1 Introduction

1.1 Motivation

The problem *maximum independent set* is a classic NP-complete graph problem[1] 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 [12], network analysis [11], rout planning [13] and computational biology [14, 15], 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 [2] 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 [2]. 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 [2] 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 [2]

```
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
7      if $G$ is not connected then
8          foreach connected component $G_i$ of $G$ do
9              $c \leftarrow c +$ Solve($G_i, 0, k - c$)   // Solve connected components independently
10         return $\min\{c, k\}$
11     $(G_1, c_1, k), (G_2, c_2, k) \leftarrow$ Branch($G, c$)   // Branch on a vertex $v$ into two subcases
12     $k \leftarrow \min\{k,$ Solve($G_1, c_1, k$)$\}$
13     $k \leftarrow \min\{k,$ Solve($G_2, c_2, k$)$\}$
14     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

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 [3]. They showed that choosing a vertex of maximum degree that minimizes the number of edges in his neighborhood for branching instead of any other vertex is optimal with respect to their complexity measure. This greedy strategy is also used by the algorithm of Akiba and Iwata[2] 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 [4]. 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 [5]. If the average degree of the graph is at most 4, they use specialized algorithms to solve the instance and 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 and otherwise, if we exclude u from the *vertex cover*, we can include v. Akiba and Iwata used the same idea in their *packing* reduction [2]. 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 [8, 9].

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 [12].

Li et al. introduced another vertex ordering for branching using *maximum independent sets* [9]. 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] Michael R. Garey, David. S. Johnson: Computers and Intractability A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, 1979.

[2] Takuya Akiba, Yoichi Iwata: Branch-and-reduce exponen-tial/FPT algorithms in practice: A case study of vertexcover. Theoretical Computer Science Volume 609 Part 1, January 4th 2016, pages: 211–225

[3] F.V. Fomin, F. Grandoni, D. Kratsch: A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM 56 (5), August 2009

[4] M. Xiao, H. Nagamochi: Exact algorithms for Maximum Independent Set. Algorithms and Computation ISAAC, December 2013, pages: 328–338

[5] N. Bourgeois, B. Escoffier, V. T. Paschos, J.M.M. van Rooij: Fast algorithms for max independent set. Algorithmica 62, 2012, pages: 382–415

[6] J. Chen, I. A. Kanj, G. Xia: Improved upper bounds for vertex cover. Theoretical Computer Science Volume 411 Issues 40-42, December 2010, pages: 3736-3756

[7] J. Kneis, A. Langer, P Rossmanith: A Fine-grained Analysis of a Simple Independent Set Algorithm. Foundations of Software Technology and Theoretical Computer Science, 2009, pages: 287–298

[8] C.-M. Li, H. Jiang, F. Manỳa On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. Computers & Operations Research Volume 84, August 2017, pages: 1-15

[9] C.-M. Li, Z. Fang, K. Xu: Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI, 2013, pages: 939-946

[10] R. Carraghan, P. M. Pardalos: An exact algorithm for the maximum clique problem. Operations Research Letters Volume 9 Issues 6, November 1990, pages: 375-382

[11] D. Puthal, S. Nepal, C. Paris, R. Ranjan, J. Chen: Efficient Algorithms for Social Network Coverage and Reach. IEEE BigData Congress, June 2015, pages 467–474

[12] P. V. Sander, D. Nehab, E. Chlamtac, H. Hoppe: Efficient Traversal of Mesh Edges using Adjacency Primitives. ACM Transactions on Graphics, Volume 27, 2008, Article No. 14

[13] T. Kieritz, D. Luxen, P. Sanders, C. Vetter. Distributed Time-Dependent Contraction Hierarchies. Experimental Algorithms, Volume 6049, 2010, pages: 83–93

[14] S. Butenko, W.E. Wilhelm: Clique-detection models in computational biochemistry and genomics. European Journal of Operational Research, Volume 173, Issue 1, August 2006, pages: 1 – 17

[15] T. M. K. Cheng, Y. Lu, M. Vendruscolo, P. Lio', T. L. Blundell: Prediction by Graph Theoretic Measures of Structural Effects in Proteins Arising from Non-Synonymous Single Nucleotide Polymorphisms. PLOS Computational Biology, 4(7), July 2008, pages: 1-9