# 1 Introduction

## 1.1 Motivation

The problem *maximum independent set* is a classic NP-complete graph problem [9] and therefore has been well studied over the last decades. Given an undirected graph, the problem is to find a set of pairwise non-adjacent vertices of largest cardinality.

Applications of *maximum independent set* and its complementary problems *minimum vertex cover* and *maximum clique* cover a variety of fields including computer graphics [17], network analysis [16], rout planning [11] and computational biology [3, 7], among others. Unfortunately, due to the complexity of those problems, finding exact solutions to most real-world instances is computationally infeasible. However, a lot of work is invested into finding new techniques to handle the complexity.

One of the best known techniques for finding exact solutions to those problems, both in theory and practice, are branch and bound algorithms. Such algorithms are generally based on kernelization. This means applying a set of reduction rules to decrease the complexity measure (in most cases the size) of an instance while still preserving the solvabilty. A solution to the original instance can then be constructed from a solution of the reduced instance in subexponential time. If an instance can not be reduced further, the algorithm branches into at least two subproblems of lower complexity which are then solved recursively. Branch and bound algorithms also use problem specific upper and lower bounds to a solution and prune the search space by eliminating solutions which do not satisfy those bounds.

So far most studies on branch and bound algorithms for the *maximum independent set* problem solely focused on finding new and improved reduction rules and lower or upper bounds, respectively. There are very few publications regarding different branching strategies. However, a comparison of three simple branching strategies by Akiba and Iwata [1] shows that the branching strategy can have a huge impact on the running time of an algorithm.

## 1.2 Contributions

This Thesis experimentally examines various different branching strategies for *maximum independent set*. We essentially followed two main approaches. The first approach is to branch on vertices that decompose the graph and then solve the resulting connected components independently. The second approach is to destroy complex structures by branching on vertices such that the structure can be reduced by kernelization afterwards. We implemented a variety of different branching strategies following both approaches and compared them to the branching strategy used by the state of the art branch and bound algorithm for *maximum independent set* proposed by Akiba and Iwata [1]. For testing we used instances from multiple graph classes.

Branching strategies following the first approach can also be used for other graph problems, but we did not evaluate this. We also did not analyze any of the branching strategies on a theoretical level.

## 1.3 Structure of Thesis

Following the brief introduction (section 1), in section 2 we state the notation and problem definitions used throughout this Thesis. Here, we also explain the two variants of a branch and bound algorithm for *maximum independent set* that we used as a basic framework for testing our branching strategies. In particular we define the reduction rules used by the algorithm.
Section 3 gives an overview of related work focusing on branching strategies used by other branch and bound algorithms for *maximum independent set* or the equivalent problems *minimum vertex cover*

and *maximum clique*. In section 4 we outline our approaches and explain the implemented branching strategies in detail. Section 5 contains the experimental results. We start by explaining our testing methodology and then state our results. Subsequently, we compare all branching strategies to each other and discuss the effectiveness of our approaches. Finally, in section 6 proposals for future work will be discussed based on the results of the Thesis.

## 2 Preliminaries

This section introduces basic notation and problem definitions used throughout this Thesis.

### 2.1 Basic Definitions

An undirected graph $G = (V, E)$ is a tuple of a set $V$ of vertices (also called nodes) and a set $E \subseteq \binom{V}{2}$ of edges. Two vertices $v, u \in V$ are called *adjacent* or neighbours, if they are connected by an edge, i.e. $\{v, u\} \in E$. The set $N(v) := \{u \in V \mid \{v, u\} \in E\}$ of all neighbours of a vertex $v \in V$ is called the *neighbourhood* of $v$ and $N[v] := N(v) \cup \{v\}$ denotes to the *closed* neighbourhood of $v$. For a subset $S \subseteq V$ the graph $G_S = (S, E \cap \binom{S}{2})$ is called a the subgraph *induced by S*.

A subset $I \subseteq V$ is called an *independent set* of $G$, if no two vertices from $I$ are adjacent, so formally if $\forall v, u \in I : \{v, u\} \notin E$. A *maximum independent set* of $G$ is an *independent set* of largest cardinality.
A subset $S \subseteq V$ is called a *vertex cover* of $G$, if for all neighbors $v$ and $u$ in $G$ either $v$ or $u$ (or both) is in $S$, so formally if $\forall\{v, u\} \in E : v \in S \vee u \in S$. A *minimum vertex cover* of $G$ is a vertex cover of minimal cardinality. If $I$ is an independent set in $G$, then $V \setminus I$ is a vertex cover in $G$.
A subset $C \subseteq V$ is called a *clique* of $G$, if any two vertices from $C$ are adjacent, so formally if $\forall v, u \in C : v, u \in E$. A *maximum clique* of $G$ is a *clique* of largest cardinality. If $I$ is an (*maximum*) independent set in $G$, then $I$ is also a (*maximum*) clique in the complement graph $\overline{G} = (V, \binom{V}{2} \setminus E)$.

A *simple path* $P = (v_1, \ldots, v_k)$ is a sequence of distinct vertices in $G$ such that $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, \ldots, k-1\}$. A subgraph of $G$ induced by a cardinality maximal subset of the vertices such that any two vertices are connected by a path is called a *connected component*. A graph that contains only one connected component is called *connected*.

A partition of $V$ into $(S, T)$ is denoted a cut and the set $C = \{\{v, u\} \in E \mid v \in S,\ u \in T\}$ is called its *cut set*. A subset $S \subset V$ is *called* a separator, if $G - S := (V \setminus S, E \cap \binom{V \setminus S}{2})$ has more connected components than $G$ i.e. the removal of $S$ from the graph splits at least one connected component of $G$.

### 2.2 Algorithm Framework

In this Thesis we solely focused on testing new branching strategies. Thus, we did not implement our own branch and bound algorithm but rather used a state of the art algorithm for *minimum vertex cover* by Akiba and Iwata [1] as a basis, and modified the branching step within it. Since *minimum vertex cover* and *maximum independet set* are complementary problems, the algorithm can be used to find the latter one by just inverting its output. This subsection briefly covers an overview of the algorithm (Algorithm 1).

Given a graph as input, the algorithm starts with the kernelization step, i.e., reducing the instance's complexity by applying a set of reduction rules (describe in 2.3). Next, the algorithm tries to prune the current branch by using different lower bounds to an optimal solution. If pruning was not successful but the reduced graph is empty, the algorithm just returns the current best solution size. Otherwise, the algorithm searches for an optimal solution in every connected component of the graph independently. This is denoted as the decompose step. If a connected component can not be reduced further, the algorithms performs a branching step. In the implementation by Akiba and Iwata, the algorithm finds a vertex $v$ of maximum degree that also minimizes the number of edges in its neighborhood and branches into two subinstances. The first case is to include $v$ into the current solution and the second case is to exclude $v$ from the current solution, including its neighbors instead. Both subinstances are then solved recursively and the current optimal solution gets updated accordingly. The algorithms also makes use of branching rules covered in 2.3 (not to confuse with the actual branching strategy) that sometimes allow further reductions on branching. The *packing* branching rule manages a set of constraints which is updated on every branch and reduction step, and therefore is also handed to the recursive subcalls. For clarity, we omitted the details of this in the pseudo code.

All of our branching strategies branch on a single vertex in each branching step. Thus, in the algorithm we only needed to change the method that selects the vertex to branch in most cases. Some of our strategies also maintain additional information which gets updated in each branching step and gets distributed to subinstances accordingly. Other branching strategies require structural information which is obtained during each kernelization step. In those cases we also modified the decompose step or the reduction rules, respectively

Since our first approach was to branch on vertices that decompose the graph, we also tested a slightly modified version of the algorithm where we added an additional connected components check before the kernelization step. This way, if the graph gets disconnected in a branching step, the instance is decomposed and kernelization afterwards becomes more efficient.

**Algorithm 1:** branch & bound algorithm for Min Vertex Cover – Akiba and Iwata [1]

```
Input: A graph $G$, current solution size $c$, current best solution size $k$
1  Solve($G, c, k$) begin
2      $G, c \leftarrow$ Reduce($G, c$)                         // Kernelization
3      $l \leftarrow$ LowerBound(G)                        // Calculate lower bounds
4      if $c + l \geq k$ then return $k$                     // Prune current branch
5      if $G$is empty then return $k$
6      if $G$ is not connected then
7          foreach connected component $G_i$ of $G$ do
8              $c \leftarrow c +$ Solve($G_i, 0, k - c$)      // Solve connected components independently
9          return $\min\{c, k\}$
10     $(G_1, c_1, k), (G_2, c_2, k) \leftarrow$ Branch($G, c$)     // Branch on a vertex $v$ into two subcases
11     $k \leftarrow \min\{k,$ Solve($G_1, c_1, k$)$\}$
12     $k \leftarrow \min\{k,$ Solve($G_2, c_2, k$)$\}$
13     return $k$
Output: the size $k$ of a minimum vertex cover or the size $n - k$ of a maximum independent set
```

## 2.3 Reduction and Branching Rules

In this subsection we explain the various reduction and branching rules used in the kernelization and branching steps of the algorithm by Akiba and Iwata [1]. We formulate all reduction and branching rules for the *maximum independent set* problem, although the algorithm was originally designed for *minimum vertex cover* and therefore uses the equivalent counterparts of those rules. The algorithm also keeps track of the order in which the reduction rules are applied, such that a correct solution to the original instance (a specific *maximum independent set* and not only the size of one) can be constructed from a solution in the reduced graph later on. For clarity, we omitted the details of this.

The first reduction rule, the degree one reduction, is actually completely contained in the dominance and unconfined reductions (covered later in this section). Nevertheless, due to its low computational costs, it is used in addition to those more general rules.

**Theorem 1.** *(Degree One Reduction) Let $G = (V, E)$ be a graph with a vertex $v$ of degree one and let $u$ be the only neighbor of $v$. Then, there is a maximum independent set that includes $v$ and therefore excludes $u$.*

*Proof.* Consider a *maximum independent set* $I$ in $G$. $I$ has to contain either $u$ or $v$ because otherwise $I \cup \{v\}$ would be an *independent set* of larger size. If $I$ contains $u$, it can not contain $v$ and thus, $(I \setminus \{u\}) \cup \{v\}$ is another *maximum independent set* that include $v$ and excludes $u$. $\square$

At the beginning of each kernelization step, the algorithm searches for vertices of degree one, includes them into the current solution and deletes their neighbors from the graph. The algorithm also checks whether removing a neighbor from the graph produces new degree one vertices, and in this case further applies the degree one reduction.

The next reduction rule deals with vertices of degree two that are not part of a triangle, i.e., whose neighbors are not adjacent and was introduced by Chen et al. [5].

**Theorem 2.** *(Degree Two Folding) Let $G = (V, E)$ be a graph with a vertex $v$ of degree two and let $u, w$ be the neighbors of $v$. Let $G' = (V', E')$ be the graph with $V' = (V \setminus N[v]) \cup \{x\}$ where $x \notin V$ and $E' = \{\{y, z\} \in E \mid y, z \in V'\} \cup \{\{x, y\} \mid y \in (N(u) \cup N(w)) \setminus \{v\}\}$ and let $I'$ be a maximum independent set of $G'$. Then,*

$$I = \begin{cases} I' \cup \{v\} & x \notin I' \\ (I' \setminus \{x\}) \cup \{u, w\} & x \in I' \end{cases}$$

*is a maximum independent set in $G$.*

*Proof.* Consider any *maximum independent set* $I$ of $G$. If $I$ contains $v$, then it can not contain $u$ and $w$. Thus, $I \setminus \{v\}$ is an *independent set* in $G'$ of size $|I| - 1$. Otherwise, if $I$ does not contain $v$, $I$ has to include at least one neighbor of $v$ (since $I$ is maximal). If $I$ contains only one neighbor of $v$, removing this neighbor from $I$ yields an *independent set* in $G'$ of size $|I| - 1$. If $I$ contains both $u$ and $w$, then $I' = (I \setminus \{u, w\}) \cup \{x\}$ is an independent set in $G'$ of size $|I| - 1$. So, in total $\alpha(G') \geq \alpha(G) - 1$. On the other hand, $I$ constructed from a *maximum independent set* $I'$ of $G'$ is an *independent set* of $G$ of size $|I| = |I'| + 1$ and thus, $I$ is a *maximum independent set* in $G$. $\square$

So, if the algorithm finds a vertex $v$ of degree two whose neighbors are not adjacent, the algorithm reduces the size of the graph by removing $N[v]$ adding a new vertex connected to $N_2(v)$ instead. This procedure is called *folding* the closed neighborhood of $v$, hence the name degree two folding.

The next two reduction rules can be used to delete single vertices that are not required in a *maximum independent set*. The first of those rules (the dominance Reduction) is fully contained in the second rule (the unconfined reduction) and therefore the algorithm only uses the latter one. However, we used the concept of dominance to design one of our branching strategies. For this reason the dominance reduction rule is also featured in detail.

**Definition 1.** *(Dominance) In a Graph $G = (V, E)$ a vertex $u$ is called dominated by a neighbor $v$, if $N[u] \subseteq N[v]$. The vertex $v$ is said to dominate $u$.*

**Theorem 3.** *(Dominance Reduction) In a Graph $G = (V, E)$, if a vertex $u$ is dominated by a neighbor $v$, then, there always exists a maximum independent set that does not include $v$, i.e.*

$$\alpha(G) = \alpha(G - v)$$

*Proof.* Consider a *maximum independent set* $I$ that does contain $v$. Since $N[u] \subseteq N[v]$, $I$ can neither contain $u$ nor any of its neighbors. But then, clearly, $I' = (I \setminus \{v\}) \cup \{u\}$ is an *independent set* of the same size as $I$ that does not include $v$. □

Thus, if a vertex $v$ dominates another vertex $u$, one could safely remove $v$ from the graph without compromising the solvabilty of the instance.

The core idea of the *unconfined reduction* proposed by Xiao and Nagamochi [19] is to detect a vertex that is not required for a *maximum independent set* and therefore can be removed from the graph by algorithmically contradicting the assumption that every *maximum independent set* contains the vertex.

**Definition 2.** *(Removable Vertex) In a graph $G = (V, E)$ a vertex $v$ is called removable, if*

$$\alpha(G) = \alpha(G - v)$$

**Definition 3.** *(Child, Parent) In a Graph $G = (V, E)$ with an independent set $I$, a vertex $v$ is called a child of $I$, if $|N(v) \cap I| = 1$ and the unique neighbor of $v$ in $I$ is called the parent of $v$.*

**Theorem 4.** *In a graph $G = (V, E)$ let $S$ be an independent set that is not maximal but is contained in every maximum independent set of $G$ and let $v$ be any child of $S$. Then, every maximum independent set includes at least one vertex from $N(v) \setminus N[S]$.*

*Proof.* Assume that there is a *maximum independent set* $I$ that includes $S$ but no vertex from $N(v) \setminus N[s]$ and let $u$ be the parent of $v$ in $S$. But then $I' = (I \setminus \{u\}) \cup \{v\}$ is an independent set of the same size as $I$, since $I$ contains no neighbor of $v$ other than $u$. This contradicts the fact that every *maximum independent set* includes $S$. □

Based on Theorem 4 the following algorithm detects so called *unconfined* vertices....TODO

**Theorem 5.** *(Unconfined Reduction)*

...TODO

The twin reduction by Xiao and Nagamochi [18] deals with pairs of degree three vertices that share the same neighborhood.

**Definition 4.** *(Twins) In a Graph $G = (V, E)$ two vertices $u$ and $v$ are called twins, if $N(u) = N(v)$ and $d(u) = d(v) = 3$.*

**Theorem 6.** *(Twin Reduction) In a Graph $G = (V, E)$ let vertices $u$ and $v$ be twins. If there is an edge among $N(u)$, then there is always a maximum independent set that includes $\{u, v\}$ and therefore excludes $N(u)$. Otherwise, let $G' = (V', E')$ be the graph with $V' = (V \setminus (N(u) \cup \{u, v\})) \cup \{w\}$ where $w \notin V$ and $E' = \{\{x, y\} \in E \mid x, y \in V'\} \cup \{\{w, x\} \mid x \in N_2(u)\})\}$ and let $I'$ be a maximum vertex cover in $G'$. Then,*

$$I = \begin{cases} I' \cup \{u, v\} & w \notin I' \\ (I' \setminus \{w\}) \cup N(u) & w \in I' \end{cases}$$

*is a maximum independent set in $G$.*

*Proof.* For the first case (there is an edge among $N(u)$) consider a *maximum independent set* $I$ that does not contain $u$ or $v$. Then, $I$ has to include at least one neighbor of $u$ and $v$, because otherwise $I \cup \{u, v\}$ would be an *independent set* larger than $I$. On the other hand, since there are neighbors of $u$ and $v$ that are adjacent, $I$ can only contain at most two neighbors of $u$ and $v$. But then, $I' = (I \setminus N(u)) \cup \{u, v\}$ is an *independent set* of the same size as I that includes $u$ and $v$.
For the second case (no edges among $N(u)$) note, that the reduction produces a set that in both cases contains exactly two vertices more than a *maximum independent set* in $G'$. Now consider a *maximum independent set* $I$ in $G$. If $N(u)$ is completely contained in $I$ ($N(U) \subseteq I$), then $I$ can not contain any vertex of $N_2(u)$, i.e., any neighbor of $w$ in $G'$. Thus, $I' = (I \setminus N(u)) \cup \{w\}$ is an *independent set* of $G'$ of size $|I| - 2$. Otherwise, $I$ contains at most two vertices from $N(u) \cup \{u, v\}$ (either $u$ and $v$ or two vertices from $N(u)$). But then, $I' = I \setminus (N(u) \cup \{u, v\})$ is also an *independent set* of $G'$ of size $|I| - 2$. In total $\alpha(G) \leq \alpha(G') + 2$ and thus, $I$ is a *maximum independent set* of $G$ □

During the kernelization step, the algorithm searches for twins $u$ and $v$. If there is an edge among $N(u)$, the algorithm includes $u$ and $v$ to the current solution and deletes $\{u, v\} \cup N(u)$. Otherwise, the algorithm still deletes $\{u, v\} \cup N(u)$ introducing a new vertex connected to $N(u) \setminus \{u, v\}$ instead.

The next reduction rule as well as its special cases were also proposed by Xiao and Nagamochi [18].

**Definition 5.** *(Alternative Sets) In a Graph $G = (V, E)$ two non empty, disjoint subsets $A, B \subseteq V$ are called alternatives, if $|A| = |B|$ and there is a maximum independent set $I$ in $G$ such that $I \cap (A \cup B)$ is either $A$ or $B$.*

**Theorem 7.** *(Alternative Reduction) In a Graph $G = (V, E)$ let $A$ and $B$ be alternative sets. Let $G' = (V', E')$ the graph with $V' = V \setminus (A \cup B \cup (N(A) \cap N(B)))$ and $E' = (E \setminus \{\{x, y\} \mid x \in (A \cup B \cup (N(A) \cap N(B)))\}) \cup \{\{x, y\} \mid x \in N(A) \setminus N[B], y \in N[B] \setminus N(A)\}$ and let $I'$ be a maximum independent set in $G'$. Then,*

$$I = \begin{cases} I' \cup A & N(B) \setminus N[A] \in I' \\ I' \cup B & N(A) \setminus N[B] \in I' \end{cases}$$

*is a maximum independent set in $G$.*

*Proof.* Consider a *maximum independent set* $I$ in $G$ and without loss of generality let $A \subseteq I$ (by definition $A$ or $B \subseteq I$). Thus, $I \cap ((A \cup B \cup (N(A) \cap N(B))) = |A|$ and $I \cap (N(A) \setminus N[B]) = \emptyset$. Now let $I' = I \setminus A$. $I'$ is an *independent set* in $G'$, since each added edge (from $E' \setminus E$) is incident to a vertex from $N(A) \setminus N[B]$ and $|I'| = |I| - |A|$. This implies $\alpha(G') + |A| \geq \alpha(G)$.
Conversely, let $I'$ be a *maximum independent set* of $G'$. Obviously, $I'$ is also an *independent set* of $G$. Since vertices from $N(A) \setminus N[B]$ are pairwise adjacent to vertices form $N(B) \setminus N[A]$, $I'$ can only contain vertices from either $N(A) \setminus N[B]$ or $N(B) \setminus N[A]$. But then, $I = I' \cup A$ or $I = I' \cup B$ respectively is an *independent set* in $G$. Thus, $\alpha(G') + |A| \leq \alpha(G)$
In total $\alpha(G') + |A| = \alpha(G)$ and $I$ is a *maximum independent set* in $G$. □

Note that the *alternative reduction* adds new edges between existing vertices of the graph. For this reason, applying the *alternative reduction* is not beneficial in every case. To counteract this, the algorithm only uses the following special cases of the *alternative reduction*.

**Definition 6.** *(Funnel) In a Graph $G = (V, E)$ two adjacent vertices $u$ and $v$ are called funnels, if $G_{N(v)\setminus\{u\}}$ is a complete graph, i.e, if $N(v) \setminus \{u\}$ is a clique.*

**Theorem 8.** *(Funnel Reduction) In a Graph $G = (V, E)$ let $u$ and $v$ be funnels. Then, $\{u\}$ and $\{v\}$ are alternative sets.*

*Proof.* We have to show that there is a *maximum independent set* that contains either $v$ or $u$. So, consider a *maximum independent set* $I$ that excludes both $u$ and $v$. Then, $I$ has to include at least one vertex from $N(v) \setminus \{u\}$, because otherwise $I \cup \{v\}$ would be an *independent set* of larger size. On the other hand, $I$ can only contain at most one vertex $x$ from $N(v) \setminus \{u\}$, since $N(v) \setminus \{u\}$ is a clique. But then, $(I \setminus \{x\}) \cup \{v\}$ is an *independent set* of the same size as $I$ that does contain $v$. Thus $\{u\}$ and $\{v\}$ are alternative sets. □

**Definition 7.** *(Desk) In a Graph $G = (V, E)$ a cycle $u_1u_2u_3u_4$ of length four with no chords (i.e., an induced 4-cycle) is called a desk, if each of the vertices has at least degree three, $N(\{u_1, u_3\}) \cap N(\{u_2, u_4\}) = \emptyset$ and $|N(\{u_1, u_3\}) \setminus \{u_2, u_4\}| \leq 2$ as well as $|N(\{u_2, u_4\}) \setminus \{u_1, u_3\}| \leq 2$.*

**Theorem 9.** *(Desk Reduction) In a Graph $G = (V, E)$ let $u_1u_2u_3u_4$ be a desk. Then, $\{u_1, u_3\}$ and $\{u_2, u_4\}$ are alternative sets.*

*Proof.* Consider a *maximum independent set* $I$ of $G$. If $|I \cap \{u_1, u_2, u_3, u_4\}| > 1$, then clearly $I \cap \{u_1, u_2, u_3, u_4\}$ is either $\{u_1, u_3\}$ or $\{u_2, u_4\}$. Otherwise, without loss of generality $u_2, u_3, u_4 \notin I$ and $|I \cap N[\{u_1, u_3\}]| = 2$. The last equation holds because $|N(\{u_1, u_3\}) \setminus \{u_2, u_4\}| \leq 2$ by definition, and $u_1$ has at least one neighbor in $N(\{u_1, u_3\}) \setminus \{u_2, u_4\}$ ($d(u_1) \geq 3$). But then, $(I \setminus \{N(\{u_1, u_3\})\}) \cup \{u_1, u_3\}$ is an independent set of the same size as $I$ that does contain $\{u_1, u_3\}$. Thus, $\{u_1, u_3\}$ and $\{u_2, u_4\}$ are alternative sets. □

During kernelization, the algorithm searches for funnels or desks and reduces those structures according to the alternative reduction.

The algorithm also uses a reduction based on a solution to the LP-Relaxation of *maximum independent set.*

$$\text{maximize} \sum_{v \in V} x_v$$
$$0 \leq x_v \leq 1 \quad \forall v \in V$$
$$x_v + x_u \leq 1 \quad \forall \{u, v\} \in E$$

Nemhauser and Trotter showed that there always exists an optimal half integral solution to the LP-Relaxation, i.e., an optimal solution where $x_v \in \{0, \frac{1}{2}, 1\}$ for all $v \in V$ [15]. They also showed that given an optimal half integral solution to the LP-Relaxation, there is always a *maximum independent set* that includes all vertices $v$ with $x_v = 1$ and excludes all vertices $u$ with $x_u = 0$. Furthermore, they showed that finding an optimal half integral solution can be reduced to computing a *maximum matching* in a bipartite graph.
Iwata et al. developed an algorithm that given any optimal half integral solution constructs another half integral solution that minimizes the number of variable with half integral value [10].
The algorithm uses this solution to the LP-Relaxation to reduce the graph and also as an upper bound to an optimal solution.

Apart from reduction rules, the algorithm also uses branching rules that allow further reductions on branching when certain conditions hold. The first branching rule, mirror branching, was introduced by Fomin et al. [8]. According to Kneis et al. it is potentially useful, if the branching vertex has a rather low degree and thus, most likely has some mirror [12].

**Definition 8.** *(Mirror) In a graph $G = (V, E)$ a vertex $u$ is called a mirror of a vertex $v$, if $u \in N_2(v)$ and $G_{N(v) \setminus N(u)}$ is a (possibly empty) complete graph, i.e. $N(v) \setminus N(u)$ is a (possibly empty) clique. The set of all mirrors of $v$ is denoted by $\mathcal{M}(v)$ and $\mathcal{M}[v] := \mathcal{M}(v) \cup \{v\}$.*

**Theorem 10.** *(Mirror Branching) In a graph $G = (V, E)$, if there is no maximum independent set that contains a vertex $v$, then, every maximum independent set also excludes $\mathcal{M}[v]$.*

*Proof.* Consider any *maximum independent set* $I$. Then, $I$ has to contain at least two neighbors of $v$ because otherwise, we could get a *maximum independent set* $I' = (I \setminus N(v)) \cup \{v\}$ that includes $v$. Now let $u \in \mathcal{M}(v)$ be a mirror of $v$. Since $N(v) \setminus N(u)$ is a clique, $I$ can only contain at most one vertex from $N(v) \setminus N(u)$. Thus, $I$ contains at least another vertex from $N(v) \cap N(u)$ and therefore has to exclude $u$. □

So, when branching on a vertex $v$, the algorithm finds its mirrors $\mathcal{M}(v)$ and considers two possible cases. The first cases is that there is a *maximum independent set* that includes $v$ and therefore exclude $N(v)$. And the second case is that no *maximum independent set* includes $v$. In this case, the vertices from $\mathcal{M}(v)$ can also be discarded from the graph.

The packing branching rule by Akiba and Iwata [1] is a generalization of the idea behind the satellite branching rule by Kneis et al. [12]. The core idea behind those rules is that when branching in the case of excluding a vertex $v$ from the solution, one can assume that no *maximum independent set* contains $v$. Otherwise, if there is a *maximum independent set* that contains $v$, the algorithm finds it in the branch that includes $v$.
Based on the assumption that no *maximum independent set* includes a vertex $v$, constraints for the remaining vertices can be derived. For example, a *maximum independent set* that does not contain $v$ has to include at least two neighbors of $v$. The corresponding constraint is $\sum_{u \in N(v)} x_u \geq 2$, where $x_u$ is a binary variable that indicates whether a vertex is included in the current solution. The algorithm creates such constraints when branching, and updates them accordingly during the kernelization and branching steps. The constraints can then be used to reduce the graph or to prune the current branch when a constraint can not be fulfilled.

## 3 Related Work

This section discusses related work. It focuses on presenting branching strategies used by other branch and bound algorithms for *maximum independent set* or its equivalent problems *minimum vertex cover* and *maximum clique*.

The most common branching strategy used for *maximum independent set* and *minimum vertex cover* is branching on a vertex of maximum degree. Fomin et al. gave a theoretical analysis of this using the measure and conquer technique with a weighted degree sum as measure [8]. They showed that choosing a vertex of maximum degree that also minimizes the number of edges in its neighborhood is optimal with respect to their complexity measure. This greedy strategy is also used by the algorithm of Akiba and Iwata[1] and serves as a baseline for comparison in our experiments. Akiba and Iwata already compared this strategy with branching on a vertex of minimum degree and the strategy of choosing a branching vertex at random. Their experiments showed that those strategies are significantly worse than branching on maximum degree vertices.

Xiao and Nagamochi proposed a branch and reduce algorithm for *maximum independent set* that, in most cases, branches on a vertex of maximum degree but also uses a special edge branching strategy to handle dense subgraphs [19]. Edge branching is based on the principle of *alternative subsets* (like in alternative reduction). Given an edge $\{u, v\} \in E$ a *maximum independent set* can only contain $u$ or $v$ but not both of them. So, if there is a *maximum independet set* that includes $u$ or $v$, then $\{u\}$ and $\{v\}$ are alternative sets. Thus, branching on the edge $\{u, v\} \in E$ yields two cases. The first case is to remove both $u$ and $v$ and to search for a *maximum independent set* that does not include $u$ and $v$. The second case is to compute the alternative reduction of $\{u\}$ and $\{v\}$, i.e., to remove $\{u, v\} \cup (N(u) \cap N(v))$ and insert an edge $\{x, y\}$ between any nonadjacent vertices $x \in N(u) \setminus N(v)$ and $y \in N(v) \setminus N[u]$ and to search for a *maximum independent set* that includes either $u$ or $v$.
The algorithm by Xiao and Nagamochi uses edge branching in degree bounded graphs on edges $\{u, v\} \in E$, where $|N(u) \cap N(v)|$ is sufficiently large (the concrete values depend on the maximum degree of the graph).

Bourgeois et al. presented a *branch and bound* algorithm for *maximum independent set* that relies on fast algorithms for graphs with low average degree [2]. If the average degree of the graph is at most 4, they use specialized algorithms to solve the instance. Otherwise the algorithm branches on a vertex of maximum degree.

Chen, Kanj and Xia developed a branch and bound algorithm for the problem *minimum vertex cover* parameterized by the size $k$ of the *vertex cover*, i.e., the problem of finding a *vertex cover* of size at most $k$ [6]. In their algorithm, they use the concept of so called tuples and good pairs. A good pair is a pair of adjacent vertices that are advantageous for branching (the details are omitted here). A tuple is a set $S$ of vertices together with the number of vertices in $S$ that can be excluded from a *minimum vertex cover*. This information can be exploited during the branching to eliminate additional vertices. For example, consider the pair $(\{u, v\}, 1)$. We know that either $u$ or $v$ can be excluded from a *minimum vertex cover* and thus, if we include $u$ to the *vertex cover*, we can exclude $v$. Otherwise, if we exclude $u$ from the *vertex cover*, we can include $v$. Akiba and Iwata used the same idea in their *packing* reduction [1]. The algorithm by Chen, Kanj and Xia maintains a set of those structures as well as vertices of high degree and updates them accordingly during kernelization and branching. At each branching step the algorithm chooses the best structure and branches on it.

Most branch and reduce algorithms for *maximum clique* use some sort of greedy coloring to find an upper bound to the size of a *maximum clique* and also to reduce the number of possible vertices for branching. Given a coloring $c : V \to \mathbb{N}$ and the size $c_{\max}$ of a current best solution, it is easy to see that for $A = \{v \in V \mid c(v) \leq c_{\max}\}$, $G_A$ can not contain a clique larger than the current best solution. Thus, only vertices from $V \setminus A$ are considered for branching. More sophisticated algorithms use a MaxSAT encoding of *maximum clique* to achieve better upper bounds and to further reduce the set of branching vertices [14, 13].

A common strategy for choosing the branching vertex is to calculate a so called *degeneracy ordering* $v_1 < v_2 < \cdots < v_n$ where $v_i$ is a vertex of smallest degree in $G - \{v_1, \ldots, v_{i-1}\}$, and to choose the vertices for branching in descending order [4].

Li et al. introduced another vertex ordering for branching using *maximum independent sets* [13]. While $G$ is not empty, they repeatedly search for *maximum independent sets* and remove them from the graph. For two vertices $u$ and $v$, $u < v$ if $u$ has been removed later than $v$ or if $u$ and $v$ have been removed at the same time and $u < v$ in the *degeneracy ordering*.

## 4 Branching Strategies

## 5 Experimental Results and Conclusions

## 6 Future Work

## References

[1] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016.

[2] Nicolas Bourgeois, Bruno Escoffier, Vangelis Th. Paschos, and Johan M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012.

[3] Sergiy Butenko and Wilbert E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *Eur. J. Oper. Res.*, 173(1):1–17, 2006.

[4] Randy Carraghan and Panos M. Pardalos. An exact algorithm for the maximum clique problem. *Oper. Res. Lett.*, 9(6):375–382, November 1990.

[5] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[6] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

[7] Tammy M. K. Cheng, Yu-En Lu, Michele Vendruscolo, Pietro Liò, and Tom L. Blundell. Prediction by graph theoretic measures of structural effects in proteins arising from non-synonymous single nucleotide polymorphisms. *PLoS Computational Biology*, 4(7), 2008.

[8] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

[9] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[10] Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1749–1761. SIAM, 2014.

[11] Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed time-dependent contraction hierarchies. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 83–93. Springer, 2010.

[12] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A fine-grained analysis of a simple independent set algorithm. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 287–298. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[13] Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 939–946. IEEE Computer Society, 2013.

[14] Chu-Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Comput. Oper. Res.*, 84:1–15, 2017.

[15] George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975.

[16] Deepak Puthal, Surya Nepal, Cécile Paris, Rajiv Ranjan, and Jinjun Chen. Efficient algorithms for social network coverage and reach. In Barbara Carminati and Latifur Khan, editors, *2015 IEEE International Congress on Big Data, New York City, NY, USA, June 27 - July 2, 2015*, pages 467–474. IEEE Computer Society, 2015.

[17] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Trans. Graph.*, 27(5):144, 2008.

[18] Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.*, 469:92–104, 2013.

[19] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017.