Turing Kernelization for finding long paths and cycles

Hu, R.

Award date:
2019

TECHNISCHE UNIVERSITEIT EINDHOVEN

MASTER THESIS

Turing Kernelization for finding long paths and cycles

Author:
Rui Hu

Supervisor:
Dr. Bart M.P. Jansen

Graduation Committee:
Bart M.P. Jansen
Hans L. Bodlaender
Kevin Verbeek

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Mathematics and Computer Science

August 15, 2018

TECHNISCHE UNIVERSITEIT EINDHOVEN

Abstract

Department of Mathematics and Computer Science

Master of Science

Turing Kernelization for finding long paths and cycles

by Rui Hu

In computer science, an NP-hard problem is regarded as 'so hard' that no algorithm can solve it quickly. In order to improve the efficiency, we would like to pre-process the problem. If the preprocessing guarantees that it can reduce the size of the problem, as well as maintain the answer to the problem unchanged, then we call such preprocessing as 'kernelization' and we say such problem admits a 'kernel'. If the answer to the problem has size k, we desire its size to be reduced to some polynomial in k, after the kernelization.

As preprocessing, we want to create a smaller instance quickly without changing its answer. However this seems impossible for some problems, since the existence of these kernelizations violates some commonly believed complexity-theoretic assumptions. As an alternative, we try a preprocessing in a different way: we first split the instance of the hard problem into small pieces, with each piece of size $poly(k)$. We can solve each piece separately (and quickly), and solve the complete problem from the combination of answers to all pieces. This is formalized in the notion of Turing kernelization.

In this thesis we work on designing Turing Kernelization for two problems, namely the k-cycle problem and the k-path problem. Following Jansen's Decompose-Query-Reduce framework, we achieve a polynomial Turing Kernel for the k-cycle problem where every piece has size $O(k^2)$, as well as a polynomial Turing Kernel for the k-path problem where every piece has size $poly(k)$. We also show an idea that may improve the size of pieces to $O(k)$ for the k-cycle problem.

Contents

Chapter 1

Introduction

1.1 Background

In computer science, a NP-hard problem means it is so hard that no algorithm can solve it quickly. The difficulty of an instance of a NP-hard problem is not just determined by its total size n, but can also depend heavily on other aspects such as the solution size or the amount of structure that is presents in the instance. To develop exact algorithms that are probably efficient on inputs that are large but not difficult, we use *parameterized complexity*. We call a decision problem as a *parameterized problem* if a parameterization is chosen. That is, for every instance of the decision problem, a non-negative integer called *parameter* is given, and such parameter determines the complexity of the instance in some way.

A parameterized problem is formed by a question $\mathcal{Q}$ and a parameter k. Instead of finding all elements satisfying the question $\mathcal{Q}$, in parameterized problems we only need to find whether there exist at least k elements that fulfilling $\mathcal{Q}$. A parameterized problem is indeed faster to be solved, as we can at least 'brute force' try every set of elements of size k and see whether this set satisfies $\mathcal{Q}$, in $O(n^k)$ time, where n is the total size of elements contained in $\mathcal{Q}$[1]. Compared with the fact that a NP-hardness problem has a superpolynomial running time in n, we can maintain an 'almost' polynomial running time to answer a parameterized problem, if we can maintain a small value of k.

However, we want the problem to be solved even more efficiently. Therefore we consider doing some preprocessing so as to reduce the size of input n of the problem. If a preprocessing guarantees that it can reduce any input size n to some polynomial in parameter k, meanwhile maintain the answer to the parameterized problem unchanged, then we call this preprocessing as a *polynomial kernelization*, and we say that the parameterized problem admits a *polynomial kernel*.

As preprocessing, we desire a kernelization creating smaller instances from parameterized problems without changing its answer as fast as polynomial time. However this seems to be impossible for some problems, since the existence of these kernelizations violates some commonly believed complexity-theoretic assumptions.[23] As an alternative, we try a preprocessing in a different way: we first split the instance of the hard problem into small pieces whose size is in polynomial of k. We can solve each piece separately (and quickly), and solve the complete problem from the combination of answers to all pieces. This is formalized in the notion of Turing kernelization.

[1] We also call n as the size of input of question $\mathcal{Q}$

1.2 Research questions

In this thesis we research on how to design Turing kernelizations for two parameterized problems, the k-cycle problem and the k-path problem, as they are announced not likely to admit regular polynomial kernelizations [26].

In the following we introduce these two problems:

- The k-cycle problem:

  Input: A graph G with n vertices

  Parameter: An integer k

  Question: Does G contain a cycle of length at least k?

- The k-path problem:

  Input: A graph G with n vertices

  Parameter: An integer k

  Question: Does G contain a path of length at least k?

Both of them are NP-complete problems [6]. As variations of the Hamiltonian cycle problem, both problems are still hard even in planar graphs [10]. Besides, both problems are known in fixed parameter tractable complex class [13].

1.3 Our contribution

In this thesis, we split an instance of the problems into pieces, by transforming an input planar graph into its quasi-4-tree decomposition. As every piece is almost quasi-4-connected, we use the fact that a quasi-4-connected graph is assured to contain a long cycle, if it has too many vertices, so as to give an upper bound of the size of each piece. We managed to achieve a polynomial Turing Kernel for the k-cycle problem where every piece has size $O(k^2)$ with the help of a self-reduction algorithm. It is noticeable that our results are only available on planar graphs, as several theorems applied are only available on planar graphs.

We finally come to an idea that may improve the size of pieces to $O(k)$ for the k-cycle problem.

1.4 Related work

The first systematic work on parameterized problems is done by Downey and Fellows in 1999[9]. Another widely accepted definition was given by Flum and Grohe in 2006[12]. Along with that quite a lot of problems are found admit polynomial kernelizations [19, 20, 21, 14, 22]. Meanwhile research also shows that, some other problems admits a polynomial kernel, only if in complexity theory it satisfies $NP \subseteq coNP/poly$, which means these problems are impossible to have polynomial kernelizations under current widely accepted assumptions[8, 4, 2]. Designing polynomial kernels for these problems slowed down during years.

People started to work on Turing kernelization since more than five year ago. In the beginning, Bodlaender et al. asked that, what will happen if we are not restricted to reduce the size by designing a kernel for the whole problem, but by dividing the problem into pieces? [5] Fernau et.al soon showed such concept is available for designing polynomial (Turing) kernels for the K-LEAF OUT-TREE problem while that problem does not admit a polynomial kernel unless $NP \subseteq coNP/poly$ [11].

As it is nature to ask whether the similar trick can be used to design polynomial Turing kernelizations for the k-cycle problem and the k-path problem, Jansen showed polynomial Turing Kernelizations for both questions [18] in 2017. His research suggests that we may design a Turing Kernelization of smaller size for both problems, if we can split a (planar) graph into strongly-connected pieces. As he used the theorem that, every graph has a tri-connected Tutte decomposition [24], he reached the result that the k-cycle problem admits a Turing Kernel of size $O(k^{2.58})$. It is noticeable that such theorem has been developed over 40 years ago. After months of publication of Jansen's research, it published another research shows that every graph has a quasi-4-tree decomposition in polynomial time [15]. As quasi-4-connected pieces are even more strongly connected, we come to this thesis to find a better Turing Kernelization.

1.5 Organization

In Chapter 2 we give preliminaries on parameterized complexity, graph theory and Turing Kernel. In Chapter 3 we present a polynomial Turing Kernel for the k-cycle problem and the k-path problems in Chapter 4. In Chapter 5 we are going to give a hopeful method in finding Turing Kernel for the k-cycle problem that which has even smaller size pieces. We will give a short conclusion for this report as well.

Chapter 2

Preliminaries

2.1 Parameterized complexity and kernels

Let X be a set and n be a non-negative integer. We use $\binom{X}{n}$ denote the collection of subsets of X of size n. Let Σ be a finite alphabet. A parameterized problem $\mathcal{Q}$ is a language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$. The second component $k \in \mathbb{N}$ is called the *parameter*. A parameterized problem is *fixed-parameter tractable* if problem 'Is $(x,k) \in \mathcal{Q}$?' can be determined in time $O(f(k) \cdot |x|^{O(1)})$, where f is a computable function that only depends on k.

Definition 1. *[1]* A *polynomial kernelization* algorithm for a parameterized problem $\mathcal{Q}$ is an algorithm A, that transforms inputs (x,k) of $\mathcal{Q}$ to inputs (x',k') of $\mathcal{Q}$, such that:

1. the algorithm uses time polynomial in $|x| + k$;
2. the algorithm transforms inputs to equivalent inputs: $(x,k) \in \mathcal{Q} \iff A(x,k) \in \mathcal{Q}$;
3. $k' \leq k$
4. $|x'| \leq f(k)$ for some function f: the value of the new parameter and the size of the new input are bounded by a function of the value of the old parameter

When used without adjective, the term kernel used in this article refers to a regular polynomial kernelization defined as Definition 1.

2.2 Turing kernelization

Now we are going to discuss about Turing Kernel. A Turing kernel is a kernel that splits an instance of a problem into pieces of size $poly(k)$, and solve every piece with the help of an oracle that can gives YES/NO answer to some problem $\mathcal{Q}'$ in constant time. Furthermore, the answer to the complete problem can be obtained from combining answers to these pieces. More formally,

Definition 2. Let $\mathcal{Q}$ be a parameterized problem and let $f := \mathbb{N} \to \mathbb{N}$ be a computable function. A *Turing Kernelization* for $\mathcal{Q}$ of size f is an algorithm that decides whether a given instance $(x,k) \in \Sigma^* \times \mathbb{N}$ is contained in $\mathcal{Q}$ in time polynomial in $|x| + k$, when given access to an oracle that decides membership in $\mathcal{Q}$ for any instance (x',k') with $|x'|, k' \leq f(k)$ in a single step.

Note that in the remaining part of this thesis we call the maximum value of $|x'|$, i.e., the maximum size of instances that sent to the oracle, as the *size* of a Turing Kernelization.

2.3 Graphs

An undirected graph G is described by a vertex set $V(G)$ and an edge set $E(G)$, where every edge consists of two different vertices from $V(G)$. We use $G[X]$ to express the induced subgraph obtained from G with the vertex set $X \subseteq V(G)$, and we write $H \subseteq G$, if G contains H as a subgraph. We use $G - X$ to express the subgraph induced by $V(G) \setminus X$. The *neighborhood* of a vertex v in graph G is the set of vertices X, such that for every $x \in X, \{x, v\} \in E(G)$, which is remarked as $N_G(v)$. We write directly $N(v)$ if there is only one graph in the context. The *degree* of a vertex is $deg(v) = |N_G(v)|$.

Graph H is a *minor* of graph G if H can be obtained from a subgraph of G by contracting edges. H is a *topological minor* of G if it can be obtained from a subgraph of G by recursively replacing a degree-2 vertex by an edge between its neighbors.

For any connected graph G, vertex set $W \subseteq V(G)$ is called *n-separator* if $G - W$ is disconnected and $|W| = n$. A *minimal separator* in a connected graph G is a vertex set $W \subseteq V(G)$, such that $G - S$ is disconnected and for any $S' \subset S$, $G - S'$ is connected. Graph G is *3-connected* if it has more than 3 vertices, and is remaining connected if less than 3 vertices are removed. G is *essentially 4-connected*[1] if G is 3-connected, and after removing any 3-separator W, one component of $G - W$ is a single vertex. A separation of a graph G is a pair (A, B) of subsets of $V(G)$ such that $A \cup B = V(G)$ there is no edge between $A - A \cap B$ and $B - A \cap B$. The *order* of a separation is $|A \cap B|$.

A *walk* in graph G is a sequence of vertices $v_1, v_2, .., v_k$ where $\{v_i, v_{i+1}\} \in E(G)$ for every $i \in [1, k-1]$. A *path* is a walk that in which every vertex is unique. An *xy-walk* is a walk such that $v_1 = x, v_k = y$. Similarly, a *xy-path* is a path that $v_1 = x, v_k = y$. Vertices x, y are *endpoints* of xy-walks/paths, and an x-path is a path that includes x as an endpoint.

The *length* of a path is its number of edges. The *interior vertices* of a path $v_1, v_2, ..., v_{k-1}, v_k$ are $v_2, ..., v_{k-1}$. A *cycle* is a v_1v_k-path, where $v_1 = v_k$ and its *length* is its number of edges.

2.4 Tree Decompositions

In this thesis we split a graph G into small pieces based on its Quasi-4-tree decomposition, therefore we need to briefly introduce the technique and some facts about tree decompositions.

Definition 3 ([3]). A tree decomposition of $G = (V, E)$ is a pair $(T, \{\mathcal{X}_i \mid i \in I\})$, where $\{\mathcal{X}_i \mid i \in I\}$ is a collection of subsets of V and $T = (I, E)$ is a tree such that:

1. $\bigcup_{i \in I} \mathcal{X}_i = V$
2. for all $\{u, w\} \in E$, there exists an $i \in I$ such that $u, w \in \mathcal{X}_i$, and
3. for all $i, j, k \in I$ with j on the path in T from i to k we have $\mathcal{X}_i \cap \mathcal{X}_k \subseteq \mathcal{X}_j$

[1] Also called quasi-4-connected

The *treewidth* of a tree decomposition $(\mathcal{X}, T)$ is $\max_{i\in I}|\mathcal{X}_i| - 1$ and the *treewidth* of a graph G is the minimal treewidth over all tree decompositions of it. A tree T is a *rooted tree* if some node $i \in V(T)$ does not have a parent. In a rooted tree T, we write $\mathcal{X}(T[j])$ as the bags of a subtree of T that rooted at $\mathcal{X}_j$. The *adhesion* of a tree decomposition is $\max_{\{i,j\}\in E(T)}|\mathcal{X}_i \cap \mathcal{X}_j|$ and if a tree T has no edges we regard the adhesion of T as 0. In the remaining part of this thesis we use the phrase 'the adhesion of $\mathcal{X}_i$ and $\mathcal{X}_j$' to express $|\mathcal{X}_i \cap \mathcal{X}_j|$, when $\mathcal{X}_i$ and $\mathcal{X}_j$ are two connected bags of a tree T. The *torso* of a bag $\mathcal{X}_i$ is a graph $\text{TORSO}(G, \mathcal{X}_i)$ obtained from $G[\mathcal{X}_i]$, by adding an edge between every pair of vertices $p, q \in \mathcal{X}_i$, if there is a pq-path in G, whose internal vertices do not belong to $\mathcal{X}_i$.

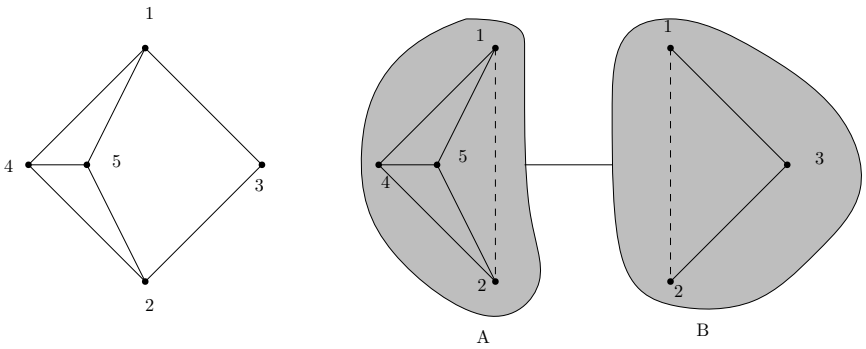


FIGURE 2.1: Consider a tree decomposition $(T, \{A, B\})$of a simple graph with 5 vertices. Note that for both bag A and B, we need to add an edge between vertex 1 and 2 in graph $\text{Torso}(G, A)$ and $\text{Torso}(G, B)$, as $\{1, 2\}$ is an adhesion and for each bag, there exists 12-path whose interior vertices are not covered by that bag.

2.5 Quasi-4-tree decomposition

In the thesis we use the fact that every graph has a quasi-4-tree decomposition and therefore we introduce the idea of quasi-4-decomposition.

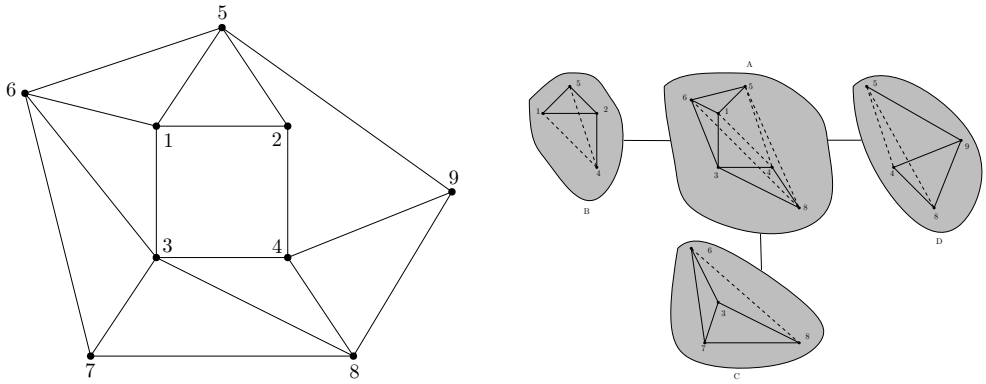
Theorem 2.5.1 ([15]). Every graph G has a tree decomposition $(T, \mathcal{X})$ of adhesion at most 3 such that for all tree nodes $\mathcal{X}_i$ the torso $\text{TORSO}(G, \mathcal{X}_i)$ is a minor of G that is either quasi-4-connected or a complete graph of order at most 4, and such that for each edge $i, j \in E(T)$ the set $X(i) \cap X(j)$ is a minimal separator in G.
Furthermore, this decomposition can be computed in cubic time.

By theorem 2.5.1 we can split a graph G into some quasi-4-connected pieces in polynomial time. Specifically, if G is a planar quasi-4-connected graph, it preserves the following theorem:

Theorem 2.5.2 ([16]). Let G be a quasi-4-connected planar graph on $n \geq 11$ vertices and C be a longest cycle of G, then $|V(C)| \geq \frac{1}{2}(n + 4)$.

Using theorem 2.5.2 we may find an upper bound of the size of every bag of the tree decomposition of G, such that if any bag has more vertices than such upper bound we can announce G has a k-cycle directly.
Finally, we also need the following lemmas in this thesis:

Lemma 2.5.3. Let $(T, \mathcal{X})$ be a tree decomposition of a graph G. Let node j be a child of node i and $\mathcal{X}(i) \cap \mathcal{X}(j) = \{x, y, z\}$. If $u \in \mathcal{X}(T[j])\backslash\mathcal{X}(i)$, then the connected component C of $G - \{x, y, z\}$ that contains u satisfies $V(C) \cap \mathcal{X}(i) = \emptyset$.



(a) An example of a graph G

(b)Example of quasi-4-tree decomposition of G

FIGURE 2.4: (a) shows an example of a graph G. Such graph has vertex set as $\{1, ..9\}$ and an edge set. (b) shows a quasi-4-tree decomposition of G. Edges in dashed lines are added in the torso. Observe the torso is either a complete graph of order at most 5 (node b, c, d), or is quasi-4-connected (node a).

Proof. We prove the claim by showing that for every $v \in \mathcal{X}(i)$, $v \notin V(C)$. Let u $\in \mathcal{X}(T[j])\backslash\mathcal{X}(i)$, and pick an arbitrary vertex $v \in \mathcal{X}(i)$. If $v \in \{x, y, z\}$ then it is trivial that $v \notin V(C)$ since $C \subseteq G - \{x, y, z\}$. Next we assume that $v \in \mathcal{X}(i)\backslash\{x, y, z\}$, note that $v \notin \mathcal{X}(T[j])$ since it is not an element of the adhesion. Assume that both $u, v \in V(C)$, then there is a uv-path $\mathcal{P}$ in C since C is a connected component and every edge in $\mathcal{P}$ must be covered by some node of the tree decomposition. Since the two ends of $\mathcal{P}$ belong to $\mathcal{X}(i)\backslash\mathcal{X}(T[j])$ and $\mathcal{X}(T[j])\backslash\mathcal{X}(i)$, then we can always find two vertices p, q on $\mathcal{P}$ such that p is the last vertex on $\mathcal{P}$ belonging to $\mathcal{X}(T[j])\backslash\mathcal{X}(i)$, and q is the successor of p on $\mathcal{P}$. Apparently edge $\{p, q\}$ cannot be covered by neither node i nor $T[j]$, since if this is the case, either p or q will appear in both i and j, i.e., is an adhesion of i and j and hence either p or q belongs to $\{x, y, z\}$, which contradicts to $C \subseteq G - \{x, y, z\}$.

Therefore it shows that if $v \in \mathcal{X}(i)\backslash\{x, y, z\}$, $v \notin V(C)$. Since $\mathcal{X}(i) = (\mathcal{X}(i)\backslash\{x, y, z\}) \cup \{x, y, z\}$, then the claim is proved. □

Lemma 2.5.4. Let $(T, \mathcal{X})$ be a quasi-4-tree decomposition of a graph G, and let $i, j \in V(T)$ satisfies that $\mathcal{X}_j$ is a child of $\mathcal{X}_i$. Then we have:

1. If $\{x, y\}$ is a minimal separator of G and $\mathcal{X}(i) \cap \mathcal{X}(j) = \{x, y\}$, then edge $\{x, y\}$ contained in Torso$(G, \mathcal{X}(i))$.
2. If $\{x, y, z\}$ is a minimal separator of G and $\mathcal{X}(i) \cap \mathcal{X}(j) = \{x, y, z\}$ and if there is a vertex $u \in \mathcal{X}(T[j])\backslash\mathcal{X}(i)$ then there exists a triangle $\triangle xyz$ contained in Torso$(G, \mathcal{X}(i))$.

Proof. Jansen has given a proof for the first point in his research on a 3-connected graph. [18] Fortunately, his proof still works in our case, since a quasi-4-connected graph is at least three-connected. Here we give a short version of his proof for completeness.

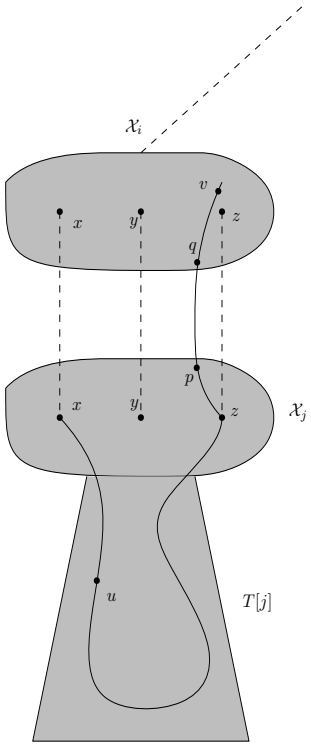


FIGURE 2.5: Illustration of a contradiction introduced in lemma 2.5.3. If a connected component passes through a vertex $u \in \mathcal{X}(T[j]) \setminus \mathcal{X}(i)$ and a vertex $v \in \mathcal{X}(i)$ then this component passes through these two bags, and hence there must be an edge $\{p, q\}$ that cannot be covered by any bags correctly.

If $\{x, y\} \in E(G)$ then the lemma is trivial. If this is not the case, and if $\{x, y\}$ is a minimal separator of G, then there exist at least two connected component C_1 and C_2 of $G - \{x, y\}$ that are adjacent to both x, y. Now assume $v_1 \in \mathcal{X}(i) \cap V(C_1)$ and $v_2 \in \mathcal{X}(i) \cap V(C_2)$, since $\mathcal{X}(i)$ is quasi-4-connected, by Menger's Theorem there are three internally vertex-disjoint v_1v_2-paths in TORSO$(G, \mathcal{X}(i))$, implies there is a v_1v_2-path passes through neither x nor y. Besides, since torso is a topological minor of G, we can restore a v_1v_2-path in G without passes through x, y as x, y is already in Torso$(G, \mathcal{X}(i))$, which contradicts that $\{x, y\}$ is a minimal separator of G.

Since there exist an xy-path $\mathcal{P}_1$ whose interior vertices belong to C_1 and an xy-path $\mathcal{P}_2$ whose interior vertices belong to C_2, and $\mathcal{X}(i)$ only contains vertex from one component, edge $\{x, y\}$ is contained in Torso$(G, \mathcal{X}(i))$ by definition of torso.

Now we focus on the case of separator of three. We show such triangle exists by showing edge $\{x, y\}$, $\{y, z\}$, $\{x, z\} \in$Torso$(G, \mathcal{X}(i))$.

First, if $\{x, y\}$, $\{y, z\}$, $\{x, z\} \in E(G)$ then the lemma is trivial. Now we assume one of them is missing, say edge $\{x, y\}$. Similarly, since $\{x, y, z\}$ is a minimal separator there exist some (at least 2) connected components such that each component is adjacent to x, y, z. Consequently, for every connected component, there exists at least one xy-path whose interior vertices only belong to that component. Since $u \in \mathcal{X}(T[j]) \setminus \mathcal{X}(i)$ and u belongs to some connected component C in $G \setminus \{x, y, z\}$, then by lemma 2.5.3 $V(C) \cap \mathcal{X}(i) = \emptyset$.

Hence there exist at least one component C has no intersections with $\mathcal{X}_i$. Consequently, there exists at least one xy-path whose interior vertices belong to C, and hence, not belong to $\mathcal{X}_i$, which shows by the definition of torso that $\text{Torso}(G, \mathcal{X}(i)$ has edge $\{x, y\}$.

For a similar reason, if $\{x, z\}, \{y, z\} \notin E(G)$, there is also a $xz(yz)$-path whose interior vertices not in $\mathcal{X}_i$ and hences there exist edges $\{x, z\}, \{y, z\}$ in $\text{Torso}(G, \mathcal{X}(i)$. Therefore three edges $\{x, y\}, \{y, z\}, \{x, z\}$ are in $\text{Torso}(G, \mathcal{X}(i)$ as $\triangle xyz$ exists, which concludes the lemma. □

Chapter 3

Turing Kernelization for finding cycles

In this section we show a polynomial Turing kernelization, such that solves the k-cycle problem with the help of an oracle that answers the existence of a k-cycle in an instance (G, k), with k is a parameter. We guarantee that the size of the kernel is bounded by $O(k^2)$. In section section 3.1 we discuss the properties of cycles and we give the construction of the kernel in section 3.2.

3.1 Properties of cycles

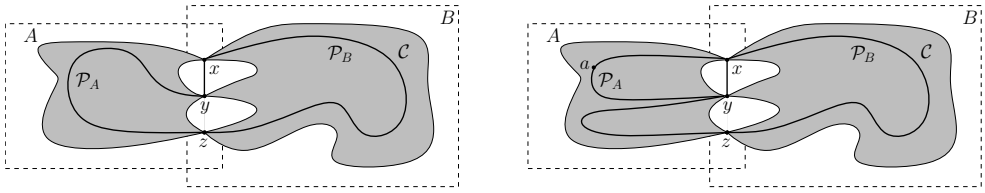
We show some properties of k-cycles in this section. As a k-cycle passes through at least k edges, the following lemma shows that when finding a k-cycle in a separation of order three, we only need to maintain few maximum-length paths in one part of the separation and safely remove other vertices from that part, meanwhile keeping the existence of the k-cycle unchanged.

Lemma 3.1.1. Let A, $B \subseteq V(G)$ be a separation of order three of a graph G with $A \cap B = \{x, y, z\}$. Let $\mathcal{P}_1, ...\mathcal{P}_6$ be subgraphs of $G[A]$ such that:

1. $\mathcal{P}_1$ is a maximum-length xy-path in $G[A]$
2. $\mathcal{P}_2$ is a maximum-length xz-path in $G[A]$
3. $\mathcal{P}_3$ is a maximum-length yz-path in $G[A]$
4. $\mathcal{P}_4$ is a maximum-length xy-path in $G[A]\backslash\{z\}$
5. $\mathcal{P}_5$ is a maximum-length xz-path in $G[A]\backslash\{y\}$
6. $\mathcal{P}_6$ is a maximum-length yz-path in $G[A]\backslash\{x\}$

If G has a k-cycle, then $G[A]$ has a k-cycle or $G[(\bigcup_{i=1}^{6} V(\mathcal{P}_i) \cup B)]$ has a k-cycle.

Proof. Consider a k-cycle $\mathcal{C}$ in G. If $V(\mathcal{C}) \subseteq A$, then we find such k-cycle in $G[A]$ and we are done. Similarly, if $V(\mathcal{C}) \subseteq B$ then the graph $G[(\bigcup_{i=1}^{6} V(\mathcal{P}_i) \cup B)]$ contains the k-cycle $\mathcal{C}$ and we are also done. Now we assume that $\mathcal{C}$ has at least one vertex $a \in A\backslash B$ and one vertex $b \in B\backslash A$. Let $\mathcal{P}_B = E(\mathcal{C}) \cap E(G[B])$, and let $P_A = E(\mathcal{C})\backslash\mathcal{P}_B$. We prove the correctness by a case distinction on the value of $V(\mathcal{P}_B) \cap \{x, y, z\}$.



(a) Subpaths of $\mathcal{C}$

(b)Wrong if $|V(\mathcal{P}_B) \cap \{x, y, z\}| = 3$

FIGURE 3.3: In (a) $\mathcal{C}$ is divided into two paths by y, z. Both $\mathcal{P}_A$ and $\mathcal{P}_B$ are yz-paths and their edges are only in one part of the separation. In (b), if $\mathcal{P}_B$ visits all three elements of the separator, then $\mathcal{P}_A$ cannot visit all of them, otherwise $\mathcal{C}$ is not a valid cycle.

1. $|V(\mathcal{P}_B) \cap \{x, y, z\}| = 0$ or 1: Since $\mathcal{C}$ is assumed to have vertices in both $A \backslash B$ and $B \backslash A$, then these two cases are impossible.

2. $|V(\mathcal{P}_B) \cap \{x, y, z\}| = 2$: Let $V(\mathcal{P}_B) \cap \{x, y, z\} = \{p, q\}$, since $\mathcal{P}_B \subseteq E(G[B])$ and p, q are two elements of the the separator, $\mathcal{P}_B$ is a pq-path in $G[B]$. Besides, since $\mathcal{P}_A = E(\mathcal{C}) \backslash \mathcal{P}_B$, by definition of separation, there does not exist an edge $\{m, n\}$ such that $m \in A \backslash B$ and $n \in B \backslash A$, then $\mathcal{P}_\mathcal{A} \subseteq E(G[A]) = E(G[A \cup B]) \backslash E(G[B])$, which means $\mathcal{P}_A$ is a pq-path in $G[A]$.
   Let $\mathcal{P}_i$ be a maximum-length pq-path in $G[A]$, if we replace $\mathcal{P}_A$ by $\mathcal{P}_i$ we obtain a new cycle. This cycle is a simple cycle because A and B are parts of separation so any paths(and hence, cycles) across these two parts must visit at least one element of the separator, since $\mathcal{P}_i$ has two ends p, q and $\mathcal{P}_i$ is a simple path, then $\mathcal{P}_i$ visits p, q only once hence does not create a new path nor a new cycle through p, q. Let $r = \{x, y, z\} \backslash \{p, q\}$, since $\mathcal{P}_B$ does not visit r then edge $(r, N(r)) \notin \mathcal{P}_B$ so adding any edge (m', n') with $m', n' \in A$ to $\mathcal{P}_i$ won't create a path to $\mathcal{P}_B$ through r, hence no new path nor cycle is created, therefore $\mathcal{P}_A \cup \mathcal{P}_B$ is a simple cycle. Besides, since $\mathcal{P}_i$ has a maximum length, this replacement does not decrease the length of the cycle, hence the resulting cycle is still a k-cycle. Since there are three possible combinations of pq, we just keep a maximum-length path between xy, yz, xz in $G[A]$, namely $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, and then we can obtain a k-cycle on $G[(\bigcup_{i=1}^{3} V(\mathcal{P}_i) \cup B)]$ after the replacement.

3. $|V(\mathcal{P}_B) \cap \{x, y, z\}| = 3$: In this case $\mathcal{P}_B$ visits all three elements of the separator then $\mathcal{P}_A$ cannot visit all these three elements, i.e., there must be a $r \in \{x, y, z\}$ such that for every $r' \in N(r)$, edge $\{r, r'\} \cap \mathcal{P}_A = \emptyset$, otherwise from the rest two elements of the separator $\{p, q\}$, we can find at least one element p such that there are two different cycles from p to p in $\mathcal{P}_A \cup \mathcal{P}_B$, since $\mathcal{C} = \mathcal{P}_A \cup \mathcal{P}_B$ then $\mathcal{C}$ is not a simple cycle. Such fact means $\mathcal{P}_A$ is a pq-path in $G[A \backslash \{r\}]$. Then we can still find a maximum-length pq-path $\mathcal{P}_i$ in $G[A \backslash \{r\}]$ to replace $\mathcal{P}_A$. Since P_i is a pq-path picked in the same graph of $\mathcal{P}_A$ then the resulting cycle is a simple cycle; since $\mathcal{P}_i$ is a maximum-length path, then the replaced cycle is remaining a k-cycle. Consider there are also three combinations of p, q, r, we just keep one maximum-length path between xy, yz, xz in $G[A] \backslash \{z\}, G[A] \backslash \{x\}$ and $G[A] \backslash \{y\}$, namely $\mathcal{P}_4, \mathcal{P}_5, \mathcal{P}_6$, and then we can obtain a k-cycle on $G[(\bigcup_{i=4}^{6} V(\mathcal{P}_i) \cup B)]$ after the replacement.

Since the separator can have at most three elements, it is impossible that $|V(\mathcal{P}_R) \cap \{x, y, z\}| > 3$ hence the cases are exhaustive and conclude the proof of Lemma 1. □

We also need the following lemma from Jansen's research:

Lemma 3.1.2. *[18]* Let $A, B \subseteq V(G)$ be a separation of order two of a graph G with $A \cap B = \{x, y\}$. Let $V(\mathcal{P}_A)$ be the vertices on a maximum-length xy-path $\mathcal{P}_A$ in $G[A]$, or $\emptyset$ if no such path exists. If G has a k-cycle, then $G[A]$ has a k-cycle or $G[V(\mathcal{P}_A) \cup B)]$ has a k-cycle.

Proof. Similar to Lemma 3.1.1, let $\mathcal{C}$ be a k-cycle in $G[A \cup B]$, if $V(\mathcal{C}) \subseteq A$ or $V(\mathcal{C}) \subseteq B$ then we are done. If $V(\mathcal{C})$ contains a vertex from $A \setminus B$ and a vertex from $B \setminus A$, then we can always use a maximum-length xy-path $\mathcal{P}_A$ in $G[A]$ to replace $V(\mathcal{C}) \cap A$, since $|\mathcal{P}_A| \geq |V(\mathcal{C}) \cap A|$. □

We now show that if given an oracle that can answer YES/NO to the question 'Is there a k-cycle in graph G?', then we can use this oracle for finding a maximum-length xy-path. The following lemma has been developed and proved by Jansen [18] and we show it here for completeness.

Lemma 3.1.3. There is an algorithm that, given an n-vertex graph G with distinct vertices x, y, and an integer k, either:

1. Determines whether there is a k-cycle in G
2. Determines the length of maximum-length xy-path and outputs its vertices

The algorithm runs in polynomial time if there is an oracle that able to decide the k-cycle problem. The oracle is queried for instance (G', k), where G' is obtained from G by adding a sequence of vertices and edges. G' has order of at most $|V(G)| + k$.

Proof. Given an input (G, k, x, y) we proceed as following: The algorithm first checks whether $\{x, y\}$ are in the same component. If this is not the case then the algorithm returns an empty set and halts, otherwise the algorithm invokes the oracle with instance (G, k). If the oracle answers YES then we achieve the first point and the algorithm reports the result and halts. We now focus on the situation that G has an xy-path but does not have a k-cycle.

Determining length: The algorithm determines the length of the longest xy-path in the following way: We create a sequence of graphs $G_0, ..., G_{k-2}$, for graph G_0 we just add an edge between x and y if such edge does not exist. For other $l > 0$, we create l degree-2 vertices $v_1, ..., v_l$, and let edge $\{x, v_1\}, .., \{v_j, v_{j+1}\}, ..., \{v_l, y\} \in E(G_l)$, for every $j < l - 1$. Then we query the oracle in sequence with instance $(G_0, k), .., (G_l, k)$ and find the smallest index l^* such that the oracle answers YES. Since G_{l^*-1} does not have a k-cycle but G_{l^*} does, and the circumference of G_{l^*} is at most circumference G_{l^*-1} plus one, then G_{l^*} has a cycle of length *exactly* k. Since in G_{l^*} we add an xy-path of length $l + 1$, then in G there is an xy-path of length at most $k - (l + 1)$. Note that this value is always positive, since value of l is bounded by $k - 2$. We choose $k - 2$ as an upper bound, because these $k - 2$ vertices, together with x, y always form a k-cycle.

Finding path: Now we can determine the vertex of the xy-path: Suppose l^* is the smallest index that a k-cycle is found, then for the $n + l^*$ vertices in G_{l^*}, we give them an index of $u_1, ..., u_{n+l^*}$. We first initialize $H = G_{l^*}$, and for $i \in [0, n + l^*]$ we query the instance of $(H - u_i, k)$ to determine whether we can find a k-cycle avoiding u_i, and if this is the case we just set $H = H - u_i$. Since the first H has a cycle of length exactly k, and the algorithm preserves the existence of a k-cycle, then after

all the vertices has been tested, the final graph H' maintains a cycle of length k, and every its vertex is necessary to this k-cycle. Let $V(l^*) = V(G_{l^*}) \setminus V(G)$ be the set of vertices that we add to G_{l^*}, then the vertex set of the maximum-length xy-path is obtained by subtracting $V(l^*)$ from $V(H')$.

Finally consider the size of the instance sent to the oracle: As we send a sequence of graphs $G_0, ..., G_{k-2}$ to the oracle, note that a graph G_l is a graph obtained from G by adding l vertices. Hence when determining length the oracle is queried by an instance with at most $|G| + (k-2)$ vertices. Besides, when finding the path we do not add vertices to the graph of the instance. Therefore the oracle is queried by an instance including a graph of at most $|G| + k$ vertices.

Hence we achieved the targets as claimed. Since the algorithm queries the oracle for $O(k)$ times when determining length and $O(n+k)$ times when finding the path, the algorithm indeed terminates in polynomial time. □

3.2 k-cycle in planar graphs

In this section we show a Turing Kernelization algorithm for determining a k-cycle on planar graphs. To achieve such kernel, we first transform a planar graph G into a quasi-4-tree decomposition. As introduced in theorem 2.5.1, every pair of adjacent bags in the tree would have adhesion of at most 3. Consider the fact that a cycle is 2-connected, then we can never find a cycle that covers vertices from two bags that have adhesion of size one. Therefore, we can avoid the existence of adhesion of size 1 by implementing the following technique:

Before the algorithm starts, we first decompose the graph into biconnected components. Note that this can be done in linear time using a DFS algorithm [17]. If there is a k-cycle in the graph, then it is contained entirely within one biconnected component. Hence we run the Turing kernelization individually on each biconnected component. In the next step we transform every biconnected components into quasi-4-connected components. As we do this on a biconnected components, all adhesions will have size at least 2 since there are no separators of size 1 in a biconnected graph.

In the following we only discuss about bags have adhesion of size 2 or 3: Approach about bags of adhesion of size 2 mainly comes from Jansen's research, since most properties of *Tutte decomposition*(which is used in his research) are preserved by *Quasi-4-decomposition*.

Algorithm 1: Reduce-C(G, G', A, B, F, k)

Precondition : G' is an induced subgraph of G, (A, B) is a separator of G' with $A \cap B = F$, and F is a minimal separator of G. F can be size 2 or 3.

Postcondition: The existence of a k-cycle in G, or the graph G' is updated by removing all vertices in $A \setminus B$ except at most 6 necessary paths. If G' initially has a k-cycle, then the deletions preserves this property.

1 Query the k-cycle oracle to determine whether $G'[A]$ has a k-cycle
2 **if** $F = \{x, y, z\}$ **then**
3 | Find vertices S of the 6 maximum-length paths mentioned in lemma 3.1.1 on $(G'[A])$
4 **else if** $F = \{x, y\}$ **then**
5 | Find vertices S of the maximum-length path mentioned in lemma 3.1.2 on $(G'[A])$
6 **if** *oracle answer YES or line 3(line 5) reports paths of length* $\geq k - 1$ **then**
7 | Report the existence of a k-cycle and halt
8 **else**
9 | Remove vertices in $A \setminus (S \cup F)$ from A
10 **end**

Lemma 3.2.1. Algorithm 1 satisfies its specifications. It calls the k-cycle oracle on a graph of at most $|A| + k$ vertices.

Proof. We first argue that the algorithm acts correctly if it reports a k-cycle. If the oracle detects a k-cycle in $G'[A]$, then clearly G' has a k-cycle as it is a supergraph of $G'[A]$. Besides, if the oracle detects a path of length at least $k - 1$ between vertices from the separator F in $G'[A]$, then its supergraph G' has such $k - 1$ length path. Note that both ends of such path is an element of a minimal separator, then by the definition of minimal separator there must be at least one connected component C' such that is adjacent to both ends and none of its interior vertices is covered by A and then we find a k-cycle. Therefore, line 6 reports a path of length $\geq k - 1$ yields a k-cycle.

We now consider the situation that no k-cycle is reported. Since S is a vertex set of 6 maximum-length paths between vertices from the separator F if G has a separator of order three, and S contains vertices forms a maximum xy-path if G has a separator of order two, then by lemma 3.1.1 and lemma 3.1.2, G' has a k-cycle if and only if $G'[S \cup B]$ has a k-cycle and hence the algorithm can safely delete vertices of $A \setminus (S \cup T)$. Here with take the union with T to prevent T is removed. This is possible to happen in the case that $S = \emptyset$.

As S includes at most 6 maximum-length paths of length at most $k - 1$, then $|S| \leq 6k$. Hence after removing vertices from $A \setminus (S \cup T)$ there will be at most $|B| + 6k$ vertices in the resulting graph G'. Further more, lemma 3.1.3 preserves every instance sent to the oracle has at most $|A| + k$ vertices. □

Lemma 3.2.1 shows a reduction rule to one part of a separation. This gives us the insight to solve the planar k-cycle problem: A planar graph G either:

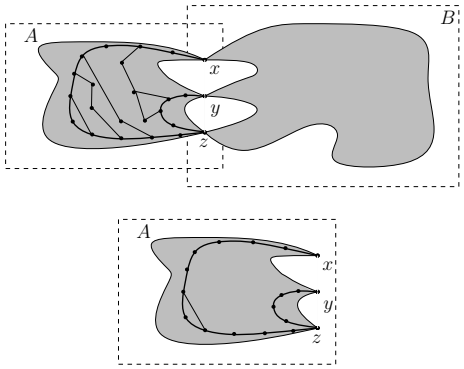


FIGURE 3.4: Illustration of how an instance of the k-cycle problem can be reduced based on a separation (A, B) of order three with $A \cap B = \{x, y, z\}$. As shown in the upper figure, if $G[A]$ contains some vertices that not belong to one of the 6 maximum-length paths, then these vertices are removed by line 9 of algorithm 1, as shown by the lower figure.

- has big bags in its quasi-4-decomposition so that we can announce it has a k-cycle directly or,
- every its bag in its quasi-4-decomposition can be reduced to a small size and queried by the oracle.

Theorem 3.2.2. The planar k-cycle problem has a polynomial Turing Kernel: it can be solved in polynomial time using an oracle that decides planar k-cycle instances with $O(k^2)$ vertices and parameter value k.

Proof. Such planar k-cycle kernel can be done by the following three steps:

Decompose. Let (G, k) be an input, we use the algorithm mentioned in theorem 2.5.1 to decompose G into a quasi-4-tree decomposition $(T, \mathcal{X})$. For each edge $i, j \in E(T)$ of the decomposition tree, the way of computing the decomposition ensures that $\mathcal{X}(i) \cap \mathcal{X}(j)$ is a minimal separator in G. Note that the size of adhesion maybe at most 3, but as discussed before, there exists a technique that prevent the existence of adhesions of size 1, therefore only discuss bags that has adhesion of size 2 or 3.

Claim 1. If there is a node $i \in V(T)$ such that $|\mathcal{X}(i)| \geq 2k$, then G has a k-cycle.

Proof. By the definition of quasi-4-tree decomposition, $\text{Torso}(G, \mathcal{X}(i))$ is a minor of G. Since planarity is preserved when taking minors, $\text{Torso}(G, \mathcal{X}(i))$ is also planar. Suppose $|\mathcal{X}(i)| \geq 2k$ then by theorem 2.5.2 the torso contains a cycle of length $\frac{1}{2}(2k + 4) \geq k$. Consider the definition of torso: As torso is a minor of G, if torso contains a k-cycle then G contains a minor that has a k-cycle. As we obtain a minor by reducing vertices, edges and contracting vertices from an original graph. As all these operations can only make a cycle shorter, if a minor of G contains a k-cycle, then the original graph G must contain a cycle of length at least k. □

Claim 1 shows that we may safely output YES if $(T, \mathcal{X})$ has a width larger than 2k. In the following we just assume the tree width is smaller than $2k$. Then we are going into the reduction phase. When doing the reduction, we make a copy G' of G, as well as a copied tree decomposition $(T', \mathcal{X}')$ from $(T, \mathcal{X})$. We assign an arbitrary node as the root of the tree and remove any leaf if it is a subset of its parent. In the progress of

Algorithm 2: Kernelize-C$(G, G', (T', \mathcal{X}'), i, k)$

Precondition : G' is an induced subgraph of G with a tree decomposition $(T', \mathcal{X}')$ of adhesion at most three, a node i of T' is specified as root.

Postcondition: The existence of a k-cycle is reported, or the graph G' and the tree decomposition $(T', \mathcal{X}')$ are updated by removing vertices of $\mathcal{X}'(T'[i]) \setminus \mathcal{X}'(i)$, resulting in $|\mathcal{X}'(T'[i])| \leq O(k^2)$. If G' initially has a k-cycle, then the deletions preserves this property.

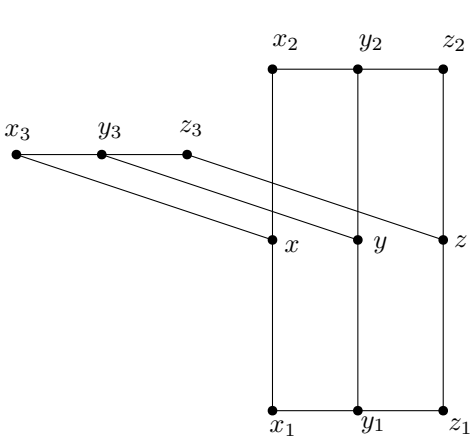
1 **for** *each child j of i in T'* **do**
2 Recursively execute Kernelize-C$(G, G', (T', \mathcal{X}'), j, k)$
3 Let $F := \mathcal{X}'(i) \cap \mathcal{X}'(j)$
4 Let $A := \mathcal{X}'(T[j])$ and $B := (V(G') \setminus A) \cup F$
5 Reduce-C(G, G', A, B, F, k)
6 **end**
7 **for** *each pair $(x, y) \in \binom{\mathcal{X}'(i)}{2}$* **do**
8 **while** *there are distinct children j_1, j_2 of i in T' such that $\mathcal{X}'(i) \cap \mathcal{X}'(j_1) = \mathcal{X}'(i) \cap \mathcal{X}'(j_2) = \{x, y\}$* **do**
9 Let $F := \{x, y\}$ Let $A := \mathcal{X}'(T[j_1]) \cup \mathcal{X}'(T[j_2])$ and $B := (V(G') \setminus A) \cup F$
10 Reduce-C(G, G', A, B, F, k)
11 **end**
12 **end**

the reduction, torso of nodes in $(T', \mathcal{X}')$ are no longer necessarily quasi-4-connected, but we preserve that their adhesion is as same as $(T, \mathcal{X})$.

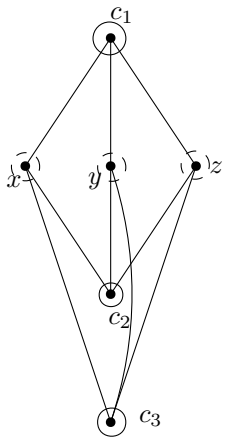
Query and reduce. In this phase, we repeatedly query the oracle about the existence of the k-cycle and reduce vertices if no such cycle is detected. We show the main procedure of the reduction phase as algorithm 2. This algorithm is initially called from the root of tree T' and is expanded recursively. The algorithm runs from bottom to up after fully expanded by first attacking a child node j of i. If j and i has an adhesion of size three then the algorithm simply calls Reduce-C to detect the k-cycle in $\mathcal{X}'(T'[j])$ and reduce vertices in that subtree. If their adhesion has size of 2 as $\{x, y\}$, the algorithm first does the same attack before checking all children of i who has same adhesion $\{x, y\}$ and finally only maintains the child in which contains the maximum-length xy-path. During the recursion, if any k-cycle is detected the algorithm reports this and halts. If the algorithm terminates without report a k-cycle, we just call the oracle for the final time with the remaining graph G' and parameter k. The output is the answer to the planar k-cycle problem.

Claim 2 Let $i \in V(T)$ be a node of the tree. Let i has p pairs (m, n) such that every pair is an adhesion of its children, and has q triples $\{x, y, z\}$ such that every triple is an adhesion of its children, then after the execution of line 11 of Algorithm 2, Torso$(G, \mathcal{X}'(i))$ has at most:

- p subtrees of adhesion of size 2, with each subtree has at most k vertices, and
- $2q$ subtrees of adhesion of size 3, with each subtree has at most $6k$ vertices



(a) An example of a bag has 3 children share a same adhesion of size 3. Note that $\{x_p, y_p, z_p\}$ can be contracted into a single vertex c_p, for $p \in [1,3]$.



(b) Such minor contains a $K_{3,3}$, as shown in normal and dashed cycles. Note that C_q are contracted vertices from x_q, y_q, z_q, for $q \in [1,3]$.

FIGURE 3.7: Shows why any bag can have at most 2 children share an adhesion of size 3: If this is not the case, then graph G must contain a minor that can be contracted to the graph shown in (b). Such minor contains a $K_{3,3}$, and hence the g contains a $K_{3,3}$ as a minor, implies that G is not planar.

Proof. We first show the part of adhesion of size 3. To prove the lemma, we first claim that every bag i can have at most two children j_1, j_2, such that $\mathcal{X}(i) \cap \mathcal{X}(j_1) = \mathcal{X}(i) \cap \mathcal{X}(j_2) = \{x, y, z\}$:

Suppose $\mathcal{X}'(T'[j_p])$ contains only $\{x, y, z\}$ then we are no longer interested in these subtrees, as they will not increase the value of $|\mathcal{X}'(T'[p])|$. Suppose there is some more vertices, consider that $\{x, y, z\}$ is a 3-separator in a planar graph G and hence from $\mathcal{X}'(T'[j_p])] \setminus \mathcal{X}(i)$ we can always find a vertex c_p, such that edges $\{x, c_p\}, \{y, c_p\}$, $\{z, c_p\}$ is covered by a minor of G. Now consider the case that if $p > 2$, then G has a minor that contains a $K_{3,3}$, which contradicts that G is a planar graph.

Now we have shown that every bag can have at most two children with common adhesion $\{x, y, z\}$, which implies there exists at most two such subtrees. Consider that in line 5 of algorithm 2 these subtrees has been invoked by Algorithm 1 and contains 6 maximum-length paths according to lemma 3.1.1, then every subtree has at most $6k$ vertices.

Now we go back to the case of adhesion of size 2. First consider line 5 of Algorithm 2: For every subtree rooted at child j of i, such that $\mathcal{X}(i) \cap \mathcal{X}(j) = \{x, y\}$, the algorithm invokes Algorithm 1 on it and the subtree only maintains a maximum-length xy-path after algorithm 1 is executed. Then in line 8 of Algorithm 2, for every two children j_1, j_2 that has adhesion $\{x, y\}$, the algorithm sets the union of the subtree that rooted at j_1, j_2 as A and invokes Algorithm 1. Considering each

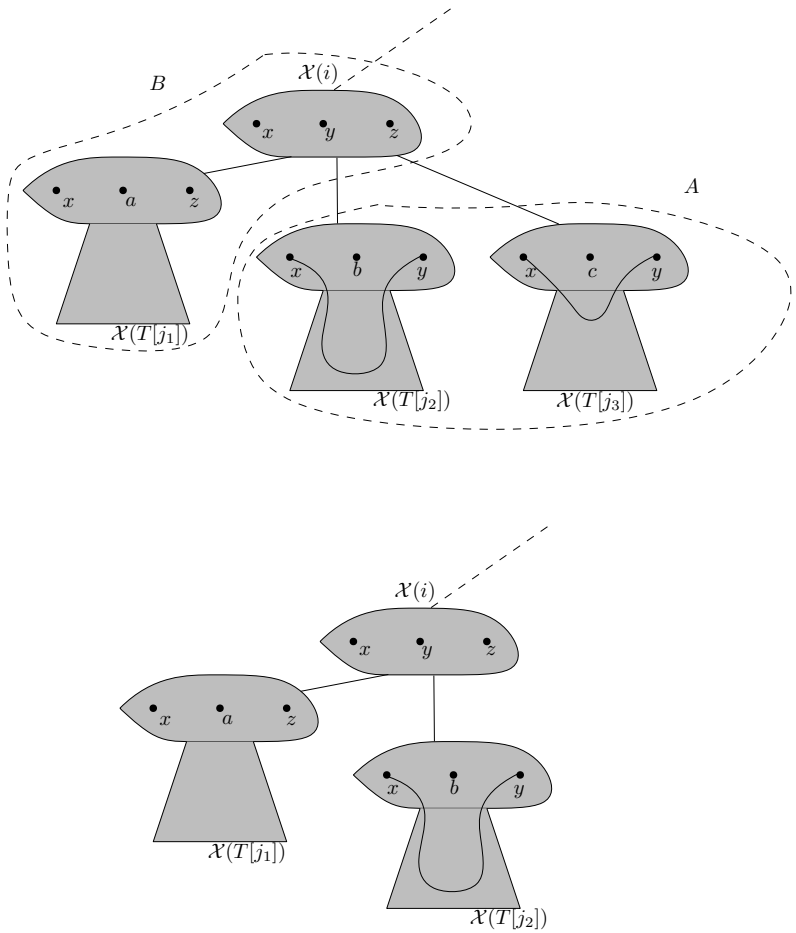


FIGURE 3.8: Illustrates the behavior between line 8 and line 11 of algorithm 2. Let j_1, j_2, j_3 be children of i such that j_2 and j_3 shares the same adhesion of size 2, as shown in the upper figure. Then in line 9, algorithm 2 sets $\mathcal{X}(T[j_2]) \cup \mathcal{X}(T[j_3])$ as A. Consider the postcondition of algorithm 1, and each the two subtrees in fact only contains an xy-path, then after line 11 only the longer xy-path is left, which implies one subtree is removed from the graph, as shown in the lower figure.

subtree only maintains an xy-path, and algorithm 1 only preserves a maximum-length xy-path, then after the execution of Algorithm 1 there is only one maximum-length xy-path left with length at most $k-2$. Note that such path can be only formed by vertices from one of the subtrees, otherwise there must exist some vertex $v \in \mathcal{X}(T[j_1]) \cap \mathcal{X}(T[j_2]) \setminus \{x, y\}$, implies that $v \in \mathcal{X}(i)$ and v is a member of the adhesion, and shows a contradiction. Therefore, after the complete execution of algorithm 2, for every adhesion $\{x, y\}$, it can only have one subtree that contains a maximum-length xy-path of size $\leq k-2$. □

Claim 2 shows that every node can maintain limited number of subtrees, with the total amount of vertices of the subtree is also limited, after a complete execution of algorithm 2 on it. Furthermore, the execution of algorithm 2 does not break the planarity, as we only remove vertices from the graph. Since the self-reduction algorithm only removes vertices from $A \setminus B$ in a separation (A, B), a minimal separator T is remaining kept after the execution of the algorithm. Thus, the oracle is invoked correctly during the execution, and since the algorithm starts the self-reduction bottom-up, we have the following claim as an upper bound of the size of

the queried instances:

Claim 3. Algorithm 2 only queries the k-cycle oracle with parameter k on planar graphs of order at most $78k^2 + k$.

Proof. As discussed, graph G preserves planarity during the execution of algorithm. As a subgraph of G, any node i preserves the planarity in $G'[\mathcal{X}(i)]$ and its torso is also planar. Recall that every quasi-4-connected torso has only limited number of vertices, then graph $\text{Torso}(G', \mathcal{X}'(i))$ is a planar graph with at most $2k$ vertices. Since quasi-4-tree decomposition ensures the adhesion of two bags is a minimal separator of graph G, by lemma 2.5.4, number of adhesion of size 2 is bounded by the number of edges in $\text{Torso}(G', \mathcal{X}'(i))$ and number of adhesion of size 3 is bounded by the number of triangle $\text{Torso}(G', \mathcal{X}'(i))$.

Consider the fact that, in a planar graph of n vertices there are at most $3n$ edges [7] and at most $3n$ triangles [25], they by **Claim 3**, a subtree rooted at node i have at most $k * 3 * (2k) + 6k * 2 * 3 * (2k) + k = 78k^2 + k$ vertices after all children of i have been treated by algorithm 2 but before i itself is treated. Then, when node i is set as A and Reduce-C is invoked in line 5, $|A| \leq 78k^2 + k$. Furthermore, if node i is set as a part of A and in line 10, then i has an adhesion of size 2 with its parent and therefore $|\mathcal{X}(T[i])| \leq k$, and $|A| \leq 2k$, as the subtree rooted at i only contains a path of length smaller than k.

We showed that the algorithm outputs correct answer and satisfies all requirement of a Turing Kernelization, which proves theorem 3.2.2. □

Chapter 4

Turing Kernelization for finding paths

We now turn to the k-path problem. The main idea is same as the k-cycle problem but is a bit more complicated. We use the same strategy as in chapter 3, that we first introduce a self-reduce rule in section 4.1, and show the Turing Kernel in section 4.2.

4.1 Properties of paths

The self-reduction rule in k-path problem is a bit more complicated than the k-cycle problem because the size of the adhesion, as well as the minimal separators, can now be 1, 2 or 3. We first discuss the situation of minimal separators of size 3.

Lemma 4.1.1. Let $A, B \subseteq V(G)$ be a separation of order three of a graph G with $A \cap B = \{x, y, z\}$. Let $\mathcal{P}_1, ..., \mathcal{P}_{27}$ be subgraphs of $G[A]$ such that:

1. $\mathcal{P}_1$ to $\mathcal{P}_3$ are maximum length u-paths in $G[A]$, for each $u \in \{x, y, z\}$.
2. $\mathcal{P}_4$ to $\mathcal{P}_9$ are maximum length u-paths in $G[A] \setminus \{v\}$, for each $u \in \{x, y, z\}$.
3. Each of $\mathcal{P}_{10}$ to $\mathcal{P}_{12}$ consists of two vertex-disjoint paths in $G[A]$, one u-path and one v-path, such that combined length of these paths is maximized, for each choice of $u, v \in \{x, y, z\}$ with $u \neq v$.
4. $\mathcal{P}_{13}$ to $\mathcal{P}_{15}$ are maximum length uv-paths in $G[A]$, for each choice of $u, v \in \{x, y, z\}$ with $u \neq v$.
5. Each of $\mathcal{P}_{16}$ to $\mathcal{P}_{18}$ consists of two vertex-disjoint paths in $G[A]$, one uv-path and one w-path, such that combined length of these paths is maximized, and for each choice of $u, v, w \in \{x, y, z\}$, u, v, w are unique.
6. $\mathcal{P}_{19}$ to $\mathcal{P}_{21}$ are maximum length uw-paths in $G[A] \setminus \{v\}$, for each choice of $u, w \in \{x, y, z\}$ with $u \neq w$.
7. Each of $\mathcal{P}_{22}$ to $\mathcal{P}_{24}$ consists of two vertex-disjoint paths in $G[A] \setminus \{u\}$, one v-path and one w-path, such that combined length of these paths is maximized, and for each choice of $u, v, w \in \{x, y, z\}$, u, v, w are unique.
8. $\mathcal{P}_{25}$ to $\mathcal{P}_{27}$ are maximum length u-paths in $G[A] \setminus \{v, w\}$, and for each choice of $u, v, w \in \{x, y, z\}$, u, v, w are unique.

If G has a k-path, then $G[A]$ has a k-path or $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$ has a k-path.

Proof. Consider a k-path $\mathcal{P}$ in G. Observe the lemma is trivial if $\mathcal{P} \subseteq G[A]$ or $\mathcal{P} \subseteq G[B]$, hence in the following we assume that $\mathcal{P}$ contains at least one edge in $E(A)$ and one edge in $E(B)$. Let $\mathcal{P}_B = E(\mathcal{P}) \cap E(G[B])$ and $\mathcal{P}_A = \mathcal{P} \setminus \mathcal{P}_B$, note that both $\mathcal{P}_A$ and $\mathcal{P}_B$ are not necessarily to be a connected component. Then we can show the correctness of the lemma by discussing how many elements of separator $\{x, y, z\}$ is occupied by $V(\mathcal{P}_B)$.

1. If $V(\mathcal{P}_B) \cap \{x, y, z\} = \{u\}$, for some $u \in \{x, y, z\}$, then $\mathcal{P}_B$ is a u-path that does not visit v and w and $\mathcal{P}_A$ is a u-path in $G[A]$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length u-path in $G[A]$, namely $\mathcal{P}_1$ to P_3, meanwhile still maintaining a k-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

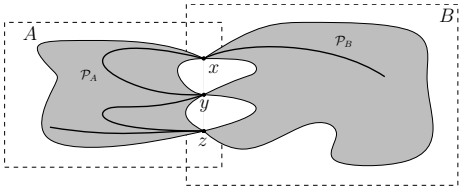
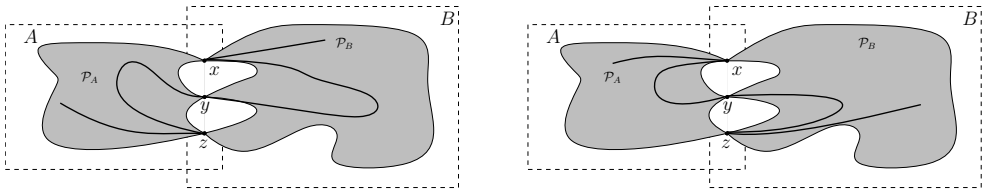


FIGURE 4.1: Displays case 1.1 when $u = x$. In this case, $\mathcal{P}_B$ is a simple x-path, hence $\mathcal{P}_A$ can be replaced by a maximum length x-path in $G[A]$.

2. If $V(\mathcal{P}_B) \cap \{x, y, z\} = \{u, v\}$ for some $u, v \in \{x, y, z\}$ and $u \neq v$, and let $w = \{x, y, z\} \setminus \{u, v\}$, then we come to three cases:

   2.1 If $\mathcal{P}_B$ is connected, and $\mathcal{P}_B$ has one end intersects with $\{x, y, z\}$: In this case $\mathcal{P}_B$ is a u-path that also occupies v and $\mathcal{P}_a$ is a u-path in $G[A] \setminus \{v\}$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length u-path in $G[A] \setminus \{v\}$, namely $\mathcal{P}_4$ to P_9, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.



(a) Case 2.1, when $V(\mathcal{P}_B) \cap \{x, y, z\} = \{x, y\}$, and $u = y$

(b) Case 2.1, when $V(\mathcal{P}_B) \cap \{x, y, z\} = \{y, z\}$, and $u = y$

FIGURE 4.4: Displays case 2.1. In case 2.1, there are two possible $\mathcal{P}_B$ whose ends $End(\mathcal{P}_B)$ that satisfies $End(\mathcal{P}_B) \cap \{x, y, z\} = \{y\}$, therefore we need 6 paths to maintain a K-path.

   2.2 If $\mathcal{P}_B$ is connected, and $\mathcal{P}_B$ has two ends intersect with $\{x, y, z\}$: In this case $\mathcal{P}_B$ is a uv-path and does not occupy w, hence $\mathcal{P}_A$ consists of two vertex-disjoint paths in $G[A]$, one is a u-path and one is v-path. Therefore, $\mathcal{P}_A$ can be replaced by two vertex-disjoint paths in $G[A]$, one u-path and one v-path with maximized length in combination, namely $\mathcal{P}_{10}$ to P_{12}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

   2.3 If $\mathcal{P}_B$ is disconnected: In this case $\mathcal{P}_B$ must be two vertex-disjoint paths, one u-path and one v-path and any $\mathcal{P}_A$ should be a connected uv-path

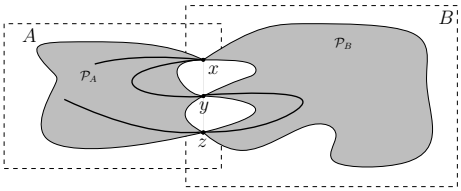


FIGURE 4.5: Displays case 2.2 when $u, v = x, y$. In this case, $\mathcal{P}_B$ is an xy-path, hence $\mathcal{P}_A$ can be replaced by two vertex disjoint u-path and v-path in $G[A]$ with a maximum combined length.

in $G[A]$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length uv-path in $G[A]$, namely $\mathcal{P}_{13}$ to P_{15}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

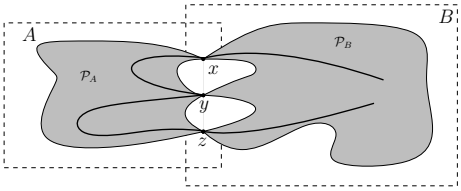


FIGURE 4.6: Displays case 2.3 when $u = x$ and $v = z$. In this case, $\mathcal{P}_B$ consists of two disconnected x-path and z-path, hence $\mathcal{P}_A$ can be replaced by a maximum length xz-path in $G[A]$.

3. If $V(\mathcal{P}_B) \cap \{x, y, z\} = \{u, v, w\}$, where u, v, w are pairwise different. We first claim that $\mathcal{P}_B$ cannot be three disconnected components otherwise $\mathcal{P} = \mathcal{P}_A \cup \mathcal{P}_B$ can never form a simple path no matter how P_A is formed. In the remaining we come to two cases based on whether $\mathcal{P}_B$ is connected:

   3.1 If $\mathcal{P}_B$ is disconnected, then $\mathcal{P}_B$ consists of a connected component that passes through u, v, and a w-path. We remain need to discuss whether u, v are both ends of the connected component:

      3.1.1 If both u, v are ends of the connected component: In this case the connected component is a uv-path and $\mathcal{P}_B$ consists of two disconnected paths, one uv-path and one w-path, and $\mathcal{P}_\mathcal{A}$ is either a uw-path in $G[A] \setminus \{v\}$ or a vw-path in $G[A] \setminus \{u\}$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length uv-path in $G[A] \setminus \{w\}$, namely $\mathcal{P}_{16}$ to P_{18}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

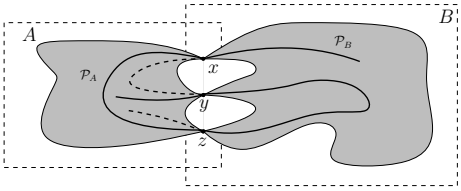


FIGURE 4.7: Displays case 3.1. In this case, $\mathcal{P}_B$ consists of two disconnected x-path and yz-path, hence $\mathcal{P}_A$ can be either an xz-path(normal line) or an xy-path(dashed line). Besides, $\mathcal{P}_A$ can also have a y-path (normal) or a z-path (dotted). Therefore we need to keep three combinations of vertex disjoint uv-paths and w-paths to maintain a k-path.

3.1.2 If only one vertex u is an end of the connected component: In this case $\mathcal{P}_B$ consists of two disconnected paths, one u-path and one w-path and $\mathcal{P}_A$ is a uw-path in $G[A] \setminus \{v\}$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length uv-path in $G[A] \setminus \{w\}$, namely $\mathcal{P}_{19}$ to P_{21}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

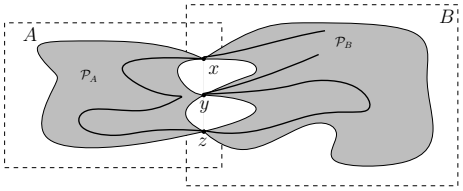


FIGURE 4.8: Displays case 3.1.2 when $u = z$ and $w = x$. In this case, $\mathcal{P}_B$ consists of two disconnected x-path and z-path, hence $\mathcal{P}_A$ can be replaced by a maximum length xz-path in $G[A] \setminus \{y\}$.

3.2 Now we assume that $\mathcal{P}_B$ is connected, then we still need to discuss how many vertices of $\{u, v, w\}$ is an end of $\mathcal{P}_B$

3.2.1 If two vertices u, v are two ends of $\mathcal{P}_B$, then $\mathcal{P}_B$ is a connected uv-path, and $\mathcal{P}_A$ can only be two vertex-disconnected paths in $G[A] \setminus \{w\}$, one is u-path and one is v-path. Therefore, $\mathcal{P}_A$ can be replaced by two vertex-disjoint paths in $G[A] \setminus \{w\}$, one u-path and one v-path with maximized length in combination, namely $\mathcal{P}_{22}$ to P_{24}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

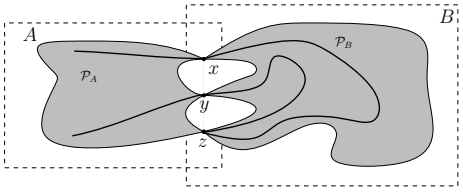


FIGURE 4.9: Displays case 3.2.1 when $u = x$ and $v = y$. In this case, $\mathcal{P}_B$ is an xy-path that passes through z, hence $\mathcal{P}_A$ can be replaced by two vertex-disjoint x-path and y-path in $G[A] \setminus \{z\}$ with a maximum combined length.

3.2.2 If only one vertex u is an end of $\mathcal{P}_B$: In this case $\mathcal{P}_B$ is a u-path and $\mathcal{P}_A$ is a u-path in $G[A] \setminus \{v, w\}$. Therefore, $\mathcal{P}_A$ can be replaced by a maximum-length u-path in $G[A] \setminus \{v, w\}$, namely $\mathcal{P}_{25}$ to P_{27}, meanwhile still maintaining a K-path, proving the existence of a k-path in $G[\bigcup_{i=1}^{27} \mathcal{P}_i \cup B]$.

As the cases are exhaustive, this concludes the proof of Lemma 4.1.1. □

This lemma shows that a separation of order three can be reduced to a subgraph that containing at most 27 (sub)-paths, meanwhile keeping the existence of k-path unchanged. Note that the length of each (sub)-path is smaller than $k - 1$, otherwise we can announce the existence of k-path directly, due to the definition of minimal separator.

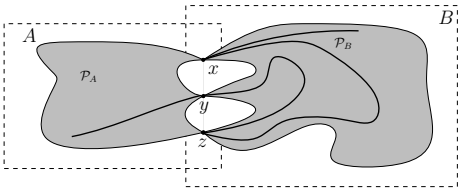


FIGURE 4.10: Displays case 3.2.2 when $u = y$. In this case, $\mathcal{P}_B$ is a y-path that passes through x, z, hence $\mathcal{P}_A$ can be replaced by a maximum length y-path in $G[A] \setminus \{x, z\}$.

In the following we show reduction rule when minimal separators has size 1 and 2. Such lemma is suggested and proved by Jansen.

Lemma 4.1.2. Let $A, B \subseteq V(G)$ be a separation, then:

- If A, B is a a separation of order one with $A \cap B = \{x\}$, let $\mathcal{P}_A$ be a maximum-length x-path in $G[A]$. If G has a k-path then $G[A]$ has a k-path or $G[V(\mathcal{P}_A) \cup B]$ has a k-path.
- If A, B is a a separation of order two with $A \cap B = \{x, y\}$. Let $\mathcal{P}_1, .., \mathcal{P}_6$ be subgraphs of $G[A]$ such that:
  1. $\mathcal{P}_1$ is a maximum-length x-path in $G[A] - \{y\}$.
  2. $\mathcal{P}_2$ is a maximum-length y-path in $G[A] - \{x\}$.
  3. $\mathcal{P}_3$ is a maximum-length x-path in $G[A]$.
  4. $\mathcal{P}_4$ is a maximum-length y-path in $G[A]$.
  5. $\mathcal{P}_5$ is a maximum-length xy-path in $G[A]$, or $\emptyset$ if no such path exists.
  6. $\mathcal{P}_6$ consists of two vertex-disjoint paths in $G[A]$, one x-path and one y-path, such that the combined length of these paths is maximized.

  If G has a k-path, then G[A] has a k-path or $G[\bigcup_{i=1}^{6} V(\mathcal{P}_i) \cup B]$ has a k-path.

The proof of lemma 4.1.2 can be checked in Jansen's research [18], which is similar to the proof of lemma 4.1.1.

Now we focus on how to achieve these reduction rules by an oracle. In order to maintain these paths, we need to know a maximum-length path with one end fixed, a maximum-length path with two ends fixed and two disconnected components with maximal combined length. Instead of constructing a new algorithm from a k-path oracle, we show that it is possible to fetch the information we need from an oracle that answers arbitrary NP-complete problem.

Definition 4. A 3-terminal graph is a tuple (G, x, y, z) where G is a graph and x, y, z are terminal vertices in G. Note that x, y, z is not necessarily unique. A *3-terminal edge property* is a function $\Pi : G \times T \to \{Yes, No\}$ such that takes a 3-terminal graph G and a subset $T \subseteq E(G)$ as input, and outputs either *Yes* or *No*. A 3-terminal edge property is polynomial decidable if there is an algorithm that can answer instance $(\Pi(G, x, y, z), T)$ in polynomial time.

For example, in this thesis we need three 3-terminal edge properties, which are:

1. 'Does the edge set T form an x-path?'

2. 'Does the edge set T form a xy-path?'

3. 'Does the edge set T form two vertex disjoint paths, one is an x-path and another is a yz-path?'

We now show how to find a maximum edge set T such that satisfies a polynomial decidable 3-terminal edge property $\Pi((G, x, y, z), T)$ in polynomial time, with the help of an oracle that answers arbitrary decision NP-complete problem $\mathcal{Q}$.

Given a 3-terminal edge property $\Pi((G, x, y, z), T)$, we first transform it to a parameterized decision problem L_Π: given a 3-terminal graph (G, x, y, z) and an integer k, is there an edge set T of size exactly k that satisfies $\Pi((G, x, y, z), T) = True$?. This decision problem is a NP problem since it can be solved by a non-deterministic Turing machine, by trying every T of size k and use polynomial time to check whether T satisfies $\Pi((G, x, y, z), T)$. Therefore, there must exist a polynomial computable transform function f, such that for any instance s satisfies $s \in L_\Pi$ if and only if $f(s) \in \mathcal{Q}$, where $\mathcal{Q}$ is some parameterized NP-complete problem. Furthermore, since the transformation function f is in polynomial time, then the size of output of the transformation $f(s)$ is bounded by some polynomial of the size of s. Regard that $f(s)$ is also the instance that queried by an oracle that solves $\mathcal{Q}$, then the queried size of the oracle is also bounded by $poly(|s|)$. As a parameter k is meaningful only if its value is no larger than $|s|$ hence the transformed value of k is still in polynomial.

Finding a maximum-satisfied set. Use these observations we may compute a maximum edge set T that satisfies a 3-terminal edge property Π: Let $|E(G)| = m$, for each $i \in [1, m]$ we consider it is an instance $s_i = ((G, x, y, z), i)$ of a parameterized problem L_Π. Next we transform every s_i into a corresponding instance of $\mathcal{Q}$, and query the $\mathcal{Q}$-oracle. If the oracle only returns no then we regard no edge set satisfies such property and output it. Otherwise we record the largest parameter k^* such that the oracle answers YES on $f(s_{k^*})$. Obviously k^* is the maximum size of the set that satisfies Π.

We may use a self-reduction algorithm finding an edge set satisfying the property Π from the value of k^*: For every edge in G we number them as $e_1, ...e_m$. Let $H := G$ and for $i \in [1, m]$, we query by sequence the instance $(f(H - \{e_i\}, x, y, z), k^*)$. If the oracle answers YES then we set $H = H - \{e_i\}$ since e_i is not necessary for satisfies the property Π. Since the algorithm ensures the remaining graph still satisfies the property and preserves the value of k^*, by the end of the algorithm the resulting graph H contains k^* edges and satisfies Π. Hence edges remaining in H is the maximum edge set we are looking for. Note that we can also describe an edge property problem which involves only one or two terminal vertices in the form of a 3-terminal edge property, by setting two or three of the terminal vertices as the same vertex. It is easy to see that the algorithm takes polynomial time and queries the oracle with instance of size and parameter in polynomial of n.

4.2 k-path in planar graphs

Using the self-reduction algorithm presented in the previous section, we now give a Turing kernel for a planar graph. The general idea is same as k-cycle problem: We first split a planar graph G into its quasi-4-tree decomposition in polynomial

time and then reduce each bag by removing vertices but only keeps few interesting paths. If some bags keep paths that has same ends then we compare them and only maintain maximum-length paths. We guarantee that during the kernelization the oracle is queried by instances whose size is in polynomial of the parameter.

Algorithm 3: Reduce-P(G', A, B, k)

Precondition : (A, B) is a separation of G' of order at most three
Postcondition: The existence of a k-path is reported, or the graph G' is updated by removing all vertices in $A \setminus B$ except $O(k)$ vertices. After the algorithm terminates, the graph $G'[A] - (A \cap B)$ maintains at most one connected component if $|A \cap B| = 1$, and at most 12 connected components if $|A \cap B| = 2$. If G' initially has a k-path, then the deletions preserves this property.

1 Let $F := A \cap B$
2 **if** *The k-path oracle answers YES on instance* $(G'[A], k)$ **then**
3 | Report the existence in G' of k-path and halt
4 **else if** $F = \{x, y, z\}$ **then**
5 | Find vertices S of the 27 maximum-length paths mentioned in lemma 4.1.1 on $(G'[A])$
6 **else if** $F = \{x, y\}$ **then**
7 | Find vertices S of the 6 maximum-length paths mentioned in lemma 4.1.2 on $(G'[A])$
8 **else if** $F = \{x\}$ **then**
9 | Find vertices S of the maximum-length X-path mentioned in lemma 4.1.2 on $(G'[A])$
10 Remove vertices in $A \setminus (S \cup F)$ from A

Lemma 4.2.1. Algorithm 3 satisfies it specifications. It works in polynomial time when access to an oracle that answers arbitrary parameterized problem Q whose underlying a NP-complete problem. The oracle is queried for instances of size of polynomial of $|A|$.

Proof. We first proof the correctness of the algorithm. If the oracle finds a k-path in $G'[A]$ then the algorithm reports this k-path hence it is obviously correct. If $G'[A]$ does not contain a k-path directly then between line 4 and line 9 the algorithm finds the maximum-length path(s) according to the order of the separator. As discussed in lemma 4.1.1 and lemma 4.1.2, removing vertices other than these paths from $G[A]$ preserves the existence of k-path hence the algorithm is remaining correct after executing line 10. As line 10 ensures $G'[A]$ has at most 27 path with each of length no larger than $O(k)$ then the size reduction we claimed in the postcondition is also true.

We now discuss the number of connected components in $G'[A] - F$ upon termination. If $|F| = 1$ then the algorithm maintains the only maximum-length x-path and such path is remaining connected in $G'[A] - F$, as $F = \{x\}$ is an end of it. When $|F| = 2$ then $G'[A]$ maintains six types of maximum-length paths, with each may yield two connected components after removing $F = \{x, y\}$, implies $G'[A] - F$ has at most 12 connected components. Here we are not interested in the number of component when $|F| = 3$. The reason has been partly shown in Chapter 3, and will be explained later.

Observe that during a single execution of Algorithm 3 it needs to determine edge sets of at most 27 3-terminal properties. As discussed, we need polynomial time for determining a single set if given suitable access to the oracle hence the algorithm terminates in polynomial time. Finally, as we only reduce edges when determining the edge set, the oracle queries for instances of size no larger than $|A|$.

Algorithm 4: Kernelize-$P(G, G', (T', \mathcal{X}'), i, k)$

Precondition : G' is an induced subgraph of G with a tree decomposition $(T', \mathcal{X}')$ of adhesion at most three, a node i of T' is specified as root.

Postcondition: The existence of a k-path is reported, or the graph G' and the tree decomposition $(T', \mathcal{X}')$ are updated by removing vertices of $\mathcal{X}'(T'[i]) \setminus \mathcal{X}'(i)$, resulting in $|\mathcal{X}'(T'[i])| \in O(k^2)$. If G' initially has a k-path, then the deletions preserves this property.

```
for each children j of i in T' do
    Recursively execute Kernelize-P(G, G', (T', X'), j, k)
    Let F:= X'(i) ∩ X'(j)
    Let A:= X'(T[j]) and B:= (V(G')\A) ∪ F
    Reduce-P(G', A, B, k)
end
for each vertex x ∈ X'(i) do
    while there are distinct children j1, j2 of i in T' such that
      X'(i) ∩ X'(j1) = X'(i) ∩ X'(j2) = {x}) do
        Let A:= X'(T[j1]) ∪ X'(T[j2]) and B:= (V(G')\A) ∪ {x}
        Reduce-P(G', A, B, k)
    end
end
for each pair (x, y) ∈ (X'(i) choose 2) do
    while there are 13 distinct children j1, ..., j13 of i in T' such that
      X'(i) ∩ X'(jl) = X'(i) = {x, y} for l ∈ [1, 13]) do
        Let F:= {x, y} Let A:= ⋃_{l=1}^{13} X'(T[jl]) and B:= (V(G')\A) ∪ F
        Reduce-P(G', A, B, k)
    end
end
```

We can achieve a Turing Kernel on the planar k-path problem if we use Algorithm 3 in a bottom-top structure. The main procedure is shown as Algorithm 4.

Lemma 4.2.2. The planar k-path problem has a polynomial Turing kernel, with each instance sent to the oracle has order of $poly(k)$. Such $poly(k)$ can be bounded as $O(k^2)$ if we are not restricted to use a YES/NO oracle.

Proof. We first show that, a planar graph G can solve a k-path problem in polynomial time when given access to a constant-time oracle that gives YES/NO answer on k-path problem, with every instance that queried by the oracle has size bounded by polynomial in the parameter.

Decompose. We compute a quasi-4-tree decomposition $(T, \mathcal{X})$ of G. Observe that if a graph contains a cycle of length $k+1$ then it contains a k-cycle, therefore we can

use the conclusion of lemma 2.5.2 as well as **claim 1**: if the width of $(T, \mathcal{X})$ is larger than $2k$ then we may report YES directly. If not, we make a copy G' of G and a copy $(T', \mathcal{X}')$ of the decomposition. We assign an arbitrarily node $i \in V(T)$ as the root of the tree.

Query and reduce. We use the same strategy as we used in the cycle problem: Algorithm 4 is invoked from the root of the tree, it then expends itself to the leaf of the tree and invokes Algorithm 3 from bottom to top. If a k-path is reported then we are done. If the algorithm terminates without reporting a k-path, we just transform the present graph G' to an instance of the oracle and invoke the oracle for a final time. The result of this invoke is the answer to the k-path problem.

The algorithm reduces the order of a subtree $T'[X'(i)]$ in two phases: it first attacks every child j of i and removes every vertices other than at most 27 paths with each length is smaller than k. Afterwards it reduces the number of children j: For those children that has same adhesion $\{x\}$, then algorithm compares the length of the x-path they contain in line 7 and removes every children except the one contains the maximum-length x-path after line 11. For those children has same adhesion $\{x, y\}$, since algorithm 3 removes all vertices except 12 connected components, if there are 13 subtrees share the same adhesion, at least one subtree is surely to be removed after the reduction step. For children share adhesion $\{x, y, z\}$, as we discussed in **Claim 2**, every bag can have at most two children that have the same adhesion of size 3 so as to maintain the planarity. Furthermore, as every subtree rooted at one of these children has been treated in algorithm 3, each subtree has order of at most $27k$. Note that since algorithm 3 preserves the existence of a k-path, and the property of quasi 4-tree decomposition guarantees every adhesion is a minimal separator, i.e., every separation (A, B) is defined correctly, then Algorithm 4 achieves its correctness by induction.

We still need to show the size of a subtree rooted at node i, after execution on it has terminated.

Lemma 4.2.3. As the execution of algorithm 4 on node i terminates, $|T'[\mathcal{X}'(i)]| = O(k^2)$

Proof. Upon termination, Torso$(G, \mathcal{X}(i))$ has at most:

- $2k$ vertices, such that every vertex can be an adhesion of size one that connect to at most one child. This is preserved by line 7 of algorithm 4, as the algorithm removes all but one child for every adhesion of size one. Consider every child contains a path of size less than k, these size-1 adhesions imply $O(k * 2K) = O(k^2)$ vertices.
- $6k$ edges, such that every pair of vertices connected by the edge can be an adhesion of size two that connect to at most 12 children. This is preserved by line 14. Consider the situation that there are 13 children share the same adhesion, then by line 14, algorithm 3 is invoked and after its execution at most 12 children are left, as argued. Since these 12 children in fact contains 6 paths with each length is smaller than k, these size-2 adhesion imply $O(6k * 3k) = O(k^2)$ vertices.
- $6k$ triangles, such that every triple of vertices is an adhesion of size 3. Consider that for a single triple of vertices, at most 2 children can share it as adhesion,

with each subtree rooted at these children contains at most $27k$ vertices, these size-3 adhesion implies $O(6k * 2 * 27k) = O(k^2)$ vertices.

Therefore, any subtree rooted at node i has $O(k^2) + O(k^2) + O(k^2) + 2k = O(k^2)$ vertices after the execution of algorithm 4 terminates on it. □

Unfortunately we cannot announce that we achieved a polynomial Turing Kernelization for the k-path problem which has a size of $O(k^2)$. This is because we are restricted to use a YES/NO oracle and hence we need do a polynomial-time transformation from the instances in our graph to some other instances that can be answered by the oracle, hence the instance queried by the oracle is bounded by $poly(k)$. It might be less interesting that our result seems be the same as Jansen's work, but in fact it shows little improvement. In the algorithm above we use a YES/NO oracle so as to follow the strict definition of Turing Kernelization. However, if we cheat a bit and are allowed to use a 'smarter' oracle such that can output the maximum edge set that satisfies an edge property directly, then we do not need to transform our instances anymore and we can announce the size of the kernelization is $O(k^2)$. As a comparison, Jansen's work has a size of $O(k^{log_2 3+1})$, even if a smarter oracle is given.

Chapter 5

Extensions and future research

In this chapter we discuss some possible improvements on the Turing Kernel. In section 5.1 we show an approach that enable the k-cycle problem on a planar graph G admits a smaller polynomial kernelization of size $O(k)$, under the condition that G has quasi-4-tree decomposition $(T, \mathcal{X})$ with adhesion only of size of 2, along with given access to an oracle that answers k-weight cycle problem. In section 5.2 we show the difficulty of applying such approach on the general planar graph G'. Finally, we represent the primary conclusions in section 5.3.

5.1 Kernel of smaller instances

In this section we show that there exists a polynomial Turing Kernelization for the planar k-cycle problem with each of instances queried by the oracle has size no larger than $O(k)$, if G is strictly defined.

Before showing the approach, we need to clarify the following definitions first.

Definition 5. A *weighted graph* G_w is described as a triple (V, E, W), where $V(G_w)$ is a vertex set and $E(G_w)$ is an edge set. $W(G_w)$ is a function $w : w(e) \to \mathbb{N}$, for every $e \in E(G_w)$.
Similar to a regular graph, we use $G_w[X]$ denote a sub-weighted-graph obtained by the vertex set $X \subseteq V(G_w)$. A *k-weight cycle* is a cycle that the total weight of its edges is at least k.

Definition 6. Let $G = (V, E)$ be a graph and $G_w = (V', E', W)$ be a weighted graph. If $V = V', E = E'$ and for every edge $e \in E', w(e) = 1$, then we call G_w is a *heavy graph* of G, and G is a *light graph* of G_w.

As a base of the approach, we first argue that a graph G has a k-cycle, if and only if its heavy graph G_w has a k-weight cycle.

Lemma 5.1.1. Let $G = (V, E)$ be a graph and let $G_W = (V, E, W)$ be its heavy graph, then G has a k-cycle, if and only if G_W has a k-weight cycle.

Proof. If G has a k-cycle, then there are k edges form a cycle in G. Since G_W has the same edge set as G, then these k edges still forms a cycle in G_W; Since every edge has weight of 1, these edges forms a k-weight cycle.

If G_W has a k-weight cycle, then there is a cycle with weight k in G_W, since every edge has weight of 1 then there are k edges forms such cycle hence we can find these k edges in G such that forms a k-cycle. □

Lemma 5.1.1 shows the equality of a k-cycle and a k-weight cycle. Now we can try to find a k-weight cycle in a weighted graph instead of finding a k-cycle in a regular graph. We have the following observation similar to Chapter 3:

Observation. Let A, B be a separation of order two in a weighted graph G_w, with $A \cap B = \{x, y\}$. Let $\mathcal{P}_A$ be the maximum-weight xy-path in $G[A]$. If G_w has a k-weight cycle, then either $G[A]$ has a k-weight cycle, or $G[V(\mathcal{P}_A) \cup B]$ has a k-weight cycle.

The observation above shows again, that we can maintain a maximum-weight xy-path in one part of the separation and safely remove other vertices in that part. Now we come to how to find a maximum-weight path that has fixed ends.

Lemma 5.1.2. There is an algorithm that, given an n-vertex weighted graph G_w with distinct vertex x, y, and an integer k, either:

1. Determine whether there is a cycle of weight at least k in G_w, or
2. Determine the weight of the maximum-weight xy-path.

The algorithm runs in polynomial time if there is an oracle that is able to determine a k-weight cycle. The oracle is queried for instance (G'_w, k), where G'_w is a graph obtained from G_w that contains x, y.

Proof. Proof here is quite similar to lemma 3.1.3, but is a bit easier:

Determine k-weight cycle. We call the oracle with instance (G, k), if the oracle answers YES they we output this and halt.

Determine maximum-weight xy-path. We make a sequence of weighted graphs $G_{w0}, .., G_{wk-2}$, where G_{wl} is a heavy graph of G_l mentioned in the proof of lemma 3.1.3. We invoke the oracle by sequence with instance $(G_{w0}, k), .., (G_{wk-2}, k)$. Similarly the smallest index l^* that the oracle answers YES contains a cycle that passes through x, y with weight of exactly k. As we add an xy-path of weight $l^* - 1$ then the maximum-weight xy-path in G has weight of exactly $k - (l^* - 1) \geq 1$. We may only use the length to achieve a Turing Kernel. □

With the access to a k-weight cycle oracle, we can give a Turing Kernel that every instance sent to oracle has size smaller than $O(k)$ in restricted graphs:

Lemma 5.1.3. The planar k-cycle problem has a polynomial Turing Kernel on some specified graph G: it can be solved in polynomial time using an oracle that decides planar k-weight cycle on instances with at most $2k$ vertices and a parameter value k. G should satisfies the following condition: For every edge in its quasi-4-tree decomposition $(T, \mathcal{X})$, the size of it's adhesion is no larger than 2.

Proof. Here we simply ignore the case of size-1 adhesions, as usual. We now show the proof on size-2 adhesions by a similar structure as what we did in chapter 3:

Algorithm 5: Reduce-weight$(G_w, G'_w, A, B, x, y, k)$

Precondition : G'_w is an induced weighted subgraph of G_w, (A, B) is a separator of G'_w with $A \cap B = \{x, y\}$, and F is a minimal separator of G. F can be size 2

Postcondition: The existence of a k - weight cycle is reported, or the graph G'_w is updated by removing all vertices in $A \backslash B$ but an . If G'_w initially has a k-weight cycle, then the deletions preserves this property.

1 Query the k-weight cycle oracle to determine whether $G'_w[A]$ has a k-weight cycle
2 Find the value of l, where l is the length of the maximum-weight xy-path on $(G'_w[A])$
3 **if** *oracle answer YES or line2 reports l has value* $\geq k-1$ **then**
4 | Report the existence of a k-weight cycle and halt
5 **else**
6 | Remove vertices in $A \setminus \{x, y\}$ from A
7 | Add an edge $e = \{x, y\}$ with $W_e = l$. If this edge already exists, set $W_e = l$ if l is larger than the existed weight.
8 **end**

Transform and Decompose. For a planar graph G We first transform it into its heavy graph G_w. As argued, finding a k-cycle in a regular graph is equal to finding a k-weight cycle in its heavy graph. Then in the following we are trying to find a k-weight cycle in G_w instead of finding a k-cycle in G.

Given a weighted graph G_w we first split into its quasi-4-tree decomposition. Note this is always possible since G_w has the same vertex and edge set as G. The resulting tree $(T, \mathcal{X}(i))$ has the same structure as we discussed in Chapter 3 and hence every adhesion is a minimal separator. Then we show the reduction algorithm as Algorithm 5 so as to remove every vertex but only keeps two vertices in the child j of a node $i \in V(T')$, as we make a copy G'_w of G_w and a copy T' of T.

Lemma 5.1.4. Algorithm 5 satisfies its specifications. It calls the k-weight cycle oracle on a graph of at most $|A| + k$ vertices.

Proof. We first show the correctness of the algorithm. Same as what we done in Chapter, if $G'_w[A]$ contains a k-weight cycle or an xy-path of weight at least $k-1$ then we report the existence of k-(weight)cycle directly. If this is not the case, in line 6 and 7 we remove every vertices in $G'_w[A]$ and record the weight of the maximum-length xy-path in the weight of edge $\{x, y\}$. As the maximum weight is maintained after line 7 in $G_w[A]$, we know from the observation that the existence of a k-weight cycle is preserved. Since only x, y are kept after line 7, therefore $|G_w[A]| = 2$ after line 7 terminates.

Since we use a similar technique for finding a maximum-weight path, we query the oracle with instances of order no more than $|A| + k$. □

Algorithm 6 shows the main self-reduction procedure. Since it is totally same as algorithm 2 we skip the proof of its correctness. The only point we need to argue is the size of instances that it sends to the oracle.

Algorithm 6: Kernelize-weight$(G_w, G'_w, (T', \mathcal{X}'), i, k)$

Precondition : G'_w is an induced weighted subgraph of G_w with a tree decomposition $(T', \mathcal{X}')$ of adhesion f size 2, a node i of T' is specified as root.

Postcondition: The existence of a k-weight cycle is reported, or the graph G'_w and the tree decomposition $(T', \mathcal{X}')$ are updated by removing vertices of $\mathcal{X}'(T'[i]) \setminus \mathcal{X}'(i)$, resulting in $|\mathcal{X}'(T'[i])| \leq O(k)$. If G'_w initially has a k - weight cycle, then the deletions preserves this property.

```
1  for each child j of i in T' do
2      Recursively execute kernelize-weight(G_w, G'_w, (T', X'), j, k)
3      Let {x, y} := X'(i) ∩ X'(j)
4      Let A:= X'(T[j]) and B:= (V(G')\A) ∪ F
5      Reduce-weight(G_w, G'_w, A, B, x, y, k)
6  end
7  for each pair (x, y) ∈ (X'(i) choose 2) do
8      while there are distinct children j1, j2 of i in T' such that
         X'(i) ∩ X'(j1) = X'(i) ∩ X'(j2) = {x, y}) do
9          Let A:= X'(T[j1]) ∪ X'(T[j2]) and B:= (V(G')\A) ∪ {x, y}
10         Reduce-weight(G_w, G'_w, A, B, x, y, k)
11     end
12 end
```

Note that algorithm 6 invokes the oracle by line 5 and line 10. As discussed, for every edge in a node $i \in V(T')$ it can be an adhesion of size 2 which connects to one child. Since each of these children has been invoked by algorithm 5 in line 5, each child only contains at most two vertices and hence $|\mathcal{X}'(T'[i])| \leq 3k * 2 + k = O(k)$. As every subtree invoked in line 10 has size of $O(k)$ then by lemma 5.1.4, in line 5 the oracle queries instances of order $O(k) + k = O(k)$, which completes the proof. □

5.2 Difficulties on general planar graph

We now show that the technique used in chapter 5.1 is not applicable when some edges of the quasi 4-tree decomposition of a graph G has adhesions of size 3.

As shown in lemma 3.1.1, when the decomposition has some edges that have adhesion of size 3, we need to maintain 6 maximum-length paths. Saving the weight of maximum-length paths leads to problems, as paths may share the same ends.

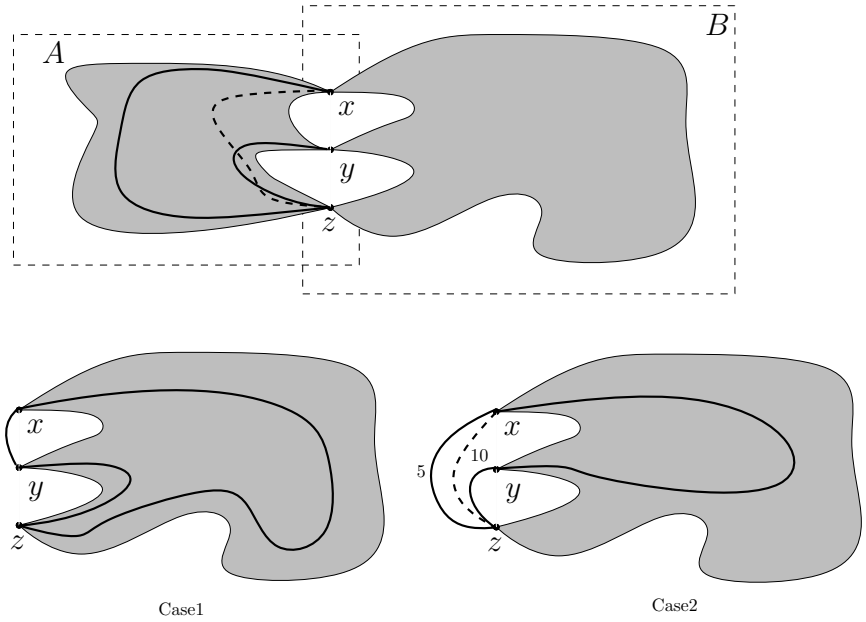


FIGURE 5.1: An example showing why errors may occur when a graph has adhesion of size 3.

Consider the example shown in Fig. 5.1. Let (A, B) be a separation of G_w. Suppose in A we need to maintain a maximum-weight xz-path in $G[A] - y$ (1, dashed) and a maximum-weight xy-path in $G[A]$ (2). Note that path (2) consists 2 sub-paths: its xz sub-path has less weight than path (1) and its zy sub-path intersects with path (1). Now consider how to save the weight of path (2): If we simply save it by adding an xy-edge (shown by case 1) then we miss the information that such path passes through z and the oracle may pick it with a path in B that also passes through z, which is wrong. If we save it by adding an xz-edge(3) and a zy-edge(4), since we can only save path (1) by adding an xz-edge (5, dashed in case 2), then the oracle will combine edge (4) and edge (5) as an xz-path, regardless that the yz-subpath of (2) and path (1) are intersected, which is also wrong.

One simple observation is: Let $\mathcal{P}_1$ be an ab-maximum-length in planar graph G and $\mathcal{P}_2$ be a cd-maximum-length in G, if $\mathcal{P}_1 \cap \mathcal{P}_1 = \{e, f\}$ then we can replace the ef-path in $\mathcal{P}_1$ by $\mathcal{P}_2$, and vice versa, without changing the length of the two paths. Note that if we replace one sub-path by the other then it equals to we remove one sub-path, as the other sub-path has already exist in the graph. Such observation leads to a natural question that, when facing intersected maximum-length paths in $G[A]$, can we replace(and hence) remove some intersected sub-paths, so as to only keep a constant number of vertices of degree larger than 2, remove other verices, and finally we can maintain a Turing kernel with invoke the oracle with instance of order $O(k)$? Unfortunately the answer is also no.

We disprove the idea, by showing that there exist two maximum-length paths $\mathcal{P}_1$ and $\mathcal{P}_2$ in G, with $|\mathcal{P}_1| = |\mathcal{P}_2| = O(n)$, such that $|\mathcal{P}_1 \cap \mathcal{P}_2| = O(n)$ and none of their sub-paths can be replaced safely.

In order to explain why some sub-paths cannot be replaced by others, we first establish the following necessary condition for the replacement:

Lemma 5.2.1. Let Let $\mathcal{P}_1$ be an maximum-length ab-path in planar graph G and $\mathcal{P}_2$ be a maximum-length cd-path in G, let $V = V(\mathcal{P}_1) \cap V(\mathcal{P}_2)$. Let $\mathcal{P}_1' \subseteq \mathcal{P}_1$ be a $v_i v_j$ sub-path of $\mathcal{P}_1$ and $\mathcal{P}_2' \subseteq \mathcal{P}_2$ be a $v_i v_j$ sub-path of $\mathcal{P}_2$. Then we can safely [1] replace $\mathcal{P}_1'$ and $\mathcal{P}_2'$ by each other, only if $V(\mathcal{P}_1') \cap V = V(\mathcal{P}_2') \cap V$.

Proof. Suppose this is not the case, then consider a vertex $v_k \in V$ with $v_k \in \mathcal{P}_1'$ and $v_k \notin \mathcal{P}_2'$. Suppose now we replace $\mathcal{P}_2'$ by $\mathcal{P}_1'$, as $v_k \in V(\mathcal{P}_2) \setminus V(\mathcal{P}_2')$, and $\mathcal{P}_2'$ visits v_k after the replacement then $\mathcal{P}_2$ visits v_k twice after the replacement, hence the new $\mathcal{P}_2$ is no longer a cycle.

Similarly, if we replace $\mathcal{P}_1'$ by $\mathcal{P}_2'$, then the new $\mathcal{P}_1$ no longer passes through v_k after the replacement. Since $\mathcal{P}_1$ is now a path in graph $G - v_k$, it does not preserve that it is still a maximum-length ab-path in $G - v_k$, as the lemma concludes. □
Now we can show the two paths as promised:

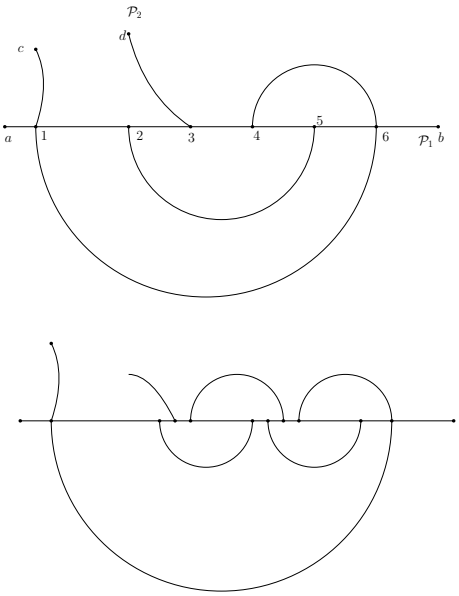


FIGURE 5.2: An example that every such-path in the two paths cannot be replace, and hence no vertices can be folded.

First consider the upper paths in fig 5.2: $\mathcal{P}_1$ passes through vertices $\{1, 2, 3, 4, 5, 6\}$ and $\mathcal{P}_2$ passes through vertices $\{1, 6, 4, 5, 2, 3\}$. Observe that only $4, 5$ sub-paths satisfies the condition in lemma 5.2.1. However they are coincided already and hence nothing can be placed in such a graph. If we call this structure as a 'unit', then a planar graph can contain infinitely many these units so that no sub-paths can replaced (shown as the lower graph), which implies that the idea we raised above is not valid. □

As such idea is proved not valid, the idea of finding a k-cycle via a weighted graph is still interesting. As some information in a regular graph can be compressed in the weight of a weighted graph, we can expect finding a Turing Kernel with even smaller instance sizes by working on weighted graphs in the future.

[1] Here 'safely' means after the replacement, $\mathcal{P}_1$ and $\mathcal{P}_2$ are still simply maximum-length paths

5.3 Conclusion

In this thesis report we go over the designing of Turing Kernelization. We achieved a polynomial Turing Kernel on the planar k-cycle problem with the help of an oracle answers k-cycle decision problem. Using the property of quasi 4-connected graph, we ensure that every instance sent to the oracle has size no larger than $O(k^2)$ which is slightly improved from Jansen's work. We get a same result from his work in planar k-path problem, since we use a NP-completeness transform algorithm in finding a maximum-length paths.

We finally raise a method to achieve a polynomial Turing Kernel of even smaller size by transforming the k-cycle problem into the k-weight cycle problem. Though such kernel is only available for very restricted graphs, we believe the idea of solving a problem in regular graph by solving an equal problem in weighted graph, can be helpful for designing Turing Kernelizations of smaller sizes.

Bibliography

[1] Hans L. Bodlaender. "Kernelization: New Upper and Lower Bound Techniques". In: *Parameterized and Exact Computation*. Ed. by Jianer Chen and Fedor V. Fomin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 17–37.

[2] Hans L Bodlaender, Bart MP Jansen, and Stefan Kratsch. "Cross-composition: A new technique for kernelization lower bounds". In: *arXiv preprint arXiv:1011.4224* (2010).

[3] Hans L. Bodlaender and Arie M.C.A. Koster. "Safe separators for treewidth". In: *Discrete Mathematics* 306.3 (2006). Minimal Separation and Minimal Triangulation, pp. 337–350. ISSN: 0012-365X. URL: http://www.sciencedirect.com/science/article/pii/S0012365X05006126.

[4] Hans L Bodlaender et al. "On problems without polynomial kernels". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, pp. 563–574.

[5] Hans L Bodlaender et al. "Open problems in parameterized and exact computation-IWPEC 2006". In: *UU-CS* 2006 (2006).

[6] Leizhen Cai and John A Ellis. "NP-completeness of edge-colouring some restricted graphs". In: *Discrete Applied Mathematics* 30.1 (1991), pp. 15–27.

[7] Reinhard Diestel. *Graph theory (Graduate texts in mathematics)*. Vol. 173. Springer Heidelberg, 2005.

[8] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. "Incompressibility through colors and IDs". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2009, pp. 378–389.

[9] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[10] Saket Saurabh Fedor V. Fomin Daniel Lokshtanov. "Exact algorithms for the Hamiltonian cycle problem in planar graphs". In: *Operations Research Letters* 34.3 (2006), pp. 269–274. ISSN: 0167-6377. DOI: https://doi.org/10.1016/j.orl.2005.04.013. URL: http://www.sciencedirect.com/science/article/pii/S0167637705000763.

[11] Henning Fernau et al. "Kernel (s) for problems with no kernel: On out-trees with many leaves". In: *arXiv preprint arXiv:0810.4796* (2008).

[12] Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer Science & Business Media, 2006.

[13] Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. "Efficient computation of representative sets with applications in parameterized and exact algorithms". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2014, pp. 142–151.

[14] Fedor V Fomin et al. "Planar F-deletion: Approximation, kernelization and optimal FPT algorithms". In: *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. IEEE. 2012, pp. 470–479.

[15] Martin Grohe. "Quasi-4-Connected Components". In: *CoRR* abs/1602.04505 (2016). arXiv: 1602.04505. URL: http://arxiv.org/abs/1602.04505.

[16] Jochen Harant, Igor Fabrici, and Stanislav Jendrol'. "On Longest Cycles in Essentially 4-connected Planar Graphs". In: *Electronic Notes in Discrete Mathematics* 55 (2016). 14th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW16), pp. 143–146. ISSN: 1571-0653. URL: http://www.sciencedirect.com/science/article/pii/S1571065316301925.

[17] John Hopcroft and Robert Tarjan. "Algorithm 447: efficient algorithms for graph manipulation". In: *Communications of the ACM* 16.6 (1973), pp. 372–378.

[18] Bart M.P. Jansen. "Turing kernelization for finding long paths and cycles in restricted graph classes". In: *Journal of Computer and System Sciences* 85 (2017), pp. 18–37. ISSN: 0022-0000. URL: http://www.sciencedirect.com/science/article/pii/S0022000016301040.

[19] Mamadou Moustapha Kanté et al. "An FPT algorithm and a polynomial kernel for linear rankwidth-1 vertex deletion". In: *Algorithmica* 79.1 (2017), pp. 66–95.

[20] Eun Jung Kim and O-joung Kwon. "A polynomial kernel for block graph deletion". In: *Algorithmica* 79.1 (2017), pp. 251–270.

[21] Stefan Kratsch. "Polynomial kernelizations for MIN F+ π 1 and MAX NP". In: *Algorithmica* 63.1-2 (2012), pp. 532–550.

[22] Stefan Kratsch and Magnus Wahlström. "Preprocessing of min ones problems: A dichotomy". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 653–665.

[23] Stefan Kratsch and Magnus Wahlström. "Two edge modification problems without polynomial kernels". In: *International Workshop on Parameterized and Exact Computation*. Springer. 2009, pp. 264–275.

[24] W. T. Tutte, Curtis Greene, and Thomas Zaslavsky. "The Tutte decomposition". In: *A Source Book in Matroid Theory*. 1986, pp. 267–331. ISBN: 978-1-4684-9199-9. DOI: 10.1007/978-1-4684-9199-9_4. URL: https://doi.org/10.1007/978-1-4684-9199-9_4.

[25] David R. Wood. "On the Maximum Number of Cliques in a Graph". In: *Graphs and Combinatorics* 23.3 (June 2007), pp. 337–352. ISSN: 1435-5914. DOI: 10.1007/s00373-007-0738-8. URL: https://doi.org/10.1007/s00373-007-0738-8.

[26] Chee K Yap. "Some consequences of non-uniform conditions on uniform classes". In: *Theoretical computer science* 26.3 (1983), pp. 287–300.