Seminar Paper: On Fixed-Parameter Tractability and Kernelization

Alexander Salostowitz
RWTH Aachen - Computer Science Bsc. - 5.Semester
alexander.salostowitz@rwth-aachen.de

Tutor: Benedikt Pago
pago@logic.rwth-aachen.de

January 14, 2022

1. Fixed-Parameter Tractability

In complexity theory our initial interest is to understand how "hard" a problem is, furthermore if there can be an efficient algorithm that solves it. Efficiency or "tractability" was initially not well defined, so the evolution of complexity theory gave rise to the complexity classes **P** and **NP**. The former class became identified with the notion of tractability [6]. As it stand for now, the question of **P** vs. **NP** is not settled, so we are still confronted with the hardness of **NP**-complete problems, which we assume "harder" than problems that can be solved in polynomial time for this paper. I follows that for **NP**-complete problems, there is something that causes exponential blowup and does not permit polynomial run time. Unfortunately, the class does not give us any information about the property or component of a certain problem that causes it. The first step is to find an appropriate parameterization of the problem.

Definition 1.1. *Let Σ be a finite alphabet.*
(1) A parameterization of Σ^* is a polynomial time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$
(2) A parameterized problem is a pair (Q, κ) with $Q \subseteq \Sigma^*$ and a parameterization κ of Σ^*

If we can find a parameterization, such that the parameter encapsulates the exponentiality of the problem, then the problem is fixed-parameter tractable.

Definition 1.2. *Let Σ be a finite alphabet and κ a parameterization.*
(1) An algorithm is an fixed-parameter tractable algorithm with respect to κ if there are computable functions $f, f' : \mathbb{N} \to \mathbb{N}$ and a polynomial $p(X)$, $p'(X)$ such that for every $x \in \Sigma^*$ the running time is at most:

$$f(\kappa(x)) \cdot p(|x|)$$

(2) A parameterized problem (Q, κ) is fixed-parameter tractable if there is an fpt-algorithm with respect to κ that decides Q

Fixed-parameter tractability offers a more refined framework to approach hard problems by capturing the core of the computational complexity with a parameter. As a matter of fact, the definition give above corresponds to strongly uniform fixed-parameter tractability [4]. It is not important to go into detail about the intricacies, because the definition given here characterizes a robust class of **FPT** [5] that will suffice for the rest of this paper. The main focus of this paper will be exploring various **FPT** techniques and applying them to a well known **NP**-complete problem. The aim is to gain an overview of powerful approaches to parameterized complexity. First of all, the notion of parameterization needs to be clarified, as well as the surrounding theory of **FPT**. If not specifically stated all definitions and theorems can be found in chapter 1 and 9 of [9] for more in-depth information.

1.1. The role of parameterization

It is important to see that the choice of parameterization is crucial. One could use the input size as parameterization, so that every **NP**-complete with that parameterization becomes **FPT**. Of course, this does not bring anything new to the table. In principle we want the parameter k to be small in comparison to the input size n of a problem. For many **NP**-complete problems there are "natural" parameterizations. For example for Vertex-Cover we can use the parameter k as the size of the vertex cover we want to find [5]. Chapter 5 of [13] goes into detail about the pitfalls of parameterization.

1.2. Theory of FPT

We established before that **FPT** is the class of all fixed-parameter decision problems. There is a hardness theory, similar to the theory of **NP**-hardness. The necessary concept of fpt-reductions can be found in [5]. The theory is based on the Weft hierarchy of complexity classes **FPT** $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[P], where all inclusions are believed to be strict [3]. This theory gives strong evidence that some parameterized problems are not in **FPT**. For example, Independent-Set is complete for W[1] [5]. This shows us that some **NP**-complete problems may be hard in a way that can not be captured by fixed-parameter tractability.

1.3. FPT problems and techniques

We want to determine if a **NP**-hard problems is in **FPT**, furthermore we want to find a fast algorithm to solve it. In principle every polynomial algorithm we already know can serve us. Some of the most useful are Shortest-Path, Maximum-Matching and Maximum-Flow. Over the years many powerful techniques were found and continue to be found. Here are some of them roughly categorized into problem domains together with some known **FPT** problems with natural parameterization.

Minimization problems: p-Vertex-Cover, p-Hitting-Set, p-Dominating-Set,...
Techniques: Bounded Search Trees [9], Iterative Compression [14], Linear programming[9], Graph Minors [11], Tree-width[9].
Maximization problems: k-Path, p-Disjoint-Triangles,...
Techniques: Color coding[9], Matroids [12].

This overview should serve as a reference to the techniques. The following section applies the technique of Bounded Search Tree to the **NP**-complete problem Vertex-Cover with natural parameterization. This problem will be the sample problem that is used throughout the paper to show and apply certain **FPT** techniques.

1.3.1. Bounded Search Tree for Vertex-Cover

p-VERTEX-COVER	
Instance:	An undirected graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter:	k
Problem:	Decide if there is a subset $S \subseteq V$ with $\|S\| \leq k$ such that for all $\{u, v\} \in V$: $u \in S$ or $v \in S$.

The Algorithm works as follows: Construct a binary tree of height k and label the root with the graph $\mathbf{G}$. For an edge $\{u, v\} \in E$ mark/delete either v or u. Create child nodes with the resulting graphs for both options, so one child for $\mathbf{G} - \{u\}$ and one child for $\mathbf{G} - \{v\}$. The set of vertices labeling a node represents what still needs to be covered for a possible vertex cover. This is repeated recursively until height k. If there is a child node with no edges remaining, we have found a vertex cover with k vertices. The constructed tree has at most 2^k vertices and so the running time is therefore $O(2^k|V|)$. This assures that p-Vertex-Cover is in **FPT** [4].

From now on we will cover the **FPT** technique of Kernelization. It can be used as a preprocessing step before combining it with other techniques.

2. Kernelization

Kernelizations are powerful many-one reductions that transform parameterized problems instances into instances whose size is bounded by the parameter. It can therefore be seen as a preliminar non-branching preprocessing step.

Definition 2.1. *Let Σ be a finite alphabet, (Q, κ) a parameterized problem and h a computable function. A Kernelization is a polynomial time computable function $K : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$:*

$$x \in Q \Leftrightarrow K(x) \in Q \quad \textit{and} \quad |K(x)| \leq h(\kappa(x))$$

Intuitively, we extract the core or as we call it here the kernel of a problem. This kernel is expected to be smaller than the initial problem, it is in fact bounded by the parameter, which we already expect to be smaller than the problem size. The definition may allow kernels that are bigger than the problem itself, but this does not make practical sense, the kernel of a problem can not become bigger than the problem itself.

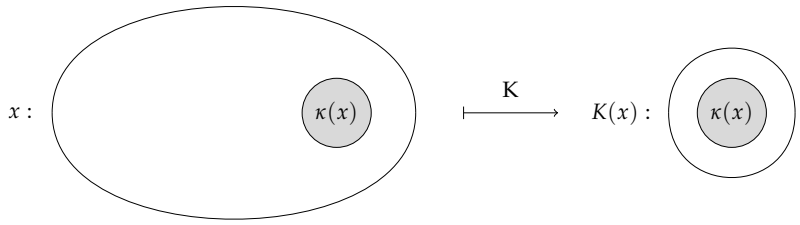


Figure 1: Kernel of a problem

Nevertheless there is a good reason to keep the definition as loose as it is. We can see that Kernelization and **FPT** may be tightly connected trough the parameter. In fact, both notions are equivalent, as the following theorem proves.

Theorem 2.1. *For every parameterized problem (Q, κ) the following are equivalent.*
(1) (Q, κ) in **FPT**
(2) Q is decidable and (Q, κ) has a Kernelization.

Proof. $(2) \Rightarrow (1)$: Let K be a Kernelizations of (Q, κ). Fpt-algorithm that decides (Q, κ) : Input $x \in \Sigma^*$, compute $K(x)$ in polynomial time and use any decision algorithm f to decide $K(x)$. Because $|K(x)| \leq h(\kappa(x))$, the running time of f is bounded in terms of the parameter $\kappa(x)$.
$(1) \Rightarrow (2)$: Let A be an fpt-algorithm solving (Q, κ) in time $f(\kappa) \cdot p(n)$ for some computable function f and polynomial $p(X)$. Without loss of generality $p(n) \geq n$ for all $n \in \mathbb{N}$. If $Q = \emptyset$ or $Q = \Sigma^*$, choose the trivial Kernelization to ϵ. Otherwise fix canonical yes and no instances $x_0 \in Q$, $x_1 \notin Q$. Now compute Kernelization K for (Q, κ) : for input $x \in \Sigma^*$ with $n = |x|$ and $k = \kappa(x)$ simulate A for $p(n)^2$ steps. If A stops and accepts (rejects) then output $x_0(x_1)$. Otherwise we know $n \leq p(n) \leq f(k)$, because $p(n) \cdot p(n) \leq p(n) \cdot f(k)$ and so output x. K can be computed in polynomial time, $|K(x)| \leq |x_0| + |x_1| + f(k)$ and $x \in Q$ if and only if $K(x) \in Q$. □

Kernelization is a useful paradigm for not only proving that a problem is in **FPT**, but also designing efficient fpt-algorithms.

2.1. Kernelization Rules

Kernelizations are reductions that can be broken down into particular steps, which we may call reduction rules. The notion of rules is widely used in the realm of Kernelization and can be formalized like in [8]. The framework of rules offers a clean way to structure a reduction and clearly differentiate between the various properties we exploit in a problem. For a parameterized problem (Q, k) a reduction rule is a function K_i that is defined like a Kernelization. It is computable in polynomial time in the input x and $(x, k) \in (Q, k)$ if and only if $K_i((x, k)) = (x', k') \in (Q, k)$. A Kernelization is finite set of reduction rules $K = \{K_1, ..., K_n\}$ and the kernel $K(x, k)$ is the instance after exhaustive application of the reduction rules [8]. K is computed in polynomial time, because every reduction rule is. Importantly equivalence is preserved trough transitivity, because every reduction rule preserves equivalence of the problem. This property is also called safety of a rule. We will see the application of Kernelization rules in the following example Kernelization of Vertex-Cover.

2.1.1. A simple kernel for Vertex-Cover

Proposition 2.1. *p-Vertex-Cover has a polynomial Kernelization that for an instance $(\mathbf{G}, k)$ with $\mathbf{G} = (V, E)$ and $k \in \mathbb{N}_{\geq 1}$ computes in time $O(k + ||\mathbf{G}||)$ an instance $(\mathbf{G}', k')$ such that:*

$$|V'| \leq 2(k')^2 \quad \textit{and} \quad |E'| \leq (k')^2$$

The following Kernelization is called Buss' Kernelization [2].

Proof. Fix canonical YES instance $(\mathbf{G}^+, 1)$ with only one single vertex and NO instance $(\mathbf{G}^-, 1)$ consisting of two disjoint edges. Let $\mathbf{G} = (V, E)$ and $k \in \mathbb{N}$ the input parameter.
Rule 1: Remove any isolated vertices, the can not be part of a vertex cover.
Rule 2: If $v \in V$ has degree $\deg(v) > k$, then v must be contained in any vertex cover of size at most k. So we continue with $(\mathbf{G} - \{v\}, k-1)$.
Rule 3: If $\deg(\mathbf{G}) \leq k$ and $\mathbf{G}$ has a vertex cover of k elements, then $\mathbf{G}$ has at most k^2 edges, else return $(\mathbf{G}^-, 1)$.
Rule 4: If $k = 0$ and there are edges left in $\mathbf{G}$, then return $(\mathbf{G}^-, 1)$, else return $(\mathbf{G}^+, 1)$.
Apply the rules until no longer possible. We either achieve a canonical instance with constant size or an instance $(\mathbf{G}', k')$, with the property that $\mathbf{G}'$ has at most $(k')^2$ edges and hence at most $2(k')^2$ vertices. The reduction can be done in linear time of the input by computing the degree of every vertex and updating the list every time a vertex is deleted.

□

We can now use a brute force algorithm to solve the reduced instance or use a better algorithm like Bounded Search Tree. In that case we can achieve an overall algorithm for p-Vertex-Cover running in time $O(2^k k^2 + |E| + |V|)$.

2.2. Problems with polynomial kernel size

From the Vertex-Cover example it can be seen that aside from efficient computability a "good" Kernelization should also have a small kernel size. In practice small means it has polynomial size in terms of the parameter. This makes sense, because the kernel causes the exponential blowup, while the kernel itself is found in polynomial time. This gives rise to the "convenient" class of problems that have polynomial Kernelization.

Definition 2.2. *A Kernelization K of a parameterized problem (Q, κ) is polynomial if there is a polynomial $p(X)$ such that $|K(x)| \leq p(\kappa(x))$ for all $x \in \Sigma^*$.*

In addition we want our Kernelization to not increase the parameter ($\kappa(K(x)) \leq \kappa(x)$). It would be beneficial if we could find polynomial Kernelization for all problems, but not all problems in **FPT** admit polynomial kernels. For example, let $Q \subseteq \Sigma^*$ be an **EXP**-complete problem with $\kappa(x) := \lceil loglog|x| \rceil$, then (Q, κ) is in **FPT**. If it had a polynomial Kernelization, then Q would be decidable in polynomial time. We can find the kernel in polynomial time and use an exponential algorithm on the kernel, which would run in polynomial time in reference to the initial input, because of the chosen parameterization. This is of course a contradiction.

An in-depth look at the classification of polynomial size and exponential-size kernels can be found in [8]. It is mostly an open question which structural can be made from the (non-)existence of kernels of particular size for various problems. Furthermore it is of interest which bounds can be achieved regarding kernel size and Kernelization time. [1] offers a framework for showing that a wide range of problems do not have polynomial kernels. It also introduces a generic lower-bound engine that a variety of **FPT** problems cannot have polynomial kernels unless the polynomial hierarchy collapses.

3. A linear kernel for Vertex-Cover

We now want to improve the previous simple Vertex-Cover Kernelization to a linear size kernel. We make use of the connection between matchings and vertex covers, the maximum cardinality of a matching is a lower bound for the minimum cardinality of a vertex cover. In addition we use the fact that Maximum-Matchings in bipartite graphs can be computed efficiently. Proofs taken from [9].

Definition 3.1. *A matching of a graph $\mathbf{G} = (V, E)$ is a set $M \subseteq E$ of pairwise disjoint edges.*
(1) A matching M is maximal if there is no matching M', such that $M \subset M'$.
(2) A matching M is maximum if there is no matching M', such that $|M| < |M'|$.

Definition 3.2. *Let $\mathbf{G} = (V, E)$ be a graph with a matching M.*
(1) An M-alternating path is a path, such that the edges alternately belong to M and $E \setminus M$.
(2) A vertex $v \in V$ is free with respect to M if it is not incident to an edge in M.
(3) An M-augmenting path is an M-alternating path whose endpoints are both free.

Lemma 3.1. *Let M be a matching in graph $\mathbf{G}$.*
M is maximum if and only if there is no M-augmenting path.

Proof. If M is maximum there can not be an M-augmenting path, because otherwise we could flip the edges of such a path and get a matching with greater cardinality. If M is not maximum, then there is an M-augmenting path. Let M' be a maximum matching and consider the subgraph of $\mathbf{G}$ induced by $M \cup M'$, including the edges that appear in $M \cap M'$ twice. Since every vertex is adjacent to at most one edge in M or M', all connected components in the subgraph are paths or cycles that are alternating with respect to M and M'. There is an M-augmenting path whose first and last edges are in M', since $|M| < |M'|$. □

Lemma 3.2. *Let $\mathbf{G}$ be a graph , $\mathbf{B}$ be a bipartite graph and $k \in \mathbb{N}$.*
(1) There is linear time algorithm that computes a maximal matching for $\mathbf{G}$.
(2) There is an algorithm running in time $O(k \cdot n)$ that decides if there is a matching of size k in $\mathbf{B}$.

Proof. (1) can be computed with a simple greedy algorithm.
(2) For the bipartite graph $\mathbf{B} = (V_1, V_2, E)$ start with a maximal matching M and increase it by repeatedly flipping the edges of an augmenting path. Such an augmenting path can be found in linear time, if the matching M is not maximum. Let $\mathbf{G}$ be the directed graph obtained from $\mathbf{B}$:
Add source s and sink t. Add directed edges from s to all free vertices in V_1 and directed edges from all free vertices in V_2 to t. Direct all edges in M from V_2 to V_1 and all edges from $E \setminus M$ from V_1 to V_2. For every M-augmenting path P, sPt is a path from s to t, conversely, for every path P' from s to t, $P' \setminus \{s, t\}$ is an M-augmenting path in $\mathbf{B}$. Finding an M-augmenting path in $\mathbf{B}$ is thus equivalent with finding an $s - t$ path in $\mathbf{G}$, which can be done in linear time. □

Definition 3.3. *Let $\mathbf{G} = (V, E)$ be a graph. A crown in $\mathbf{G}$ is a bipartite graph $\mathbf{C} = (I, S, E')$ such that:*
(1) I is an independent set of $\mathbf{G}$ and $S = S(I) := \{v \in V \mid \exists u \in I : \{v, u\} \in E\}$.
(2) E' is the set of edges between I and S in $\mathbf{G}$, thus $(I \cup S, E')$ is a subgraph of $\mathbf{G}$.
(3) $\mathbf{C}$ has a matching of cardinality $|S| =: val(\mathbf{C})$, which denotes the value of the crown.

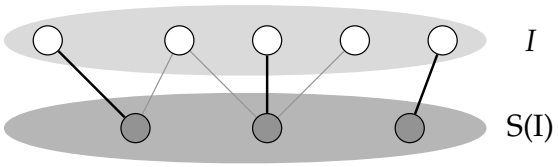


Figure 2: A crown of size 3

For every graph $\mathbf{G} = \{V, E\}$ denote the minimum cardinality of a vertex cover by $vc(\mathbf{G})$. For $\mathbf{H} = (W, F)$ denote $\mathbf{G} \setminus \mathbf{H}$ as the induced subgraph $\mathbf{G}$ with vertex set $V \setminus W$.

Lemma 3.3. *Let $\mathbf{G}$ be a graph and $\mathbf{C}$ a crown in $\mathbf{G}$, then:*

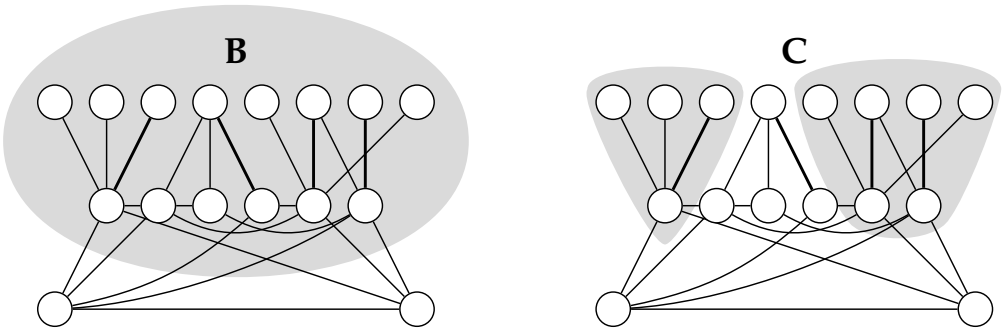
$$vc(\mathbf{G}) = vc(\mathbf{G} \setminus \mathbf{C}) + val(\mathbf{C})$$

Proof. Let $\mathbf{G} = (V, E)$ and suppose $\mathbf{C} = (I, S(I), E')$, with $C := I \cup S(I)$, then $val(\mathbf{C}) = |S(I)|$. We can observe that $S(I)$ is a minimum vertex cover of $\mathbf{C}$, because it covers all edges of $\mathbf{C}$ and to cover all edges of a matching of $\mathbf{C}$ of cardinality $|S(I)|$, at least $|S(I)|$ vertices are needed. Let X be a minimum vertex cover of $\mathbf{G}$. Then $X \cap C$ is a vertex cover of $\mathbf{C}$ and $X \setminus C$ is a vertex cover of $\mathbf{G} \setminus \mathbf{C}$. It follows $vc(\mathbf{G}) = |X| = |X \setminus C| + |X \cap C| \geq vc(\mathbf{G} \setminus \mathbf{C}) + |S(I)|$.
Conversely, let X' be a minimum vertex cover of $\mathbf{G} \setminus \mathbf{C}$, then $X := X' \cup S(I)$ is a vertex cover of $\mathbf{G}$ and it follows: $vc(\mathbf{G}) \leq |X| \leq |X'| + |S(I)| = vc(\mathbf{G} \setminus \mathbf{C}) + |S(I)|$ □

We can transform an instance $(\mathbf{G}, k)$ of p-Vertex Cover into an equivalent smaller instance $(\mathbf{G}', k')$ by deleting a crown $\mathbf{C}$ from $\mathbf{G}$ and reducing $k' = k - val(\mathbf{C})$. This fact follows from the Lemma 3.3 above.

Theorem 3.4. *p-Vertex-Cover has a polynomial Kernelization that for an instance $(\mathbf{G}, k)$ with $\mathbf{G} = (V, E)$ computes in time $O(k \cdot ||\mathbf{G}||)$ an instance $(\mathbf{G}', k')$, such that $|V'| \leq 3k$.*

Proof. For the Kernelization let the input $\mathbf{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. First compute a maximal matching L of $\mathbf{G}$. If $|L| > k$, then $vc(\mathbf{G}) \geq |L| > k$, so return the trivial NO-instance $(\mathbf{G}^-, 1)$. We now assume $|L| \leq k$ and let I be the vertices that are free with respect to L. By the maximality of L, I is an independent set of $\mathbf{G}$. If $|I| \leq k$, then $|V| = 2|L| + |I| \leq 3k$, so return the input instance $(\mathbf{G}, k)$. In the following we assume that $|I| > k$ and construct a crown $\mathbf{C}$ of $\mathbf{G}$ with at least $|V| - 3k$ vertices. Let $\mathbf{B} = (I, S, E')$ be the bipartite graph with $B := I \cup S(I)$ and whose edge set consists of all edges between I and $S(I)$. The Kernelization algorithm now computes $\mathbf{B}$ and a maximum matching M of $\mathbf{B}$. If $|M| > k$, then $vc(\mathbf{G}) \geq |M| > k$, so return the trivial NO-instance $(\mathbf{G}^-, 1)$. Otherwise, if $|M| \leq k$ and $|M| = |S(I)|$, then $\mathbf{B}$ is a crown. In this case the crown $\mathbf{C} := \mathbf{B}$ has at least $|V| - 2|L| \geq |V| - 2k$ vertices. In the following, assume that $|M| \leq k$ and $|M| < |S(I)|$. There is at least one vertex in I that is free with respect to M, because $|I| > k$. Let J be the set of all free vertices with respect to M in I. Let C be the set of all vertices in $\mathbf{B}$ that can be reached by an M-alternating path from a vertex in J. Let $\mathbf{C}$ be the induced subgraph of $\mathbf{B}$ with vertex set C. Let $I' := C \cap I$.



(a) Bipartite graph $\mathbf{B}$ with matching M (b) The crown $\mathbf{C}$

Claim 1: $S(I') = C \cap S(I)$.
Proof: Clearly $S(J) \subseteq C \cap S(I)$. Let $v \in I' \setminus J = (C \cap I) \setminus J$. We prove that $S(v) \subseteq C$. By the definition of C, the vertex v is reachable from a vertex in J by an alternating path P. The last edge e of P is in M, because the first edge is not and the length of P is even. Let $w \in S(v)$. If $\{v, w\} = e$, then w is reachable by the alternating path $P \setminus v$. Otherwise, $\{v, w\} \notin M$ and w is reachable by the alternating path Pw. This proves that $S(I') \subseteq C \cap S(I)$. To prove $C \cap S \subseteq S(I')$, let $v \in C \cap S(I)$. Let P be an M-alternating path from a vertex in J to v and let $w \in I$ be the predecessor of v on this path, then $w \in C \cap I = I'$ and thus $v \in S(I')$.

Claim 2: $\mathbf{C}$ is a crown.
Proof: Conditions (1) and (2) of Definition 3.3 are satisfied with $I := I'$ by claim 1. For condition (3), let M' the intersection of M with the edge set of $\mathbf{C}$. For contradiction suppose $|M'| < |S(I')|$. Then there exists a vertex $v \in S(I')$ that is free with respect to M'. There exists an M-alternating path P from a vertex in J to v whose last edge is not in M, since $v \in S(I') \subseteq C$.The vertex v is free with respect to M, because if $\{v, w\} \in M$ for some w, then Pw would be an M-alternating path. Hence $w \in C$ and therefore $\{v, w\} \in M'$. Furthermore all vertices in J are free with respect to M. Thus the path P is actually M-augmenting, this contradicts M being a maximum matching of $\mathbf{B}$ by Lemma 3.1.
To compute the Crown $\mathbf{C}$ a similar approach as in Lemma 3.2 can be used to compute augmenting paths.
Let $c := val(\mathbf{C}) = |M'|$

Claim 3: $|C| \geq |V| - 3k + 2c$
Proof: C consists of the vertices in J and the $2c$ endpoints of the vertices in M'. $|V| = |I| + 2|L| = |J| + |M| + 2|L|$, and thus $|J| = |V| - 2|L| - |M| \geq |V| - 3k$

Let $\mathbf{G}' := \mathbf{G} \setminus \mathbf{C}$ and $k' := k - c$, then by Lemma 3.3 $\mathbf{G}$ has a vertex cover of size k if and only if $\mathbf{G}'$ has a vertex cover of size k'. For $\mathbf{G}'$ the number of vertices is $|V \setminus C| \leq 3k - 2$. If $k' = 0$ the Kernelization returns either the trivial YES-instance $(\mathbf{G}^+, 1)$ if $\mathbf{G}'$ has no edges or the trivial NO-instance $(\mathbf{G}^-, 1)$ if there is an edge left. Otherwise returns $(\mathbf{G}', k')$. The running time is: $O(||\mathbf{G}||) + O(k \cdot ||\mathbf{B}||) + O(||\mathbf{B}||) \leq O(k \cdot ||\mathbf{G}||)$. $O(||\mathbf{G}||)$ for computing the maximal matching L, independent set I and bipartite graph $\mathbf{B}$. $O(k \cdot ||\mathbf{B}||)$ for computing the maximum matching M. $O(||\mathbf{B}||)$ for computing $\mathbf{C}$. □

The technique of crown reductions can be used as a general technique for many other graph problems [8]. In this case it already results in a linear kernel for Vertex-Cover, but we can find a better kernel with the next technique.

3.1. Kernelization through Linear Programming

The last Kernelization technique that will be covered is Linear Programming, it is a powerful technique that can lead us to an even smaller kernel of vertex cover.

Definition 3.4. *A linear program consists of:*
(1) n variables $x_1, ..., x_n$.
(2) m linear inequalities in these variables.
(3) a linear objective function (minimization, maximization).

We first need to state an important theorem.

Theorem 3.5. *There is a polynomial time algorithm that computes an optimal solution for a given linear program with rational coefficients.*

Proof. See [7]. □

Theorem 3.6. *p-Vertex-Cover has a Kernelization that for an instance* (G, k) *with* $G = (V, E)$ *computes in polynomial time an instance* (G', k')*, such that* $|V'| \leq 2k'$.

Proof. See [9]. □

We will now look at the basic idea but not give a full proof. The formulation of the minimum vertex cover problem as a linear program goes as follows. Let $G = (V, E)$ be a graph and construct the linear program $L(G)$:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \text{subject to} \\ x_v + x_w \geq 1, & & \text{for all } \{v, w\} \in E, \\ x_v \geq 0, & & \text{for all } v \in V, \\ x_v \leq 1, & & \text{for all } v \in V. \end{array}$$

Integral solutions of $L(G)$ naturally correspond to vertex covers of G. It can now be shown that $L(G)$ has an optimal solution that is half-integral ($x_v \in \{0, \frac{1}{2}, 1\}$ for all $v \in V$) that can be computed in polynomial time, by using a certain transformation.

Lemma 3.7. *Let* $G = (V, E)$ *be a graph and* $(x_v)_{v \in V}$ *an optimal half-integral solution of* $L(G)$*. For* $r \in \{0, \frac{1}{2}, 1\}$*, let* $V_r := \{v \in V \mid x_v = r\}$ *and let* G_r *be the induced subgraph of G with vertex set* V_r*. Then:*
(1) $vc(G_{\frac{1}{2}}) \geq |V_{\frac{1}{2}}| \setminus 2$
(2) $vc(G_{\frac{1}{2}}) = vc(G) - |V_1|$

Proof. See [9] □

For an instance (G, k) of p-Vertex-Cover, let V_r and G_r be defined like in Lemma 3.7 and let $k' := k - |V_1|$. If $k' < 0$, then by Lemma 3.7 G does not have a vertex cover of k elements, return trivial NO-instance. If $k' = 0$ return trivial NO-instance if $G_{\frac{1}{2}}$ has at least one edge and trivial YES-instance otherwise. If $k' > 0$ and $|V_{\frac{1}{2}}| > 2k'$, then by Lemma 3.7 $G_{\frac{1}{2}}$ does not have a vertex cover of k' elements, return trivial NO-instance. Otherwise return $(G_{\frac{1}{2}}, k')$.

Linear programming can again be used as a general Kernelization technique. It can arguably be seen as more powerful than Crown reductions, because it can be used in more areas, aside from graphs. Concerning p-Vertex-Cover, it is one of the most well studied **FPT** problems and as such we may already have found the smallest possible kernel size for it.

Theorem 3.8. *p-Vertex-Cover has no problem kernel formed by a graph with 1.36k vertices unless* $\mathbf{P} = \mathbf{NP}$.

Proof. See [8]. □

4. Research and open problems

FPT is a field of many open questions. For one there is always the question which problems are in **FPT** or equivalently have a Kernelization and which techniques that are found work most effectively. Furthermore there is a ever growing list of problems that admit a polynomial kernel, as well as questions how it can be shown that a problem does not have a polynomial kernel. The paradigm of Kernelization offers the possibility of preprocessing, so running a Kernelization before any other algorithm. It is not trivially clear if this can be done with every algorithm, for example it is not clear how to combine Kernelization with approximation. A kernel of a problem does not have to have the same approximation properties as the original problem. Research is been done to combine both "worlds" and find an overarching framework of parameterized optimization problems, that encompasses parameterized algorithms, approximation algorithms and Kernelization [10].

References

[1] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[2] J. F. Buss and J. Goldsmith. Nondeterminism within p. In C. Choffrut and M. Jantzen, editors, *STACS 91*, pages 348–359, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[3] R. de Haan and S. Szeider. The parameterized complexity of reasoning problems beyond NP. *CoRR*, abs/1312.1672, 2013.

[4] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In P. Clote and J. B. Remmel, editors, *Feasible Mathematics II*, pages 219–244, Boston, MA, 1995. Birkhäuser Boston.

[5] R. G. Downey and M. R. Fellows. Parameterized complexity. In *Monographs in Computer Science*, 1999.

[6] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[7] L. L. S. A. Grötschel, Martin. *Geometric algorithms and combinatorial optimization*. Springer, 1988.

[8] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, Mar. 2007.

[9] F. Jorg and M. Grohe. *Parameterized complexity theory*. Springer, 2006.

[10] D. Lokshtanov, F. Panolan, M. S. Ramanujan, and S. Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 224–237, New York, NY, USA, 2017. Association for Computing Machinery.

[11] L. Lovász. Graph minor theory. *Bull. New Ser. Am. Math. Soc.*, 43(1):75–86, Oct. 2005.

[12] D. L. Neel and N. A. Neudauer. Matroids you have known. *Mathematics Magazine*, 82(1):26–41, 2009.

[13] R. Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, 2008.

[14] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.