph \( \mathcal{R}(G \setminus \{v\}) \). Thus, we know that in the branch where we include \( v \) in the vertex cover, the first branching rule that applies on the graph \( \mathcal{R}(G \setminus \{v\}) \) is \( B_4 \). Without loss of generality, we assume that \( B_4 \) is applied on the vertex \( w_1 \). Thus, in the branch where we include \( w_1 \) in the vertex cover, we know that \( \mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_1\}) \) contains \( w_2 \) and \( w_3 \) as degree 4 vertices. This implies that in the graph \( \mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_1\}) \) one of the Branching Rules \( B_1, B_2, B_3 \) or \( B_4 \) apply on a vertex \( w_i^* \). Hence, we combine the execution of the rule \( B_6 \) along with the subsequent executions of \( B_4 \) and one of the rules \( B_1, B_2, B_3 \) or \( B_4 \) (see Fig. 6.10).

We let the instances corresponding to the nodes of this subtree be \((G, k), (G_1, k_1), (G_2, k_2), (G'_2, k'_2), (G_3, k_3) \) and \((G'_3, k'_3)\), where \( G_1 = \mathcal{R}(G \setminus \{v\}), G_2 = \mathcal{R}(G_1 \setminus N[w_1]), G'_2 = \mathcal{R}(G_1 \setminus \{w_1\}), G_3 = \mathcal{R}(G'_2 \setminus N[w_1^*]) \) and \( G'_3 = \mathcal{R}(G'_2 \setminus \{w_1^*\}) \).

Lemma 6.4.1, and the fact that preprocessing rules do not increase the measure implies that \( \mu(G_1, k_1) \leq \mu(G, k) \).
Now, since $B_4$ has been applied to branch on $w_1$, the analysis of the drop of measure due to $B_4$ shows that $\mu(G_2, k_2) \leq \mu(G_1, k_1) - \frac{3}{2}$ and $\mu(G'_2, k'_2) \leq \mu(G_1, k_1) - \frac{1}{2}$. Similarly, since, in the graph $G'_2$, we branch on vertex $w'_1$ using one of the rules $B_1$, $B_2$, $B_3$ or $B_4$, we have that $\mu(G_3, k_3) \leq \mu(G'_2, k'_2) - \frac{3}{2}$ and $\mu(G'_3, k'_3) \leq \mu(G'_2, k'_2) - \frac{1}{2}$.

Combining these, we get that $\mu(G_3, k_3) \leq \mu(G, k) - \frac{5}{2}$ and $\mu(G'_3, k'_3) \leq \mu(G, k) - \frac{3}{2}$. This completes the analysis of rule $B_6$ where $v$ is included in the vertex cover. Combining the analysis for both the cases results in a branching vector of $(\frac{3}{2}, \frac{5}{2}, \frac{5}{2}, \frac{3}{2}, 2)$ for the rule $B_6$.

Finally, we combine all the above results to obtain the following theorem.

**Theorem 6.4.16.** VERTEX COVER ABOVE LP can be solved in time $\mathcal{O}^*((2.3146)^{k-vc^*(G)})$.

**Proof.** Let us fix $\mu = \mu(G, k) = k - vc^*(G)$. We have thus shown that the preprocessing rules do not increase the measure. Branching Rules $B_1$ or $B_2$ or $B_3$ results in a $(1, 1)$ decrease in $\mu(G, k) = \mu$, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + T(\mu - 1)$ which solves to $2^\mu = 2^{k-vc^*(G)}$. 

118
<table>
<thead>
<tr>
<th>Rule</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>B5</th>
<th>B6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Vector</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>$(\frac{1}{2}, \frac{3}{2})$</td>
<td>$(\frac{3}{2}, \frac{5}{2}, 1)$</td>
<td>$(\frac{3}{2}, \frac{5}{2}, \frac{5}{2}, 2)$</td>
</tr>
<tr>
<td>Running time</td>
<td>$2^\mu$</td>
<td>$2^\mu$</td>
<td>$2^\mu$</td>
<td>$2.1479^\mu$</td>
<td>$2.3146^\mu$</td>
<td>$2.3146^\mu$</td>
</tr>
</tbody>
</table>

Figure 6.11: A table giving the decrease in the measure due to each branching rule.

Branching Rule B4 results in a $(\frac{1}{2}, \frac{3}{2})$ decrease in $\mu(G,k) = \mu$, resulting in the recurrence $T(\mu) \leq T(\mu - \frac{1}{2}) + T(\mu - \frac{3}{2})$ which solves to $2.1479^\mu = 2.1479^{k-vc^*(G)}$.

Branching Rule B5 combined with the next step in the algorithm results in a $(1, \frac{3}{2}, \frac{3}{2})$ branching vector, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + 2T(\mu - \frac{3}{2})$ which solves to $2.3146^\mu = 2.3146^{k-vc^*(G)}$.

We analyzed the way the algorithm works after an application of Branching Rule B6 before Theorem 6.4.16. An overview of drop in measure is given in Figure 6.4.6.

This leads to a $(\frac{3}{2}, \frac{5}{2}, 2, \frac{3}{2}, \frac{5}{2})$ branching vector, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + 2T(\mu - \frac{3}{2})$ which solves to $2.3146^\mu = 2.3146^{k-vc^*(G)}$.

Thus, we get an $O(2.3146^{(k-vc^*(G))\mu^{O(1)}})$ algorithm for VERTEX COVER ABOVE LP.

6.5 Applications

In this section we give several applications of the algorithm developed for VERTEX COVER ABOVE LP.

6.5.1 An algorithm for ABOVE GUARANTEE VERTEX COVER

Since the value of the LP relaxation is at least the size of the maximum matching, our algorithm also runs in time $O^*(2.3146^{k-m})$ where $k$ is the size of the minimum vertex
cover and $m$ is the size of the maximum matching.

**Theorem 6.5.1.** Above guarantee vertex cover can be solved in time $O^*(2.3146^\ell)$ time, where $\ell$ is the excess of the minimum vertex cover size above the size of the maximum matching.

Now by the known reductions in [44, 78, 88] (see also Figure 5.1) we get the following corollary to Theorem 6.5.1.

**Corollary 6.5.2.** Almost 2-SAT, Almost 2-SAT($v$), R Horn-Backdoor Set Detection can be solved in time $O^*(2.3146^k)$, and KVD$_{pm}$ can be solved in time $O^*(2.3146^{\frac{k}{2}}) = O^*(1.5214^k)$.

### 6.5.2 Algorithms for Odd Cycle Transversal and Split Vertex Deletion

We describe a generic algorithm for both Odd Cycle Transversal and Split Vertex Deletion. Let $X, Y \in \{\text{Clique, Independent Set}\}$. A graph $G$ is called an $(X,Y)$-graph if its vertices can be partitioned into $X$ and $Y$. Observe that when $X = Y = \text{independent set}$, this corresponds to a bipartite graph and when $X = \text{clique}$ and $Y = \text{independent set}$, this corresponds to a split graph. In this section we outline an algorithm that runs in time $O^*(2.3146^k)$ and solves the following problem.

<table>
<thead>
<tr>
<th>(X,Y)-DELETION SET</th>
<th>Parameter: $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An undirected graph $G$ and a positive integer $k$</td>
<td></td>
</tr>
<tr>
<td><strong>Question:</strong> Does $G$ have a vertex subset $S$ of size at most $k$ such that its deletion leaves a $(X,Y)$-graph?</td>
<td></td>
</tr>
</tbody>
</table>
We solve the \((X,Y)\)-DELETION SET problem by using a parameter preserving reduction to the ALMOST 2 SAT\((V)\) problem.

**Construction**: Given a graph \(G = (V, E)\) and \((X, Y)\), we construct a 2-SAT formula \(\phi(G, X, Y)\) as follows. The formula \(\phi(G, X, Y)\) has a variable \(x_v\) for each vertex \(v \in V\). We now add the following clauses to \(\phi(G, X, Y)\).

- If \(X = \text{clique}\), then, for every non-edge \((u, v) \notin E\), we add the clause \((x_u \lor x_v)\).
- If \(X = \text{independent set}\), then for every edge \((u, v) \in E\), we add the clause \((x_u \lor x_v)\).

Similarly, if \(Y = \text{clique}\), then for every non-edge \((u, v) \notin E\), we add the clause \((\bar{x}_u \lor \bar{x}_v)\) and if \(Y = \text{independent set}\), then for every edge \((u, v) \in E\), we add the clause \((\bar{x}_u \lor \bar{x}_v)\). This completes the construction of \(\phi(G, X, Y)\).

**Lemma 6.5.3.** Given a graph \(G = (V, E)\) and \((X, Y)\), let \(\phi(G, X, Y)\) be the 2-SAT formula obtained by the above construction. Then, \((G, k)\) is a Yes instance of \((X,Y)\)-DELETION SET if and only if \((\phi(G, X, Y), k)\) is a Yes instance of ALMOST 2 SAT\((V)\).

**Proof.** Suppose there is a set \(S \subseteq V\) such that \(|S| \leq k\) and \(G \setminus S\) is an \((X,Y)\)-graph. Let \(S_v\) be the set of variables of \(\phi = \phi(G, X, Y)\) which correspond to the vertices in \(S\). Clearly, \(|S_v| \leq k\). We claim that \(\phi \setminus S_v\) is satisfiable by the following assignment. For each vertex in the \(X\)-partition of \(G \setminus S\), assign the corresponding variable the value 0 and for each vertex in the \(Y\)-partition of \(G \setminus S\), assign the corresponding variable the value 1.

Suppose that this assignment does not satisfy \(\phi \setminus S_v\) and let \(C\) be an unsatisfied clause. By the construction, we know that \(C\) is of the form \((x_u \lor x_v)\) or \((\bar{x}_u \lor \bar{x}_v)\). We consider only the first case, since the second is analogous to the first. If \((u, v) \in E\), then it must be the case that \(X = \text{independent set}\) (by construction). Since this clause is unsatisfied, the value assigned to both \(x_u\) and \(x_v\) was 0. But this implies that \(u\) and \(v\) lie in the \(X\)-partition of \(G \setminus S\), where \(X = \text{independent set}\), which is a contradiction. Similarly, if
(u, v) \not\in E, then it must be the case that \( X = \text{clique} \) (by construction). Since this clause is unsatisfied, the value assigned to both \( x_u \) and \( x_v \) was 0. But this implies that \( u \) and \( v \) lie in the \( X \)-partition of \( G \setminus S \), where \( X = \text{clique} \), which is a contradiction.

Conversely, let \( S_v \) be a set of variables of \( \phi = \phi(G, X, Y) \) such that \( |S_v| \leq k \) and \( \phi \setminus S_v \) is satisfiable. Let \( \rho \) be a satisfying assignment to \( \phi \setminus S_v \) and let \( S \) be the set of vertices of \( G \) which correspond to \( S_v \). Clearly, \( |S| \leq k \). We now define the following partition of the vertices in \( G \setminus S \). For each vertex of \( G \setminus S \), if the corresponding variable is assigned 0 by \( \rho \), then add it into partition \( A \) or into partition \( B \) otherwise. We claim that the partition \( (A, B) \) of \( G \setminus S \) is an \((X, Y)\) partition. Suppose that \( A \) is not an \( X \)-partition, where \( X = \text{clique} \). We only consider this case since the remaining cases can be argued analogously. Consider a non-edge \((u, v)\) such that \( u, v \in A \). But, by the construction, \( \phi \) contains the clause \((x_u \lor x_v)\). Since \( G \setminus S \) contains both the vertices \( u \) and \( v \), it must be the case that \( \phi \setminus S_v \) contains both \( x_u \) and \( x_v \), implying that it contains the clause \((x_u \lor x_v)\). But, by the construction of the set \( A \), \( \rho \) assigned 0 to both \( x_u \) and \( x_v \), which is a contradiction. This completes the proof of the lemma.

Combining the above lemma with Theorem 6.5.1, we have the following.

**Theorem 6.5.4.** \((X, Y)\)-DELETION SET can be solved in time \( O^*(2^{3.146k}) \).

As a corollary to the above theorem we get the following new results.

**Corollary 6.5.5.** ODD CYCLE TRANSVERSAL and SPLIT VERTEX DELETION can be solved in time \( O^*(2^{3.146k}) \).

Observe that the reduction from EDGE BIPARTIZATION to ODD CYCLE TRANSVERSAL represented in Figure 5.1, along with the above corollary implies that EDGE BIPARTIZATION can also be solved in time \( O^*(2^{3.146k}) \). However, we note that there is an algorithm for this problem due to Guo et al. [46], running in time \( O^*(2^k) \).
6.5.3 An algorithm for König Vertex Deletion

A graph $G$ is called König if the size of a minimum vertex cover equals that of a maximum matching in the graph. Clearly bipartite graphs are König but there are non-bipartite graphs that are König (a triangle with an edge attached to one of its vertices, for example). Thus the König Vertex Deletion problem, as stated below, is closely connected to Odd Cycle Transversal.

<table>
<thead>
<tr>
<th>König Vertex Deletion (KVD)</th>
<th>Parameter: $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An undirected graph $G$ and a positive integer $k$</td>
<td></td>
</tr>
<tr>
<td><strong>Question:</strong> Does $G$ have a vertex subset $S$ of size at most $k$ such that $G \setminus S$ is a König graph?</td>
<td></td>
</tr>
</tbody>
</table>

If the input graph $G$ to König Vertex Deletion has a perfect matching then this problem is called KVD$_{pm}$. By Corollary 6.5.2, we already know that KVD$_{pm}$ has an algorithm with running time $O^*(1.5214^k)$ by a polynomial time reduction to Above Guarantee Vertex Cover, that maps $k$ to $k/2$. However, there is no known reduction if we do not assume that the input graph has a perfect matching and it required several interesting structural theorems in [81] to show that KVD can be solved as fast as Above Guarantee Vertex Cover. Here, we outline an algorithm for KVD that runs in $O^*(1.5214^k)$ and uses an interesting reduction rule. However, for our algorithm we take a detour and solve a slightly different, although equally interesting problem. Given a graph, a set $S$ of vertices is called König vertex deletion set (kvd set) if its removal leaves a König graph. The auxiliary problem we study is following.
### Parameter: $k$

**Input:** An undirected graph $G$, a König vertex deletion set $S$ of size at most $k$ and a positive integer $\ell$

**Question:** Does $G$ have a vertex cover of size at most $\ell$?

This fits into the recent study of problems parameterized by other structural parameters. See, for example, **ODD CYCLE TRANSVERSAL** parameterized by various structural parameters [57] or **TREEWIDTH** parameterized by vertex cover [7] or **VERTEX COVER** parameterized by feedback vertex set [56] or **DOMINATING SET** parameterized by max-leaf number [30]. For our proofs we will use the following characterization of König graphs.

**Lemma 6.5.6.** [81, Lemma 1] A graph $G = (V, E)$ is König if and only if there exists a bipartition of $V$ into $V_1 \cup V_2$, with $V_1$ a vertex cover of $G$ such that there exists a matching across the cut $(V_1, V_2)$ saturating every vertex of $V_1$.

Note that in **VERTEX COVER PARAM BY KVD**, $G \setminus S$ is a König graph. So one could branch on all subsets of $S$ to include in the output vertex cover, and for those elements not picked in $S$, we could pick its neighbors in $G \setminus S$ and delete them. However, the resulting graph need not be König adding to the complications. Note, however, that such an algorithm would yield an $O^*(2^k)$ algorithm for **VERTEX COVER PARAM BY OCT**. That is, if $S$ were an odd cycle transversal then the resulting graph after deleting the neighbors of vertices not picked from $S$ will remain a bipartite graph, where an optimum vertex cover can be found in polynomial time.

Given a graph $G = (V, E)$ and two disjoint vertex subsets $V_1, V_2$ of $V$, we let $(V_1, V_2)$ denote the bipartite graph with vertex set $V_1 \cup V_2$ and the edge set described as $\{\{u, v\} : \{u, v\} \in E \text{ and } u \in V_1, v \in V_2\}$. Now, we describe an algorithm based on Theorem 6.3.8, that solves **VERTEX COVER PARAM BY KVD** in time $O^*(1.5214^k)$.
Theorem 6.5.7. Vertex Cover Param by KVD can be solved in time $O^*(1.5214^k)$.

Proof. Let $G$ be the input graph, $S$ be a kvd set of size at most $k$. We first apply Lemma 6.3.1 on $G = (V, E)$ and obtain an optimum solution to LPVC($G$) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V_{1/2}]$). Due to Lemma 6.3.2, this implies that there exists a minimum vertex cover of $G$ that contains all the vertices in $V^x_1$ and none of the vertices in $V^x_0$. Hence, the problem reduces to finding a vertex cover of size $\ell' = \ell - |V^x_1|$ for the graph $G' = G[V_{1/2}]$. Before we describe the rest of the algorithm, we prove the following lemma regarding kvd sets in $G$ and $G'$ which shows that if $G$ has a kvd set of size at most $k$ then so does $G'$. Even though this looks straight forward, the fact that König graphs are not hereditary (i.e. induced subgraphs of König graphs need not be König) makes this a non-trivial claim to prove.

Lemma 6.5.8. Let $G$ and $G'$ be defined as above. Let $S$ be a kvd set of graph $G$ of size at most $k$. Then, there is a kvd set of graph $G'$ of size at most $k$.

Proof. It is known that the sets $(V^x_0, V^x_1, V^x_{1/2})$ form a crown decomposition of the graph $G$ [20]. In other words, $N(V^x_0) = V^x_1$ and there is a matching saturating $V^x_1$ in the bipartite graph $(V^x_1, V^x_0)$. The set $V^x_0$ is called the crown and the set $V^x_1$ is called the head of the decomposition. For ease of presentation, we will refer to the set $V^x_0$ as $C$, $V^x_1$ as $H$ and the set $V^x_{1/2}$ as $R$. In accordance with Lemma 6.5.6, let $A$ be the minimum vertex cover and let $I$ be the corresponding independent set of $G \setminus S$ such that there is a matching saturating $A$ across the bipartite graph $(A, I)$. First of all, note that if the set $S$ is disjoint from $C \cup H$, $H \subseteq A$, and $C \subseteq I$, we are done, since the set $S$ itself can be taken as a kvd set for $G'$. This last assertion follows because there exists a matching saturating $H$ into $C$. Hence, we may assume that this is not the case. However, we will argue that given a kvd set of $G$ of size at most $k$ we will always be able to modify it in a way that it is of size at most $k$, it is disjoint from $C \cup H$, $H \subseteq A$, and $C \subseteq I$. This will
allow us to prove our lemma. Towards this, we now consider the set $H' = H \cap I$ and consider the following two cases.

1. $H'$ is empty. We now consider the set $S' = S \setminus (C \cup H)$ and claim that $S'$ is also a kvd set of $G$ of size at most $k$ such that $G \setminus S'$ has a vertex cover $A' = (A \setminus C) \cup H$ with the corresponding independent set being $I' = I \cup C$. In other words, we move all the vertices of $H$ to $A$ and the vertices of $C$ to $I$. Clearly, the size of the set $S'$ is at most that of $S$. The set $I'$ is independent since $I$ was initially independent, and the newly added vertices have edges only to vertices of $H$, which are not in $I'$. Hence, the set $A'$ is indeed a vertex cover of $G \setminus S'$. Now, the vertices of $R$, which lie in $A$, (and hence $A'$) were saturated by vertices not in $H$, since $H \cap I$ was empty. Hence, we may retain the matching edges saturating these vertices, and as for the vertices of $H$, we may use the matching edges given by the crown decomposition to saturate these vertices and thus there is a matching saturating every vertex in $A'$ across the bipartite graph $(A', I')$. Hence, we now have a kvd set $S'$ disjoint from $C \cup H$, such that $H$ is part of the vertex cover and $C$ lies in the independent set of the König graph $G \setminus S'$.
2. $H'$ is non empty. Let $C_1$ be the set of vertices in $A \cap C$ which are adjacent to $H'$ (see Fig. 6.5.8), let $C_2$ be the set of vertices in $C \cap S$, which are adjacent to $H'$, and let $P$ be the set of vertices of $R \cap A$ which are saturated by vertices of $H'$ in the bipartite graph $(A, I)$. We now consider the set $S' = (S \setminus C_2) \cup P$ and claim that $S'$ is also a kvd set of $G$ of size at most $k$ such that $G \setminus S'$ has a minimum vertex cover $A' = (A \setminus (C_1 \cup P)) \cup H'$ with the corresponding independent set being $I' = (I \setminus H') \cup (C_1 \cup C_2)$. In other words, we move the set $H'$ to $A$, the sets $C_1$ and $C_2$ to $I$ and the set $P$ to $S$. The set $I'$ is independent since $I$ was independent and the vertices added to $I$ are adjacent only to vertices of $H$, which are not in $I'$. Hence, $A'$ is indeed a vertex cover of $G \setminus S'$. To see that there is still a matching saturating $A'$ into $I'$, note that any vertex previously saturated by a vertex not in $H$ can still be saturated by the same vertex. As for vertices of $H'$, which have been newly added to $A$, they can be saturated by the vertices in $C_1 \cup C_2$. Observe that $C_1 \cup C_2$ is precisely the neighborhood of $H'$ in $C$ and since there is a matching saturating $H$ in the bipartite graph $(H, C)$ by Hall’s Matching Theorem we have that for every subset $\hat{H} \subseteq H$, $|N(\hat{H}) \cap (C_1 \cup C_2)| \geq |\hat{H}|$. Hence, by Hall’s Matching Theorem there is a matching saturating $A'$ in the bipartite graph $(A', I')$. It now remains to show that $|S'| \leq k$.

Since $N(H') = C_1 \cup C_2$ in the bipartite graph $(C, H)$, we know that $|C_1| + |C_2| \geq |H'|$. In addition, the vertices of $C_1$ have to be saturated in the bipartite graph $(A, I)$ by vertices in $H'$. Hence, we also have that $|C_1| + |P| \leq |H'|$. This implies that $|C_2| \geq |P|$. Hence, $|S'| \leq |S| \leq k$. This completes the proof of the claim.

But now, notice that we have a kvd set of size at most $k$ such that there are no vertices of $H$ in the independent set side of the corresponding König graph. Thus, we have fallen into Case 1, which has been handled above.
This completes the proof of the lemma.

We now show that $\mu = \text{vc}(G') - \text{vc}^*(G') \leq \frac{k}{2}$. Let $O$ be a kvd set of $G'$ and define $G''$ as the König graph $G' \setminus O$. It is well known that in König graphs, $|M| = \text{vc}(G'') = \text{vc}^*(G'')$, where $M$ is a maximum matching in the graph $G''$. This implies that $\text{vc}(G') \leq \text{vc}(G'') + |O| = |M| + |O|$. But, we also know that $\text{vc}^*(G') \geq |M| + \frac{1}{2}(|O|)$ and hence, $\text{vc}(G') - \text{vc}^*(G') \leq \frac{1}{2}(|O|)$. By Lemma 6.5.8, we know that there is an $O$ such that $|O| \leq k$ and hence, $\text{vc}(G') - \text{vc}^*(G') \leq \frac{k}{2}$.

By Corollary 6.3.9, in time $O^*(2.3146^{\text{vc}(G')-\text{vc}^*(G')})$, we can compute a minimum vertex cover of $G'$ and hence in time $O^*(2.3146^{k/2})$. If the size of the minimum vertex cover obtained for $G'$ is at most $\ell'$, then we return yes else we return no. We complete the proof of the theorem with a remark that, in the algorithm described above, we do not, in fact, even require a kvd set to be part of the input.

It is known that, given a minimum vertex cover, a minimum sized kvd set can be computed in polynomial time [81]. Hence, Theorem 6.5.7 has the following corollary.

**Corollary 6.5.9.** KVD can be solved in time $O^*(1.5214^k)$.

Since the size of a minimum Odd Cycle Transversal is at least the size of a minimum König Vertex Deletion set, we also have the following corollary.

**Corollary 6.5.10.** VERTEX COVER PARAM BY OCT can be solved in time $O^*(1.5214^k)$.

### 6.5.4 A simple improved kernel for VERTEX COVER

We give a kernelization for VERTEX COVER based on Theorem 6.3.8 as follows. Exhaustively, apply the Preprocessing rules 1 through 3 (see Section 6.3). When the rules no longer apply, if $k - \text{vc}^*(G) \leq \log k$, then solve the problem in time $O^*(2.3146^{\log k})$ =
\(O(n^{O(1)})\). Otherwise, just return the instance. We claim that the number of vertices in the returned instance is at most \(2k - 2\log k\). Since \(k - \text{vc}^*(G) > \log k\), \(\text{vc}^*(G)\) is upper bounded by \(k - \log k\). But, we also know that when Preprocessing Rule 1 is no longer applicable, all \(\frac{1}{2}\) is the unique optimum to LPVC\((G)\) and hence, the number of vertices in the graph \(G\) is twice the value of the optimum value of LPVC\((G)\). Hence, \(|V| = 2\text{vc}^*(G) \leq 2(k - \log k)\). Observe that by the same method we can also show that in the reduced instance the number of vertices is upper bounded by \(2k - c\log k\) for any fixed constant \(c\). Independently, Lampis [65] has also shown an upper bound of \(2k - c\log k\) on the size of a kernel for VERTEX COVER for any fixed constant \(c\).

### 6.6 Conclusion

We have demonstrated that using the change in LP values to analyze branching algorithms can give powerful results for parameterized complexity. We believe that our algorithm is the beginning of a race to improve the running time bound for ABOVE GUARANTEE VERTEX COVER. Furthermore, the running time bound for the classical VERTEX COVER problem, has seen no improvement in the last several years after an initial plethora of results. We believe that our algorithm may lead towards an improvement in this time bound by reducing the need to resort to too many refined branchings, which is possibly the reason why the progress in this direction has stagnated.

Our other contribution is to exhibit several parameterized problems that are equivalent to or reduce to ABOVE GUARANTEE VERTEX COVER through parameterized reductions.