# midterm report for CS260:
the vertex cover problem

Osayd Abdu (142461), Abdulelah Alneghaimish (159296),
Lukas Larisch (154273), Elaf Islam (142724)

## 1 Introduction

Our course project will cover the vertex cover problem, a problem included in Karp's list of 21 NP-complete problems [1, 2]. We will implement two algorithms for finding a minimal vertex cover on undirected graphs: The first algorithm with exponential running time uses a reduction from the max-CLIQUE problem [8], the second algorithm solves the problem with help of tree decompositions [4, 5, 6], which makes it possible to experimentally examine the constants hidded in Courcelle's theorem (1990):

**Theorem.** *Every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth.*

Hence, we will test our implementations on graphs of bounded treewidth and randomly generated partial $k$-trees graphs, as well on "named" graphs as included in the SageMath package [7].

## 2 Problem description

In the following we use standard graph terminology [3].

**Definition.** *Given a (undirected) graph $G = (V, E)$, a* vertex cover *$C$ of $G$ satisfies the following conditions:*

- $C \subseteq V(G)$
- $\{u, v\} \in E(G) \implies u \in C \vee v \in C$

*A* minimum vertex cover *is a vertex cover of minimum size among all vertex covers of $G$.*

This means that a vertex cover $C$ of a graph $G$ is a subset of the vertices of $G$, such that all edges of $G$ have an endpoint in $C$.

# 3 The algorithms

## 3.1 VERTEX COVER $\leq_p$ CLIQUE

We are going to use a reduction from the vertex cover problem to the maximum clique problem since both problems are NP-complete [8]: In order to solve the vertex cover problem on a graph $G$, we solve the clique problem on a graph $G'$ derived from $G$ first. We then convert the solution of the clique problem to a solution of the vertex cover problem.

**Definition.** *A* clique $C$ *of a graph* $G = (V, E)$ *satisfies the following conditions:*

- $C \subseteq V(G)$
- $u, v \in C \implies \{u, v\} \in E(G)$

*A* maximum clique *is a clique of maximum size among all cliques in* $G$*.*

In other words, a clique $C$ in a graph $G$ induces a complete subgraph in $G$.

**Definition** (polynomial reduction)**.** *Let* $L_1, L_2$ *be decision problems. We say that* $L_1$ *is reducible to* $L_1$ *in polynomial time, denoted* $L_2 \leq_p L_2$*, when there exists a deterministic algorithm* $M$*, such that* $x \in L_1 \iff M(x) \in L_2$*, for all* $w \in L_1$*.*

*This means that there exists a deterministic algorithm* $M$ *for converting a solutions of* $L_2$ *on the modified input of* $L_1$ *to a solution of* $L_1$*.*

**Lemma.** *VERTEX COVER* $\leq_p$ *CLIQUE.*

*Proof.* Let $\mathrm{VC}(G, k)$ be an instance of the vertex cover problem on a graph $G$. We state that the solution of $\mathrm{CLIQUE}(\bar{G}, |V(G)| - k)$, where $\bar{G}$ denotes the complement graph of $G$, is a solution of $\mathrm{VC}(G, k)$.

Let VC be a vertex cover of $G$, $|\,\mathrm{VC}\,| = k$. Then, $\{v, w\} \in E(G) \implies v \in \mathrm{VC} \vee w \in \mathrm{VC}$ f.a. $v, w \in V(G)$. In $\bar{G}$ this is $v, w, \notin \mathrm{VC} \implies \{v, w\} \notin E(G)$ f.a. $v, w \in V(G)$. I.e. $\{v, w\} \in E(\bar{G})$. Hence, $V(G) - \mathrm{VC}$ is a clique in $\bar{G}$.

Let CLIQUE be a clique in $\bar{G}$ of size $|V(G)| - k$. For $\{v, w\} \in E(G)$, we have $\{v, w\} \notin E(\bar{G})$, such that $v \notin \mathrm{CLIQUE} \vee w \notin \mathrm{CLIQUE}$, because CLIQUE is a clique. Therefore, $v \in V(G) - \mathrm{CLIQUE} \vee w \in V(G) - \mathrm{CLIQUE}$, causing $\{v, w\}$ to be covered by $V(G) - \mathrm{CLIQUE}$. □

## 3.2 Algorithm 1

The first algorithm solves the minimum vertex cover problem using a solution for the maximum clique problem.
The algorithm for solving the maximum clique problem in a graph $G$ that Östergård suggested in 2002 starts from a vertex $v_i$, $i = n, \dots, 1$, $n := |V(G)|$, and builds the largest clique that includes $v_i$ from the set of vertices $S_i = \{v_i, ..., v_n\}$. So, $S_n := \{v_n\}$ is considered first (clearly, $S_n$ is the largest clique itself). The algorithm then considers the largest clique in $S_{n-1}$ that contains $v_{n-1}$ and so on. The added advantage of this algorithm is the pruning strategy it uses to reduce the number of possible cliques.

## 3.3 Algorithm 2

The second algorithm solves the minimum vertex cover problem with help of tree decompositions. The algorithm is an extension of the approach for solving the VC-problem on trees, i.e. a two-phase dynamic programming algorithm that, now on a treedecomposition, first creates tables for the power set of the vertices contained in a bag. This is initially done for bags corresponding to leaves of a tree decomposition. Then, all vertex covers listed in the created tables will be extended, if possible, to vertex covers of a larger subgraph of the original graph in a bottom-up manner until the root of the tree decomposition is reached. In the second phase, an optimal vertex cover is computed by a top-down run beginning on the root of the tree decomposition which examines the computed tables until the leafs is reached.
The algorithm will work on *nice tree decompositions.* For such type of TD's it is sufficient to state four rules on how to compute the tables within the algorithm. These four distinct rules are applicable on different types of nodes of a nice tree decomposition. The algorithm has running time $\mathcal{O}(2^{tw(G)+1} \cdot n^{O(1)})$, hence is a polynomial time algorithm for graphs of bounded treewidth.

**Definition.** *A* tree decomposition *of a graph $G$ is a pair $(T, \beta)$, where $T$ is a tree and $\beta : V(T) \to \mathcal{P}(V(G))$ is a map such that*

*(T1) for every edge $e \in E(G)$ there is a node $t \in V(T)$ with $e \subseteq \beta(t)$, and*

*(T2) for all $v \in V(G)$ the set $\beta^{-1}(v) := \{t \in V(T) : v \in \beta(t)\}$ is non-empty and connected in $T$.*

*We refer to the sets $\beta(t)$ of a tree decomposition as* bags. *The width of a tree decomposition is it's maximal bag size minus 1. The* treewidth *of $G$,* $\mathrm{tw}(G)$, *is defined as the minimum width over all tree decompositions of $G$.*

**Definition.** *A tree decomposition $(T, \beta)$ of a graph $G$ is* nice, *if it satisfies the following conditions:*

- *$T$ is a rooted tree.*
- $d_T(t) \leq 2$ *f.a.* $t \in V(T)$.
- *Every node $t \in V(T)$ is of one of the following types:*

***leaf*** *$t$ is a leaf of $T$ and $|\beta(t)| = 1$.*
***introduce*** *$t$ has exactly one child $s$, $\beta(s) \subseteq \beta(t)$ and $|\beta(t)| = |\beta(s)| + 1$.*
***forget*** *$t$ has exactly one child, $\beta(t) \subseteq \beta(s)$ and $|\beta(s)| = |\beta(t)| + 1$.*
***join*** *$t$ has exactly two children $t_1, t_2$ and $\beta(t) = \beta(t_1) = \beta(t_2)$.*

**Lemma.** *Every graph $G$ with $V(G) \neq \emptyset$ has a nice tree decomposition of width* $\mathrm{tw}(G)$.

*sketch.* We apply the following transformations as long as possible. We will finally obtain a nice tree decomposition.

**Step 1:** Choose a root $r \in V(T)$ and orient all edges away from that root.

**Step 2:** If there is a node $t \in V(T)$ with more that two children, $\{t, s_1\}, \ldots, \{t, s_k\} \in E(T)$ with $k > 2$, perform the following transformation:

- $E(T) := E(T) \backslash \{\{t, s_i\}, i \in \{2, \ldots, k\}\}$.
- subdivide $\{t, s_1\}$ resulting in a new node $u$ and the edges $\{t, u\}, \{u, s_1\}$.
- $\beta(u) = \beta(t)$.
- $E(T) := \{\{u, s_i\}, i \in \{2, \ldots, k\}\}$.

**Step 3:** If there is a node $t$ with exactly 2 children $t_1, t_2$, such that $\beta(t) \neq \beta(t_1)$ or $\beta(t) \neq \beta(t_2)$, subdivide the edges $\{t, t_i\}, i \in \{1, 2\}$, resulting in two new nodes $u_1, u_2$. Set $\beta(u_i) = \beta(t), i \in \{1, 2\}$. Now, $t$ is an join node. We can procede with making the $u_i$'s forget or introduce nodes.

**Step 4:** Let $t$ be a node with exactly one child $s$. Let $\beta(t) \subseteq \beta(s)$ (the other case can be treated analogously). Let $M := \beta(s) \backslash \beta(t)$. If $|M| > 1$, then subdivide $\{t, s\}$, resulting in a new node $u$. Set $\beta(u) = \beta(t) \cup \{v\}$ for $v \in M$.

**Step 5:** Replace leaves by a sequence of introduce nodes: A leaf $t$ with $\beta(t) = \{v_1, \ldots, v_k\}$ is replaced by a path $P := (t_1, \ldots, t_k)$ with $\beta(t_i) := \{v_i, \ldots, v_k\}$. □

**Theorem.** *Let $G$ be a graph. There is an algorithm that, given a nice tree decomposition $(T, \beta)$ of $G$ of width $\operatorname{tw}(G) =: k$, computes a minimum vertex cover of $G$ in running time $\mathcal{O}(2^{k+1} \cdot n^{\mathcal{O}(1)})$.*

**proof.** *Let $t \in V(T)$. By $R_T(t)$, we denote the set of vertices that are reachable from $t$, including $t$. Let $\beta(S) := \cup_{t \in S} \beta(t)$ for $S \in V(T)$. For $t \in V(T)$, let $G[t] := G[\beta(R_T(t))]$.*

*For $t \in V(T)$ and $X \subseteq \beta(t)$, let $\mathrm{VC}_t(X)$ be the minimum size of a vertex cover $X'$ in $G[t]$ with $X' \cap \beta(t) = X$, if such a $X'$ exists. Otherwise, $\mathrm{VC}_t(X) = \infty$.*

*Using the propositions below, we can compute $\mathrm{VC}(X)$ f.a. $X \subseteq \beta(t), t \in V(T)$ in time $n^{\mathcal{O}(1)} \cdot 2^{k+1}$. The size of a minimum vertex cover of $G$ is $\min_{X \in \beta(r)} \mathrm{VC}(X)$, were $r \in V(T)$ is the root of $T$.*
*We now can construct a minimum vertex cover by tracing back through the tree decomposition.*

**Proposition.** ***leaf:*** *Let $t$ be a leaf of $T$ with $\beta(t) = \{v\}$. Then $\mathrm{VC}(\{v\}) = 1$ and $\mathrm{VC}(\emptyset) = 0$.* □

**Proposition.** ***forget:*** *Let $t$ be a forget node of $T$ with child $s$ and $\beta(s) \backslash \beta(t) = \{v\}$. Then f.a. $X \in \beta(t), \mathrm{VC}(X) = \min\{\mathrm{VC}(X), \mathrm{VC}(X \cup \{v\})\}$.*

*Proof.* Let $X'$ be a minimum vertex cover of $G[t] = G[s]$ with $X' \cap \beta(t) = X$. If $x \notin X'$ then $X' \cap \beta(s) = X$. Hence, $|X'| \geq \mathrm{VC}_s(X)$. If $x \in X'$, then $|X'| \geq \mathrm{VC}_s(X \cup \{v\})$ for the same reason.
Let $X_1, X_2$ be the vertex covers that determine $\mathrm{VC}_t(X), \mathrm{VC}_s(X \cup \{v\})$. Both are vertex covers of $G[t]$. Hence, $\mathrm{VC}_s(X) \leq \min\{|X_1, X_2|\}$. □

**Proposition.** ***introduce:*** *Let $t$ be an introduce node of $T$ with child $s$ and $\beta(t) \backslash \beta(s) = \{v\}$. Then f.a. $X \in \beta(t)$:*

*(1) If $X$ is not a vertex cover of $G[\beta(t)]$ then $\mathrm{VC}_t(X) = \infty$.*

*(2) If $X$ is a vertex cover of $G[\beta(t)]$ and $v \in X$ then $\mathrm{VC}_t(X) = \mathrm{VC}\, s(X \backslash \{v\}) + 1$.*

*(3) If $X$ is a vertex cover of $G[\beta(t)]$ and $v \notin X$ then $\mathrm{VC}(X) = \mathrm{VC}(X)$.*

*Proof.* (3): $N_{G[t]}(v) \subseteq \beta(s) \subset X$, since $X$ is a vertex cover of $G[\beta(t)]$. □

**Proposition.** ***join:*** *Let $t$ be a join node of $T$ with children $s_1, s_2$. Then f.a. $X \in \beta(t)$, $\mathrm{VC}_t(X) = \mathrm{VC}\, s_1(X) + \mathrm{VC}_{s_2}(X) - |X|$.*

*Proof.* If $X'$ is a vertex cover of $G[t]$ with $X' \cap \beta(t) = X$, then $X' \cap \beta(R_T(s_i)$ is a vertex cover of $G[s_i], i \in \{1, 2\}$. They share $|X|$ vertices. The vertex covers $X_i$ of $G[s_i]$ that are compatible with $X$ can be combined to a vertex cover of $G[t]$ of stated size.

□

## 4 Implementation & Experiments

Our code is implemented in C++ using boost graphs. We implemented a python interface in order to perform our experiments. For the second algorithm, we use TdLib for the provision of tree decompositions; it is currently the fastes solver for the tree decomposition problem [10] [11]. Running times for exact and approximative solvers are heavily varying. The asymtotical running time behavior on the input size will be reported in the next Section.

We evaluate our implementations on bounded treewidth graphs, partial k-trees, and named graphs included in the SageMath project. We do not consider graphs of treewidth greater than 30 as graphs of bounded treewidth. Hence, we do not include them in our tests although we report them.

### 4.1 Test graphs

We choose control flow graphs (CFGs) and partial $k$-trees as graphs of bounded treewidth [9].

The CFGs are derived from the C library, version 2.5 of the operating system Contiki and the operating system FUZIX [12]. A repository including all CFGs can be found on github [15].

| package | #graphs | avg/med./max #vert. | avg/med./max #edges | avg/med./max tw |
|---|---|---|---|---|
| stdlib | 142 | 37.47/23/544 | 39.45/24/609 | 1.85/2/4 |
| contiki | 1082 | 36.49/19/1452 | 38.08/19/1591 | 1.71/2/7 |
| fuzix | 529 | 42.00/24/587 | 45.04/24/668 | 1.97/2/6 |

Table 1: Statistics for the considered CFGs.

**Definition** (partial k-tree). *Let $k \in \mathcal{N}$. The class of* $k$-trees *is a graph class recursively defined as follows.*

- *$K_k$ is a k-tree.*
- *If $G$ is a k-tree and $X \subseteq V(G)$ is a k-clique in $G$, then the graph $G'$ obtained from $G$ by the following operations is a k-tree.*
  - *add a new vertex $v \notin V(G)$ to $G$.*
  - *add the edges $\{\{v, w\} : w \in X\}$ to $G$.*

*A* partial $k$-tree *is a subgraph of a k-tree.*

**Lemma.** *Let $G$ be a partial k-tree. Then* $\text{tw}(G) \leq k$.

We consider several partial $k$-trees of different size and several $k$'s for our experiments. More precisely, we generated random partial $k$-trees for $k \in \{1, \ldots, 15\}$ with $n \in \{50, 100, 200, 250, 500\}$ vertices and deleted each edge with probability $p \in \{.97, .95, .90, .80, .70\}$. We generated 5 different graphs for each $(k, n, p)$.

| $n$ | avg/med./max #edges | avg/med./max tw |
|---|---|---|
| 50 | 47.70/28/196 | 3.57/2/15 |
| 100 | 102.48/60/455 | 4.40/3/15 |
| 200 | 210.58/120/922 | 5.09/4/15 |
| 250 | 264.48/153/1116 | 5.27/4/15 |
| 500 | 537.26/309/2260 | 5.90/5/15 |

Table 2: Statistics for the considered $k$-trees.

Finally, the "named" graphs are derived from the graph database of Sage-Math and can be found on github [7], [14]. Statistics on graphs included in this set that we do not consider for our experiments are stated in the appendix.

| #graphs | avg/med./max #vert. | avg/med./max #edges | avg/med./max tw |
|---|---|---|---|
| 125 | 83.05/25/3282 | 166.63/60/6561 | 9.79/7/29 |

Table 3: Statistics for the named graphs.

### 4.2 Results

## 5 Summary

## References

[1] Karp, R. M. (1972). Reducibility Among Combinatorial Problems.. In R. E. Miller & J. W. Thatcher (eds.), Complexity of Computer Computations (p./pp. 85-103), : Plenum Press, New York. ISBN: 0-306-30707-3

[2] The vertex cover problem, `https://en.wikipedia.org/wiki/Vertex_cover`

[3] Diestel, R. (2005). Graph Theory. Springer-Verlag Heidelberg, New York.

[4] H. L. Bodlaender, A tourist guide through treewidth, Acta Cybernetica

[5] Neil Robertson and P.D Seymour (1991), Graph minors. I. Excluding a forest, Journal of Combinatorial Theory, Series B, pp 39 - 61

[6] Stefan Arnborg (1985), Efficient Algorithms for Combinatorial Problems with Bounded Decomposability, BIT, pp 2–23

[7] SageMath, `http://www.sagemath.org/`

[8] Patric R.J. Östergård (2002), A fast algorithm for the maximum clique problem. Discrete Applied Mathematics

[9] Krause P., Larisch L. (2017), The treewidth of C, manuscript

[10] Larisch L, TdLib, `https://github.com/freetdi/tdlib`

[11] The PACE 2017 challenge, `https://pacechallenge.wordpress.com/pace-2017/track-a-treewidth/`

[12] Fuzix, `https://github.com/EtchedPixels/FUZIX`

[13] Adam Dunkels and Björn Grönvall and Thiemo Voigt (2004): Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensorss Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)

[14] Named graphs, `https://github.com/freetdi/named-graphs`

[15] Control flow graphs, `https://github.com/freetdi/named-graphs`

## 6 Appendix

| name | $\lvert V\rvert$ | $\lvert E\rvert$ | tw |
|---|---|---|---|
| BrouwerHaemersGraph | 81 | 810 | 54 |
| CameronGraph | 231 | 3465 | 177 |
| Cell120 | 600 | 1200 | 112 |
| DejterGraph | 112 | 336 | 42 |
| FoldedCubeGraph_7 | 64 | 224 | 31 |
| GossetGraph | 56 | 756 | 44 |
| HallJankoGraph | 100 | 1800 | 87 |
| HigmanSimsGraph | 100 | 1100 | 77 |
| HyperStarGraph_10_5 | 252 | 630 | 84 |
| JohnsonGraph_10_4 | 210 | 2520 | 140 |
| KneserGraph_10_2 | 45 | 630 | 35 |
| KneserGraph_8_3 | 56 | 280 | 34 |
| M22Graph | 77 | 616 | 55 |
| NonisotropicUnitaryPolarGraph_3_3 | 63 | 1008 | 55 |
| OddGraph_5 | 126 | 315 | 48 |
| PasechnikGraph_2 | 49 | 441 | 38 |
| PasechnikGraph_3 | 121 | 3025 | 104 |
| SimsGewirtzGraph | 56 | 280 | 36 |
| SquaredSkewHadamardMatrixGraph_2 | 49 | 588 | 41 |
| SquaredSkewHadamardMatrixGraph_3 | 121 | 3630 | 109 |
| SwitchedSquaredSkewHadamardMatrixGraph_2 | 50 | 525 | 40 |
| SwitchedSquaredSkewHadamardMatrixGraph_3 | 122 | 3355 | 109 |
| SymplecticDualPolarGraph_4_4 | 85 | 850 | 64 |
| SymplecticPolarGraph_4_4 | 85 | 850 | 64 |
| T2starGeneralizedQuadrangleGraph_4 | 64 | 576 | 47 |

Table 4: Named graphs of tw $> 30$ that we did not consider for our experiments.