Utrecht University

Thesis

Computing Science

Department of Information and Computing Sciences

Point Line Segment Cover

Author:
Roland Koimans
r.a.w.koimans@students.uu.nl
6187552

Supervisors:
Prof. Dr. Hans Bodlaender
Dr. Till Miltzow

January 2023

Contents

Abstract

In this thesis, we consider the Point Line Segment Cover problem in a parameterized setting. In this problem, we have to decide for a given set of points in the plane, and an integer k, whether all points can be covered by at most k disjoint non-intersecting line segments. This problem is a more constrained version of the well studied Point Line Cover problem. We introduce new concepts, such as Reserved Line Segments and Hybrid Points, which are used to prove that the Point Line Segment Cover is kernelizable, with a kernel size of $O(k^6)$. This also implies that the Point Line Segment Cover problem is Fixed Parameter Tractable. In this thesis, we also give an algorithm that solves the kernelized instance. The theory has been implemented and experimented with, and through solving various instances it is shown that the kernelization step can significantly decrease the solving time for small values of k.

1 Introduction

In the field of computer science, there are many hard problems for which there does not exist an algorithm that can find an exact solution in a reasonable amount of time. There is often a trade-off between hardness of a problem, time complexity of the algorithm and whether we find an exact solution or an approximation.

In this thesis, we specify ourselves to parameterized problems, which are decision problems with some fixed parameter k. Downey and Fellows [1] have laid the groundwork for parameterized complexity and have defined the complexity class Fixed Parameter Tractable (FPT). FPT-algorithms solve parameterized problems in terms of k, which is often small compared to the total input size. For small values of k, FPT-algorithms are often much faster than general purpose algorithms. Books that give a good overview on respectively parameterized algorithms and complexity and on kernelization are by Cygan et al. [2] and Fomin et al. [3].

A well studied parameterized problem is the Point Line Cover, that asks the question if a set of points in the plane can be covered by at most k lines. The main goal of this thesis is to analyze a variant of this Point Line Cover problem to further expand knowledge of problems in FPT. This variant is the Point Line Segment Cover, which uses non-intersecting line segments instead of lines.

This thesis consists of two parts. The first part is an analytical part, in which we show how the Point Line Segment Cover problem relates to the Point Line Cover, how the Point Line Segment Cover can be kernelized and proven to be Fixed Parameter Tractable. Secondly we discuss a more practical part, for which a kernelization and solving algorithm has been implemented. This allows for experimentation of various values of k on different types of input instances. The results of this experimentation show how the kernelization time and solving time scale in various situations.

Lastly, we define some aspects that can be used for future works, which were out of scope for this thesis.

1.1 Related Work

The starting point of this research is Point Line Cover (PLC), a specific case of the Hyperplane Cover problem. The non-parameterized version of PLC is proven to be NP-hard by Megiddo and Tamir [4] and APX-hard by Brodén et al. [5]. The current best algorithm for parameterized PLC is by Wang et al. [6]. The algorithm works on any parameterized Hyperplane Cover instance and thus can be applied to PLC, where $d = 2$, and achieves a running time of $O^*(k^k/1.35^k)$.

Kratsch et al. [7] discovered a tight lower bound of $\Omega(k^2)$ points on the kernel size of the Point Line Cover. A reduction from Vertex Cover to Point Line Cover was given, which means that literature on Vertex Cover might be relevant to this thesis.

Besides directly related work to the problem, it is important to look at related geometric research to gain a better understanding of the underlying principles that can help to attain the goals of this thesis. For most relevant geometric theory, Computational Geometry by de Berg et al. [8] is an excellent resource for geometric theory in the field.

Since both points and lines have two parameters, namely an x-coordinate and a y-coordinate for points and a slope and an intercept for lines, they can be mapped to each other. This is the notion of Point Line Duality [8]. These transformations allow for more ways to approach the problem. Applied to the Point Line Cover problem, it now becomes the Line Point Cover problem, where instead of covering points with lines, we cover lines with points. For example, a line that covers three points in the primal plane, is a point on the intersection of three lines in the dual plane. Fekete et al. [9] explore this as hitting sets for lines.

Lastly, Langerman and Morin [10] introduced algorithms to solve the Dim-Set-Cover problem, an abstract variant of covering problems, which the Hyperplane Cover, and thus the Point Line Cover problem is part of.

2 Preliminaries

In this thesis three main parameterized problems will be discussed. These problems are defined as follows:

- Point Line Cover
  - **Input:** A set of points P in $\mathbb{R}^2$ and a non-negative integer k.
  - **Question:** Does there exist a set of lines L, such that all points in P lie on at least one line in L, and $|L|$ is at most k?
- Point Line Segment Cover
  - **Input:** A set of points P in $\mathbb{R}^2$ and a non-negative integer k.
  - **Question:** Does there exist a set of disjoint (In this thesis: non-intersecting) line segments L, such that all points in P lie on at most one line segment in L, and $|L|$ is at most k?
- Point Line Segment Cover with maximum length
  - **Input:** A set of points P in $\mathbb{R}^2$, a non-negative integer k and a non-negative float l.
  - **Question:** Does there exist a set of disjoint line segments L, such that all points in P lie on at most one line segment in L, $|L|$ is at most k and the total length of line segments in L is at most l?

One of the main goals of the research is to determine the parameterized complexity of the Point Line Segment Cover problem. In this thesis, these problems are all discussed in their parameterized form. This means that they consist of two parts: a decision problem and a parameter, often denoted as k. Specifically in this thesis, k will denote the amount of line (segments) that can be used, and l denotes the maximum length of the combined line segments that are used in the solution.

Parameterized problems can fall under various complexity classes. The Complexity Zoo [11] provides a good overview of the current defined classes known in the field of computer science. The following complexity classes are relevant to this research:

- Para-NP-Complete, which are problems that are NP-complete for some fixed value of k.
- Slice-wise polynomial (XP), which are all problems that are solvable in the form of $n^{f(k)}$, where n is the input size, f a computable function and k the parameter.
- Fixed Parameter Tractable (FPT), which are all problems that are solvable in the form of $f(k) * n^{O(1)}$, or equivalently $f'(k) + n'^{O(1)}$, where n is the input size, f a computable function and k the parameter.
- A parameterized analogue of NP, W[1] is part of the W-hierarchy. Any problem that is W[1]-hard can be assumed to not be FPT. Problems in this complexity class can often be reduced from clique.

Related to FPT is the notion of kernelization: preprocessing algorithms that run in polynomial time which convert an instance (n, k) to an equivalent instance (n', k'), such that its size is now bounded by a function of k, so $n', k' \leq h(k)$. Equivalent here means that the two instances always have the same solution. Any finite algorithm on (n', k') will then be dependent on k', and thus be in FPT.

To prove that a problem is in FPT, one of the following approaching can be used:

1. Find a way to solve the problem in the form of $f(k) * n^{O(1)}$.
2. Find a way to solve the problem in the form of $f(k) + n^{O(1)}$. (See Theorem 1.)

3. Prove that the problem is kernelizable. (See Theorem 2.)

Theorem 1 *Any problem in FPT, that is any problem that is solvable in $f(k) * n^c$ time, is also solvable in $f'(k) + n^{c'}$ time.*

Proof. We distinguish two cases: either $n \leq f(k)$ or $n \geq f(k)$. If $n \leq f(k)$, it follows that $f(k) * n^c \leq f(k)^{c+1}$. If $n \geq f(k)$, it follows that $f(k) * n^c \leq n^{c+1}$. This means that for whatever value of n, $f(k) * n^c \leq max\{f(k)^{c+1}, n^{c+1}\}$. Since the maximum of two values is at most its sum, $max\{f(k)^{c+1}, n^{c+1}\} \leq f(k)^{c+1} + n^{c+1}$. This means that any problem that is solvable in $f(k) * n^c$ is also solvable in $f'(k) + n^{c'}$. □

Theorem 2 *Any decidable problem in FPT is kernelizable, and any decidable and kernelizable problem is also in FPT.*

Proof. This proof is separated in two parts. First, we show that any kernelizable problem is in FPT. Take any instance I of a kernelizable problem. If it is kernelized to I', then $n' \leq f(k)$. Any finite algorithm g can then be ran on I', resulting in a running time in the form of $g(f(k)) + n^{O(1)}$ and thus is in FPT.

Secondly, any problem in FPT is kernelizable. Assume we have an algorithm A for the problem instance I, and the value of k is given. We can then construct a kernelization algorithm Q as follows: First, we run algorithm A on I for $|I|^{c+1}$ steps. If the algorithm terminates within that time, then we output I with the corresponding Yes/No answer. It is also a possibility that the algorithm does not terminate in $|I|^{c+1}$ steps. In that case we know that $f(k) * |I|^c \geq |I|^{c+1}$. This means that $|I| \leq f(k)$ and thus $|I| + k \leq f(k) + k$, resulting in a kernel of size $f(k) + k$. □

3 PLC to PLSC

3.1 Point Line Cover

The starting point of this thesis is the well-studied Point Line Cover (PLC). Recall the definition:

Point Line Cover

- **Input:** A set of points P in $\mathbb{R}^2$ and a non-negative integer k.
- **Question:** Does there exist a set of lines L, such that all points in P lie on at least one line in L, and $|L|$ is at most k?

The kernelization algorithm for PLC efficiently reduces an instance I to I' with at most k^2 points, which will always yield the same answer as the original instance [7]. This reduction is done by exhaustively applying a reduction rule:

Reduction Rule 1 *If there exists a line with at least $k+1$ points on it, remove all points on this line and decrease k by 1.*

Theorem 3 *Reduction Rule 1 is safe.*

Proof. If in the instance there is a line with at least $k+1$ points on it and we achieve to cover all points, there are two options: either this line is in the solution, or all points on this line have to be covered by separate lines. The latter implies that at least $k+1$ lines are needed to cover all these points seperately. This is however impossible, since only k lines are available. From there it follows that the only possibility in such a situations is to take the line with at least $k+1$ lines in the solution. □

Theorem 4 *If Reduction Rule 1 has been exhaustively applied to an instance, the maximum amount of points is k^2.*

Proof. If Reduction Rule 1 can no longer be applied, it means that any line has at most k points on them. Since there are at most k line segments, each with at most k points on them, the upper bound for a potential YES instance is k^2 points. □

3.2 Point Line Segment Cover

In this thesis, we will study the parameterized complexity of the Point Line Segment Cover problem (PLSC). This problem is a more constrained version of PLC. There are two extra constraints added. Firstly, instead of using lines to cover points, we use line segments to cover points. Secondly, the line segments must be non-intersecting, also called disjoint in this paper. This implies that a point can only be covered by one line segment, whereas in PLC a point can be covered by at least one line.

If a solution to PLC is obtained, it can be transform to a PLSC solution with a larger value for the parameter k. This can give a quick YES answer in cases where a high value of k is available. This transformation is done in two parts. Firstly, all line segments are shortened to line segments. This means that for the transformed version, we need at least k line segments. Secondly, all intersections must be resolved to adhere to the constraints of PLSC. For every intersection, we can take one of the line segments and skip over an intersection, breaking up the line segment in two. This wil then cost an extra line segment per intersection. Formally we define this transformation as follows:

Theorem 5 *Any PLC YES-instance I with parameter k can be transformed to a PLSC YES-instance I' with parameter $k' = k + i$, with k' being the parameter for the PLSC instance, k the parameter for the PLC instance and i the amount of intersections in the PLC solution.*

Proof. Firstly, transform each line to the smallest line segment that still covers all points on that line. The total amount of line segments remains k. Secondly, at every remaining intersection, take one of the line segments that is part of the intersection and split it in two, leaving a gap at the point of the intersection. Each split then results in a new line segment. The total amount of line segments used in the PLSC instance is then $k' = k + i$. Figure 1 is a visual example of how the intersecting line segments can be broken up to adhere to the PLSC constraints.

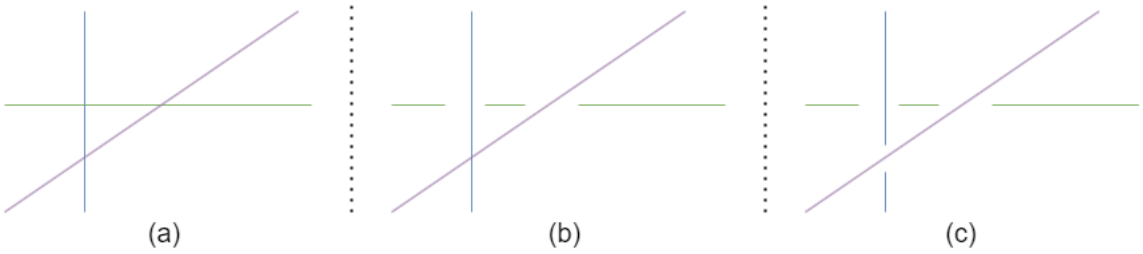

Figure 1: PLC to PLSC

It is important to note that this translation does not give an optimal solution, since line segments are not guaranteed to cover any points at all. An example of this phenomenon is given in Figure 2, where the middle line segment (highlighted in red) does not have any points on it after the application of this algorithm. The upside of this method is that it does provide an upper bound for k', namely $k' = k + k^2$, since the maximum amount of intersections in the PLC instance is k^2.

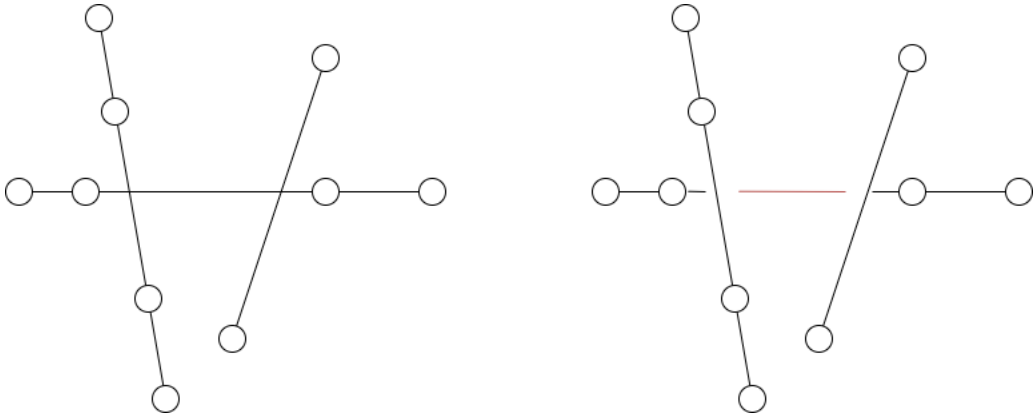

Figure 2: PLC to PLSC, with an empty line segment

3.3 Kernelization

Since PLSC has added constraints compared to PLC, the reduction rules for kernelization for PLC no longer work. However, they can be used as starting point to develop reduction rules for PLSC. To define and prove the kernelization process, some new concepts are defined.

3.3.1 Definitions

Definition 1 *A Reserved Line Segment (RSL) is the shortest line segment that covers all points on a line with at least $k + 1$ points on this line.*

Definition 2 *A Reserved Point is a point that is covered by a Reserved Line Segment*

Definition 3 *An Unreserved Point is a point that is not covered by any Reserved Line Segment.*

In the PLC problem it was shown that a line with at least $k+1$ points on it must be taken. This no longer works in PLSC, since such a line segment can be broken up by a second line segment that would intersect it. However, there are at most $k-1$ line segments that can break up this line segment. This means that at least a part of this line segment must be in the solution. Therefore, we reserve this entire line segment, and figure out in the solving process exactly which parts of this line segment are in the solution. Figure 3 shows an instance with $k = 3$, where two RSLs cover at least 4 points each. It is important to note that at this stage, RSLs may intersect and that not all points in the instance have to be covered.

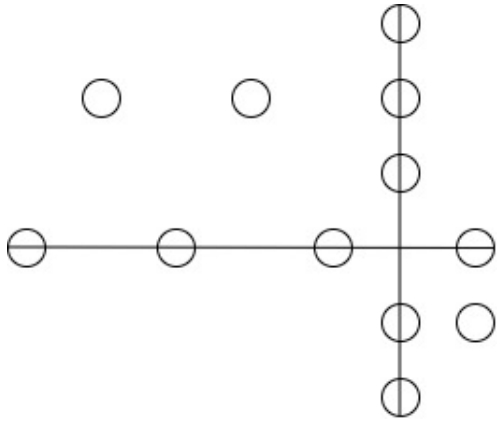

Figure 3: Two Reserved Line Segments in an instance with $k = 3$

Definition 4 *The endpoints of all Reserved Line Segments in the solution are defined as Hybrid Points.*

When a RSL intersects with another in PLSC, one of the two line segments must be adjusted to adhere to the disjunction constraint. A line segment can then be manipulated in two ways: either another line segment breaks the RSL in two by going through it, or another line segment goes through one of the RSL's endpoints, in which case the reserved line segment is shortened. In the second case, we see that endpoints of reserved line segments behave differently, as they are reserved, but can become free if needed, without any extra cost.

Definition 5 *Potential Line Segments are line segments between two unreserved points.*

Definition 6 *Potential Hybrid Points are the points on a Reserved Line Segment that are closest to, but not on, an intersection between a Potential Line Segment and this Reserved Line Segment.*

Potential Line Segment and Potential Hybrid Points will be used to determine what points exactly can be removed from the instance in the kernelization process. It is important to note that the set of Potential Line Segments does not contain all line segments that are viable in the solution, as line segments between an unreserved point and a reserved point can also be valid. Figure 4 gives a visual overview of all definitions for an instance with $k = 3$. Part a shows the Reserved Line Segment in red, part b shows the Hybrid Points that goes with the Reserved Line Segment in blue, part c shows all Potential Line Segments in purple and part d shows all Potential Hybrid Points in yellow.

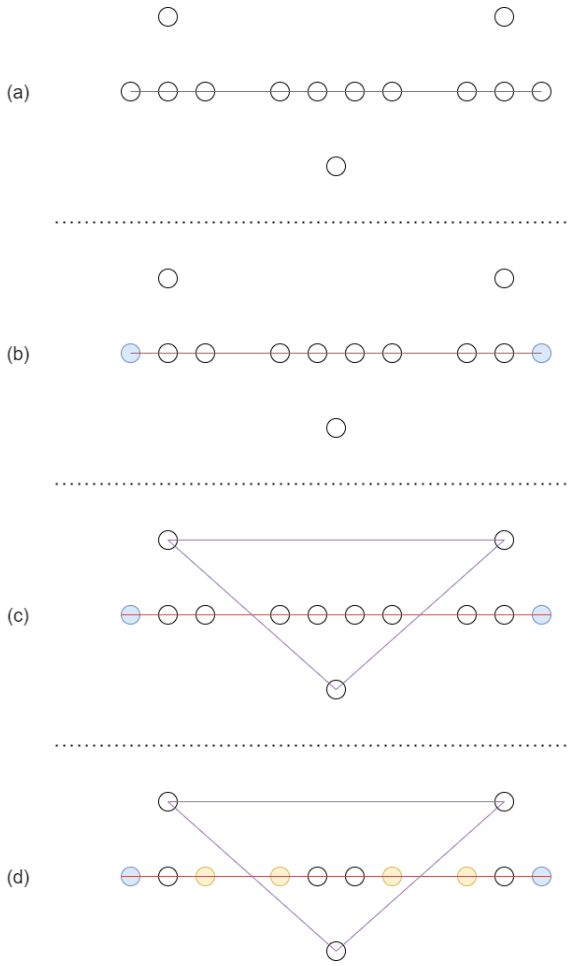

Figure 4: Overview of all definitions. Part a shows the Reserved Line Segment in red, part b shows the Hybrid Points that goes with the Reserved Line Segment in blue, part c shows all Potential Line Segments in purple and part d shows all Potential Hybrid Points in yellow.

3.3.2 Reduction Rules

A kernel for PLSC will be defined by reduction rules that transform an instance I to I'. A reduction rule is *safe* when the answer to I' is the same as the answer to I. To prove that the kernel is correct, all reduction rules must be proven safe.

Reduction Rule 1 *If there exists a line with at least $k+1$ unreserved points on it, a Reserved Line Segment is constructed out of all the points on this line.*

Theorem 6 *Reduction Rule 1 is safe.*

Proof. This Reduction Rule only changes information about the instance and does not remove any. Some unreserved points may change to a reserved points, but a naive algorithm will not use this property to solve the instance. Therefore, the answer to an instance with Reduction Rule 1 applied to it will have the same answer as the same instance without Reduction Rule 1 applied to it. This means that Reduction Rule 1 is safe. □

After exhaustively applying Reduction Rule 1, the instance already has an upper bound on unreserved points. Exceeding this upper bound means that an answer can no longer be found.

Theorem 7 *If Reduction Rule 1 has been applied exhaustively on an instance, the instance has at most k^2 unreserved points.*

Proof. For every line that has at least $k + 1$ points on it, Reduction Rule 1 has created an RSL. Since this Reduction Rule has been applied exhaustively, the maximum amount of points on a line is k. Because we can use at most k line segments, the maximum amount of unreserved points is thus $k * k = k^2$. □

Reduction Rule 2 *Remove all points that are at least k points away from any (Potential) Hybrid Points on a Reserved Line Segment.*

Consider a simple instance where Reduction Rule 1 has been applied. The instance consists of at most k^2 unreserved points, divided by a long RSL with many points on it, a magnitude bigger than k. Because of size of this RSL in this example, Potential Line Segments with one endpoint on one side of the RSL and the other endpoint on the other side of the RSL will always intersect with the RSL. The first question to answer is why it would be beneficial to cross the reserved line segment. Figure 5 shows an example how extra line segments can cross the reserved line segment "for free", once one line segment has crossed. As shown, this means that per crossing Potential Line Segment, there are at most 3 relevant points, from which 2 Potential Hybrid Points. Any crossing line segment beyond those points must split the reserved line segment again.

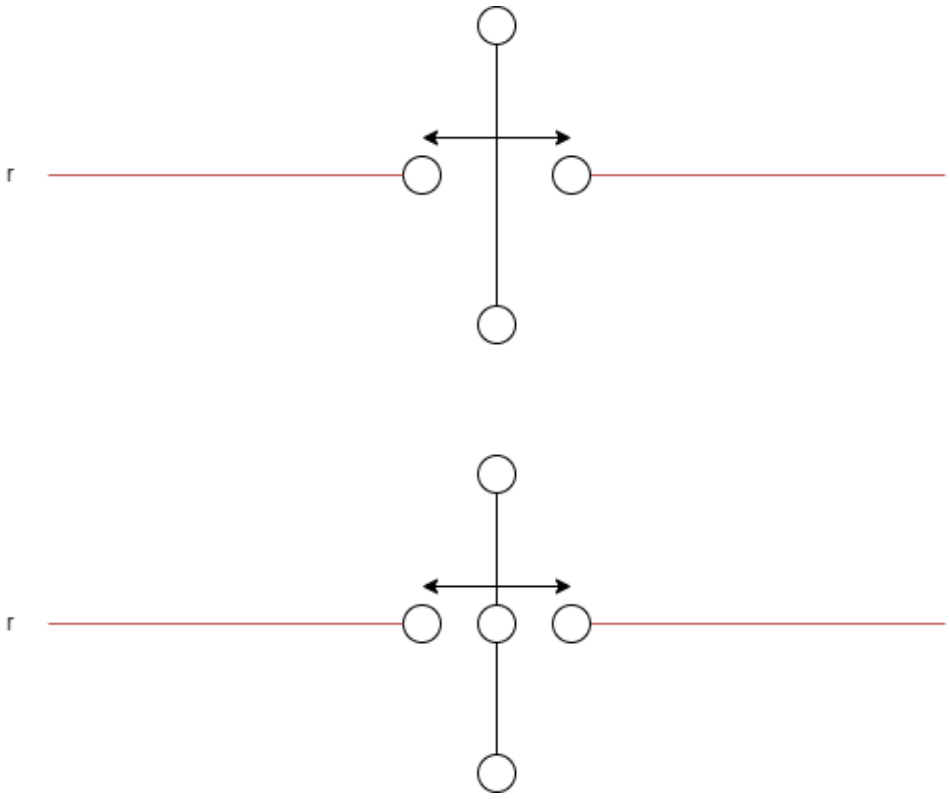

Figure 5: Range where a second line segment can cross without costing an extra line segment

If the Potential Line Segments and their respective Potential Hybrid Points are drawn, it can be noted that there can be sections on the RSL that are many points away from the nearest (Potential)

Hybrid Point. This means that the only way for a regular line segment to interact with points on this section is by connecting an unreserved point with a reserved point. This reserved point must be a (Potential) Hybrid Point, because otherwise the RSL must be split in two, and this costs an extra line segment. Choosing this line segment then makes the point adjacent to the newly covered reserved point Hybrid. Since we have at most k line segments, all points that are at least $k-1$ points away from a (Potential) Hybrid Point cannot be part of any other line segment than the RSL, and therefore can be removed.

Theorem 8 *Reduction Rule 2 is safe.*

Proof. Using Figure 6 as example, we can distinguish a couple cases. The line segment r has many points on it, where only the points next to intersections and the endpoints are shown. p_1, p_2 and p_3 are points that do not lay on the reserved segment. s_1 and s_2 are Potential Line Segments. The section r_2 is encapsulated by two potential crossings s_1 and s_2. In the actual solution, we can have three cases:

1. Both crossings are taken. Between s_1 and s_2 there are no points, because this would result in extra potential crossings and r_2 would be encapsulated by these line segments instead. This means that besides the two endpoints that define r_2, all other points on that line segment can be removed without changing the solution.

2. One of the crossings is taken. This means that either points to the right of s_2 or the left of s_1 can still interact with r_2 by connecting with one of the endpoints. It is exclusive to the endpoints, because this would shorten r_2 instead of splitting it in two, resulting in a worse solution. This means that points at least k points away from these endpoints can never be in the solution for this situation and can be safely left away.

3. None of the crossings is taken. In this case the line segment becomes limited by either endpoints of the entire reserved line segment or another taken line segment that crosses the reserved line segment, in which case situation 1 or 2 can be applied again in a different context.

Point 2 is the most restrictive of the situations. Generalizing this to all situations where one crossing is taken, we can indeed remove all points that are more than k points away from a (Potential) Hybrid Point without changing the solution. Therefore, Reduction Rule 2 is safe. □

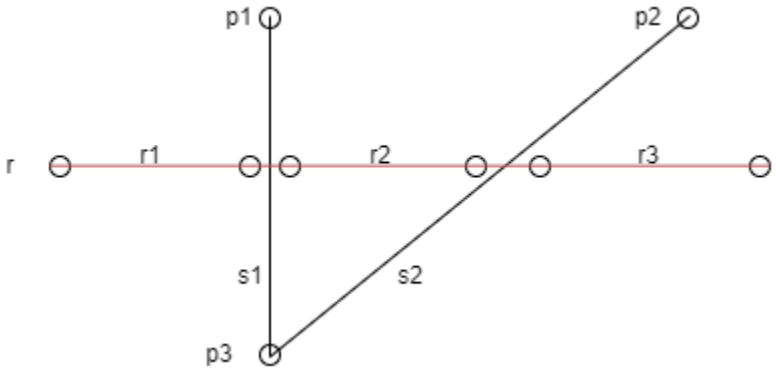

Figure 6: RSL with Potential Line Segments

Theorem 9 *A kernel with $O(k^6)$ points can be created by exhaustively applying Reduction Rule 1 and Reduction Rule 2.*

Proof. Applying Reduction Rule 1 results in a maximum of k Reserved Line Segments with at most k^2 unreserved points. In the worst case, half of those unreserved points lie on either side of all Reserved Line Segments, in which case there are $O(k^4)$ sections per Reserved Line Segments created by Potential Line Segments. Reduction Rule 2 caps the length of these sections at $O(k)$ points per section. This means that there are at most $O(k^5)$ points per Reserved Line Segment. Because we have at most k Reserved Line Segments, the total amount of points on Reserved Line Segments is $O(k^6)$ and $O(k^2)$ unreserved points. Reduction Rule 1 and 2 can be applied because they are proven safe. Therefore PLSC admits a kernel of $O(k^6)$ points. □

For a problem to be properly kernelizable, it not only needs to depend on k as shown in the previous theorem, but the kernelization algorithm also needs to run in polynomial time. Therefore the runtime of the kernelization algorithm has to be analyzed.

Theorem 10 *There is an algorithm that gives a kernel with $O(k^6)$ points for the Point Line Segment Cover problem in $O(n^3)$ time.*

Proof. Using the previously explained proofs and observations, we can kernelize a PLSC instance in the following way:

1. Check for line segments with at least $k+1$ points on it, and mark these as reserved line segments. Line segments with at least 2 points on them that are a subset of a larger line segment are considered duplicate and should be discarded. If there are more than k reserved line segments, return NO as answer to the problem. If there are exactly k reserved line segments and still one or more unreserved points, return NO as answer to the problem.
2. If there are still unreserved points and the previous step did not give a definite NO answer, count the amount of points. If there are more than k^2 unreserved points left, return NO as answer to the problem.
3. Remove the excess points by drawing a line segments between every two unreserved points and see where they intersect with reserved line segments. Any points on reserved line segments that are k points away from either these intersections or intersecting reserved line segments on either side can be removed.

In step one, for every line segment defined by two points, we have to check every point to see if it is on the line segment between them. This means this step takes $O(n^3)$ time. Step two requires to check every point to see whether they are on a reserved line segment, therefore this step takes $O(n)$ time. The third step requires a constant time check for every time one of the at most k^2 line segments intersects with at most k reserved line segments, so k^3 intersection checks. All points on the reserved line segments will be counted once, so this step takes $O(n)$ time. In total, this results to $O(n^3)$ time, which is polynomial. □

Now that it is proven that PLSC is kernelizable, it can be concluded that PLSC is in FPT:

Theorem 11 *The Point Line Segment Cover Problem is Fixed Parameter Tractable.*

Proof. Theorem 2 has proven that any problem that is kernelizable, is also Fixed Parameter Tractable. Theorem 9 and Theorem 10 have proven that the Point Line Segment Cover Problem is kernelizable. Therefore, it can be concluded that the Point Line Segment Cover Problem is indeed Fixed Parameter Tractable. □

3.4 Solving the kernelized instance

Previous section has shown that there is an upper bound to the amount of unreserved points in a kernelized instance. There is also a lower bound to the amount of points in an instance:

Theorem 12 *Any Point Line-Segment Cover instance with $n \leq 2k$ points is always a YES-instance.*

Proof. If all points are sorted lexicographically on ascending x-coordinate and then y-coordinate, then we can we can do the following: take the first uncovered point and draw a line segment that covers this point and the next one. If multiple points have the same x-coordinate, then enforce an ordering through ascending y-coordinate. If n is uneven, then an additional line segment is needed to cover only the last point. This way, there is never another point between these two points and therefore every line segment cannot intersect with another. Every line segment has the potential to cover two points, and thus any instance with $n \leq 2k$ points is a YES-instance. A visual example of this proof is given in figure 7. □

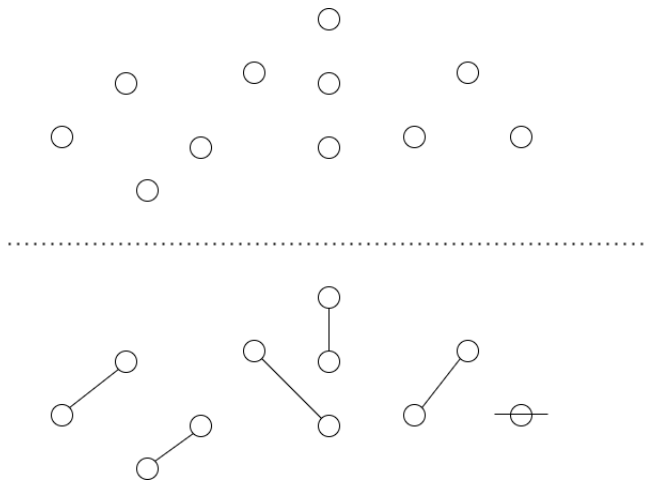

Figure 7: Upper bound

When the instance falls between the two bounds, the solution must be found by searching through a bounded search tree (BST). This is done in two stages. Firstly, if the instance has been kernelized, some line segments can be tagged as reserved. These are not guaranteed to adhere to the disjunction constraint of the problem and may intersect. In this stage the kernelized instance is taken as root of the tree, and there is a new branches per intersection. If there are multiple intersections, then this results in multiple layers of the tree.

Once the intersections have been resolved and the instances adhere to the constraints, all unreserved points have to become covered by new line segments. The set of possible line segments is then any line segment between two unreserved and uncovered points or between an unreserved uncovered point and a reserved point. For every line segment in this set, a new branch is made. Any line segment that has any of the just covered points as endpoint is then removed from the set of possible line segments, as well as any line segment that intersects with an already placed line segment. It is important to note that line segments where both endpoints are the same point is possible. The length of such a line segment is infinitely small, or 0 in the implementation. An empty set means that all points are covered.

Whenever a chosen line segment hits or intersects with a reserved line segment, the reserved line segment must be adjusted to ensure no constraints of PLSC are violated. Where this intersection on the reserved line segment happens, dictates the action that must be taken:

- On a point: Separate the reserved line segment in two, where each reserved line segment has an original endpoint and a new endpoint, which is the point adjacent to the point that is on the

intersection. The point on the intersection is unreserved and covered by the newly chosen line segment.

- Between two points: Separate the reserved line segment in two, where each reserved line segment has an original endpoint and a new endpoint, which is the point closest to the intersection on either side.
- On an endpoint of the reserved line segment: Shorten the reserved line segment and let the new line segment cover this point.
- On a point on a reserved line segment that consists of only one point: Remove the reserved line segment and let the new line segment cover this point. This is a neutral action in terms of change in line segment usage.

3.5 Point Line Segment Cover with Maximum Length

Another restriction that can be imposed on PLSC is that besides only using a maximum of k line segments, the total length of all line segments together can be at most l. This variation is chosen to give some context to the results of PLSC. For example, if a solution is possible for PLSC, usually not the entire tree has to be searched for the solution. However, if there is no solution due to the length constraint, we can see how much time is needed to go through the entire tree for this specific instance.

To compare the two, PLSC with maximum length also needs to be proven to be in FPT.

Theorem 13 *The Point Line Segment Cover with Maximum Length is Fixed Parameter Tractable.*

Proof. The kernelization method for the regular PLSC can be reused, as it is constructed with the property that no potential solution is removed. This means that the Reduction Rules for regular PLSC are safe and a kernel can be constructed. Therefore, this variant is also kernelizable and because of that in FPT. □

The length of line segments are calculated using Pythagoras' Theorem. The addition of roots is defined to be in CH [12], which is in PSPACE [11]. To limit the complexity for the scope of this thesis, the root is rounded to a specific decimal, or in the case of the implementation, the value is stored within a float. It is important to note that this means that we no longer have an exact solution, but an approximation. However, the chance that this approximation changes the outcome from the actual outcome is extremely low, which is only when the parameter l is precisely between the approximation and actual answer.

4 Implementation

4.1 Instance Generation

Instance generation plays an important role for experimentation with PLSC. Most of the theory assumes many points are colinear, especially since reserved line segments only occur with more than k points on a line. Therefore, generating random points is not a viable options. The chance that three points are colinear is already extremely low.

Therefore, the goal of instance generation is to create instances that are meaningful and are able to display situations that are described in the theory of PLSC. There are two ways to achieve this goal. The first one is to craft instances by hand. This is useful to examine edge cases and to observe the behavior of only one variable between instances. An example of this would be to make multiple identical instances, with the only difference being the amount of points on a specific line.

The second way of instance generation is to automate the process with some randomness in mind. This is useful to observe the general behavior of the theory and to avoid the bias that comes with handcrafting instances. For this thesis, these instances are mostly used to ensure the program works as intended. The following methods of instance generation have been implemented:

- **Random**

  Input: p total amount of points.

  Restrictions: $p > 0$.

  Description: Completely random points. This serves as a control instance.

- **Sometimes Co-linear**

  Input: p total amount of points. x percentage.

  Restrictions: $p \geq 2$, $100 \geq x \geq 0$.

  Description: Start with 2 random points. The next point has a x percent chance to be placed randomly and a $100 - x$ percent chance to be placed on or as an extension of a line segment that can be drawn between any two of the previous points.

- **Always Co-linear**

  Input: x starting points, p total amount of points.

  Restrictions: $x \geq 2$, $p \geq 2$.

  Description: Start with x random points. The next point will be place on or as an extension of a line segment that can be drawn between any two of the previous points.

- **Extension With Limited Direction**

  Input: p total amount of points, x distance.

  Restrictions: $p > 0$, $x > 0$.

  Description: Start with a random point. The next point will choose a random point and place a new point in one of the following directions relative to the chosen point: $(+x, 0)$, $(+x, +x)$, $(+x, -x)$, $(0, +x)$, $(0, -x)$, $(-x, 0)$, $(-x, +x)$, $(-x, -x)$. If there is already a point at this place, choose a new random point and repeat the same process.

- **Reserved Line Segment Generator**

  Input: p total amount of points outside of reserved line segments, x amount of reserved line segments, k reserved line segment parameter.

  Restrictions: $p \geq 0$, $x > 0$, $k \geq 2$.

Description: Create reserved line segments by placing two random points and extending the line segment defined by those two points randomly until the line segment is of size somewhere between $k+1$ and k^3. Finally, place p random points not on any line segments.

- **Reserved Line Segment Generator With Forced Crossing**

  Input: p total amount of points outside of reserved line segments, x amount of reserved line segments, k reserved line segment parameter.

  Restrictions: $p \geq 0$, $x > 0$, $k \geq 2$.

  Description: Create reserved line segments by placing two random points and extending the line segment defined by those two points randomly until the line segment is of size somewhere between $k+1$ and k^3. Any reserved line segment after the first one should have a point on any other reserved line segment as one of the initial points. Finally, place p random points not on any line segments.

Within the implementation, all coordinates have integer values. This is so that no intersection between a line segment and a point is lost due to floating point inaccuracies. The downside of this decision is that the interactions between line segments are limited due to the limited amount of point placement. Another downside is that when placing points on a line, the next point on that line with integer coordinates may lay very far away compared to the other points in the instance.

4.2 Kernelization

After an instance has been generated or opened from a file, the instance can be kernelized following the reduction rules described earlier in this thesis. In essence, the kernelization step in the implementation does the following after being supplied the parameter k:

1. Locate all reserved line segments and tag them as such.
2. Remove excess points from reserved line segments if they exist.

Once the kernelization step is complete, all that remains is the set of reserved line segments, the set of possible line segments to choose from and at most k^2 points tagged as non-reserved. If this number of non-reserved points is greater than k^2, then a solution is no longer possible and a NO answer can already be returned.

4.3 Solving

Given an instance, whether kernelized or not, will be solved with a bounded search tree. All branches can be represented as a set of line segments between two uncovered unreserved points and one uncovered unreserved point and a uncovered reserved point. Once a line segment is taken, all line segments in this set that have any of the just covered points as endpoint are then removed. The depth of the tree is bound by k, as we can have no more than k line segments. Once the set of possible line segments is empty, all points are covered. In the normal variant of the problem, if this is the case, and the amount of used line segments is still equal or less than k, an answer is found. In the variant of the problem where there is also a length parameter, the length is also checked in this moment.

4.3.1 BFS vs DFS

The two main ways to traverse the tree is in a breadth-first or depth-first fashion. For this implementation, depth first is chosen. This is done for two reasons. The first one is that the tree is extremely broad, since the branching factor is high. Using a data structure for breadth-first would result in an immense amount of objects needing to be stored. With depth-first, many objects can quickly be discarded once the upper bound of k line segments has been reached. The second reason is to quickly

return YES if there are more line segments available than needed. For this problem, any valid solution suffices, not just the best one. Therefore, depth-first is more likely to find a solution quicker if there are many solutions available.

5 Experiments

PLSC is proven to be kernelizable and in FPT. It is interesting to see whether the kernelization step actually speeds up solving the problem, as well as observing how the solving time scales with certain variables.

The following instances, constructed in various ways, are tested:

- Instances with only a variable amount of random points;
- Instances with a variable amount of points on a single reserved line segment;
- Instances with a fixed single reserved line segment and a variable amount of random points that don't lie on the reserved line segment;
- Instances with a static amount of random points with between them a variable amount of reserved line segments of fixed size;

These instances can be tested in various way. The kernelization step can be enabled or disabled and the length parameter can be included or excluded. Using a combination of these options gives a good overview of the performance of the algorithm.

Since the problem scales quickly in time, a timeout of 90 minutes is set per instance. This is enough to give a good overview of the scaling in time for smaller problems, without spending too much time for all instances combined.

6 Results

The instances described in the previous section have been kernelized and solved by incrementing k by 1, starting with $k = 1$. In this section only fragments of the results are shown. The full tables can be found in the appendix.

6.1 Test Instances

6.1.1 Random points

These test instances consist of purely randomly generated points. The idea of these instances is for it to serve as a control group, as the kernelization step cannot produce much of value. The following table shows the results for instances with 1, 2, 11 and 12 random points. The instance with 13 points would take more than 90 minutes and return a timeout.

Table 1: Instances with only randomly generated points

Points	k	Answer	Inst Cons.	Kernelization Time	Solving Time	NK Inst Cons.	NK Solving Time
1	1	true	2	00:00:00.0039501	00:00:00.0026792	2	00:00:00.0037121
2	1	true	1	00:00:00.0005511	00:00:00.0001764	4	00:00:00.0006234
11	1	false	1	00:00:00.0015330	00:00:00.0000026	2739	00:00:00.0279661
	2	false	1	00:00:00.0004836	00:00:00.0000008	71031	00:00:00.6442016
	3	false	1	00:00:00.0002925	00:00:00.0000007	1168887	00:00:09.3151399
	4	false	12077487	00:00:00.0003170	00:01:32.4629718	12077487	00:01:28.2357634
	5	false	77529087	00:00:00.0005704	00:09:26.1007534	77529087	00:09:03.1756636
	6	true	1041667	00:00:00.0006890	00:00:06.7658626	1041667	00:00:06.6883672
12	1	false	1	00:00:00.0020632	00:00:00.0000025	3741	00:00:00.0443673
	2	false	1	00:00:00.0002960	00:00:00.0000008	112797	00:00:01.1714270
	3	false	1	00:00:00.0002822	00:00:00.0000007	2216493	00:00:20.6964083
	4	false	28627533	00:00:00.0003087	00:04:15.1631630	28627533	00:04:08.1586494
	5	false	243245133	00:00:00.0006063	00:34:28.9053410	243245133	00:32:56.8814210
	6	true	66493665	00:00:00.0005639	00:08:47.8440194	66493665	00:08:20.5858481

As seen in the table, there is no clear difference between solving the kernelized version and non-kernelized (NK) version. For higher values of k, an equal amount of instances have to be considered, because if there are no reserved line segments found, the instance remains unchanged by the kernelization process. For the instances with 11 and 12 points, only a singular instance has to be considered after kernelization. This is due to the fact that with kernelization, we have an upper bound of at most k^2 unreserved points. If there are more unreserved points than that, then the answer is immediately known. The final observation of this table is that when the answer is true, for example for $k = 6$ in the instances with 11 and 12 points, the amount of instances considered and solving time is significantly less than $k = 5$, where the answer was still false. This can be explained by the fact that with $k = 5$, the entire solving tree has to be traversed to make sure there is no solution. For the $k = 6$ variant, the search stops when a solution is found, which can happen a lot earlier. The time saved by finding a solution early can be measured by adding the extra length constraint, which will be discussed in a later section.

6.1.2 Fixed random points, variable length of one reserved line segment

The following table shows some instances that have a single reserved line segment and three points not on the line. Every instance the reserved line segment gets lengthened with 20 extra points.

Both the kernelized and non-kernelized version have been considered for these instances. The non-kernelized version scales rapidly. With 40 points on the reserved line segment, it already takes 4 hours for $k = 2$. The rest of the table for non-kernelized version has been left empty, as it simply takes too long to calculate. The amount of instances considered and the solving time for the kernelized version

Table 2: Fixed random points, variable length of a single reserved line segment

Points on RSL	k	Answer	Inst. Cons.	Kernelization Time	Solving Time	NK Inst. Cons.	NK Solving Time
20	1	false	1	00:00:00.0326929	00:00:00.0014378	35297	00:00:01.4171314
	2	false	1	00:00:00.0135501	00:00:00.0000046	2978561	00:01:48.0381747
	3	true	198	00:00:00.0116515	00:00:00.0055761	2448915	00:01:18.2698268
40	1	false	1	00:00:00.4804957	00:00:00.0000123	369797	00:01:05.6552182
	2	false	1	00:00:00.4101423	00:00:00.0000181	96637979	04:04:23.3483077
	3	true	255	00:00:00.3613198	00:00:00.0061483		
100	1	false	1	00:00:37.6238518	00:00:00.0000553		
	2	false	1	00:00:34.1135616	00:00:00.0000358		
	3	true	255	00:00:33.4502654	00:00:00.0058553		

remains roughly the same. This is because the solving process always remains the same, namely one reserved line segment (with an upper bound of points due to the kernelization process) and three random points. The kernelization step scales reasonably with the amount of points on the reserved line segment. Finally, for $k = 1$ and $k = 2$, only one instance has to be considered due to the fact that one goes to the reserved line segment and more than one line segment is needed for the rest of the points.

6.1.3 Fixed random points, fixed length of variable amount of reserved line segments

The following table consists of instances with two points with the same y-coordinate, with a variable number of vertical reserved line segments consisting of 10 points, placed between the two points.

Table 3: Fixed random points, fixed length of variable amount of reserved line segments

RLS	k	Answer	Instances Considered	Kernelization Time	Solving Time
1	1	false	1	00:00:00.0079234	00:00:00.0016385
	2	false	183	00:00:00.0012320	00:00:00.0035731
	3	true	15	00:00:00.0016307	00:00:00.0006804
2	1	false	1	00:00:00.0229842	00:00:00.0022208
	2	false	1	00:00:00.0032824	00:00:00.0000136
	3	false	525	00:00:00.0023773	00:00:00.0068482
	4	true	27	00:00:00.0014971	00:00:00.0020048
3	1	false	1	00:00:00.0914007	00:00:00.0072018
	2	false	1	00:00:00.0149651	00:00:00.0003459
	3	false	1	00:00:00.0046629	00:00:00.0000146
	4	false	939	00:00:00.0032979	00:00:00.0206848
	5	true	38	00:00:00.0021205	00:00:00.0040271
8	1	false	1	00:00:03.5307030	00:00:00.2602055
	2	false	1	00:00:00.5423969	00:00:00.0087571
	3	false	1	00:00:00.1885382	00:00:00.0013511
	4	false	1	00:00:00.0833153	00:00:00.0003018
	5	false	1	00:00:00.0415730	00:00:00.0001700
	6	false	1	00:00:00.0211080	00:00:00.0001110
	7	false	1	00:00:00.0099273	00:00:00.0000960
	8	false	1	00:00:00.0050074	00:00:00.0000535
	9	false	4149	00:00:00.0060971	00:00:00.2708256
	10	true	93	00:00:00.0041590	00:00:00.0350020

Due to the high amount of reserved line segments, the instances considered for low values of k is only one. For example, for 8 reserved line segments, it is already known that $k >= 9$ is required. Furthermore, these examples highlight a problem with starting with low k values. Because of the greater amount of potential reserved line segments with smaller k values, kernelization takes significantly more time. This can be seen clearly in the case with 8 reserved line segments and $k = 1$, where kernelization takes 3 seconds, while in the same instance with $k = 8$ kernelization takes less than a millisecond.

6.1.4 Variable random points, fixed length of one reserved line segment

The following table consists of instances with a reserved line segment of 60 points, and a variable amount of random points around it. With the fixed reserved line segment, and random points that

Table 4: Variable random points, fixed length of one reserved line segment

Random Points	k	Answer	Instances Considered	Kernelization Time	Solving Time
1	1	false	1	00:00:03.0792969	00:00:00.0014401
	2	true	2	00:00:02.7430270	00:00:00.0026466
2	1	false	1	00:00:03.0840361	00:00:00.0000216
	2	true	10	00:00:02.7586321	00:00:00.0002956
3	1	false	1	00:00:03.1663195	00:00:00.0000224
	2	false	224	00:00:02.7553824	00:00:00.0013724
	3	true	16	00:00:02.6375758	00:00:00.0007380
8	1	false	1	00:00:03.8059171	00:00:00.0000264
	2	false	1	00:00:02.8101017	00:00:00.0000016
	3	false	400941	00:00:02.7274087	00:00:29.2645286
	4	false	11348595	00:00:02.5175227	00:13:03.3450598
	5	true	6553737	00:00:02.4461500	00:06:31.6762935
9	1	false	1	00:00:03.9473646	00:00:00.0000274
	2	false	1	00:00:02.7573878	00:00:00.0000019
	3	false	670288	00:00:02.6245817	00:00:56.2546237
	4	false	21636804	00:00:02.5294348	00:28:51.7106491
	5	TIMEOUT			

have a minimal chance to be co-linear, the kernelization time is roughly equal. Like in the previous section, lower k values tend to take a little more time due to the amount of potential reserved line segments that have to be considered. The solving time again scales heavily with the amount of random points.

6.2 Length constraint

As seen in the previous result sections, having a solution in the bounded search tree can significantly speed up the solving time. By adding a length constraint using a k value for which a solution exists, the time saved by this phenomenon can be displayed. For each category described in the previous sections, one instance is taken and solved again with different values of l. The first value is always 0. This means that the entire tree has to be traversed. The second value of l is the maximum float value. This is essentially the same as removing the constraint, since the length in these instances will always be smaller than this value. Comparing the two then shows how much of a speed up finding a solution earlier in the tree is. For some instances a third value of l is added, which is the length of the solution with minimal length, a so called 'optimal' solution. Where this third value is absent, the instance with float.MaxValue as l-value already resulted in an optimal solution.

As shown in all of these instances, the solving time for a solution is only a fraction of the solving time for an instance where there is no solution. The kernelization time remains the same. This is because the full kernelization algorithm always has to be completed.

Table 5: Instances with added length constraint

Instance	k	l	Answer	Inst. Cons.	Kernelization Time	Solving Time
6 points	3	0	false	3928	00:00:00.0040800	00:00:00.0189259
	3	float.MaxValue	true	448	00:00:00.0004098	00:00:00.0019966
	3	738	true	1576	00:00:00.0004391	00:00:00.0058513
3 points, 1 RSL of 60 points	3	0	false	4614	00:00:10.9776450	00:00:00.0638206
	3	float.MaxValue	true	255	00:00:10.8589252	00:00:00.0051503
3 points, 6 RSL of 10 points	8	0	false	19456	00:00:00.0092781	00:00:00.7590744
	8	float.MaxValue	true	71	00:00:00.0027998	00:00:00.0203413
6 points, 1 RSL of 40 points	5	0	false	35000712	00:00:02.3212540	00:21:52.9389384
	5	float.MaxValue	true	2295	00:00:02.3130146	00:00:00.1220677
	5	1058	true	84893	00:00:02.3450554	00:00:02.8894118

7 Conclusions

In this thesis the Point Line Segment Cover problem (PLSC), a more constrained version of the Point Line Cover problem (PLC), is analyzed. In particular, PLSC is proven to be kernelizable, and therefore Fixed Parameter Tractable (FPT). The kernelization method is based on two main ideas. Firstly, line segments with at least $k + 1$ points on them can be tagged as reserved. At least a part of this line segment must be in the solution, if one exists. Reserving line segments exhaustively puts an upper bound of k^2 unreserved points on the instance. Secondly, endpoints of reserved line segments and already chosen line segments can be tagged as hybrid points. These points are covered, but can be freed by shrinking the line segment by one point. Since crossing a reserved line segment with another line segment is a costly operation in terms of line segment usage, this is only done if enough hybrid points can be used afterwards. Combining these ideas results in a kernel of at most $O(n^6)$ points. The kernel can then be solved by traversing a bounded search tree.

Both the kernelization algorithm and solving algorithm are implemented, and various instances have been created and solved using this implementation. A significant speed up in solving speed can be seen between kernelized and non-kernelized instances when the instance has more or longer reserved line segments. The kernelization time scales slowly with amount of reserved line segments and reasonably with the amount of points on a line segment. The solving time is mostly dependent on the amount of unreserved points in the instance.

Lastly, when a solution is possible, there is a high chance that only a part of the solving tree has to be traversed. To that end, another constraint was added, namely the total line segment length l. Varying the value of l has shown that there is a significant decrease in solving time when the solution can be found without traversing the entire tree.

8 Future Work

There are still some aspects where efficiency can be gained, but were outside of the scope of this thesis. As seen in the result section, a lot of time can be saved traversing the solving tree more sophisticated. However, in this thesis this only has been done naively.

Furthermore, in the proposal of this thesis some extra constraints for PLSC were introduced, but not solved. These variants are:

- Point Line-Segment Cover with limited intersections
  - **Input:** A set of points P in $\mathbb{R}^2$, a non-negative integer k and a non-negative integer i.
  - **Question:** Does there exist a set of line segments L, such that all points in P lie on at most one line segment in L, $|L|$ is at most k and all line segments in L are disjoint, with the exception that line segments may intersect at most i times?
- Partial Point Line-Segment Cover
  - **Input:** A set of points P in $\mathbb{R}^2$, a non-negative integer k and a non-negative integer p.
  - **Question:** Does there exist a set of line segments L, such that at least $|P| - p$ points in P lie on at most one line segment in L, $|L|$ is at most k and all line segments in L are disjoint?
- Point Line-Segment Cover with free points
  - **Input:** A set of points P in $\mathbb{R}^2$, a non-negative integer k and a non-negative integer p.
  - **Question:** Does there exist a set of line segments L, such that exactly $|P| - p$ points in P lie on at most one line segment in L, $|L|$ is at most k and all line segments in L are disjoint?
- Connected Point Line-Segment Cover (Path)
  - **Input:** A set of points P in $\mathbb{R}^2$, a non-negative integer k.
  - **Question:** Does there exist a set of line segments L, such that $|L|$ is at most k, all points in P lie on at most one line segment in L with the exception that, if $|L| \geq 2$, every line segment has at least one endpoint $p \in P$ in common with another line segment?

References

[1] Rodney G Downey and Michael Ralph Fellows. *Parameterized Complexity*. Springer, 2012.

[2] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[3] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

[4] Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1(5):194–197, 1982.

[5] Björn Brodén, Mikael Hammar, and Bengt J Nilsson. Guarding lines and 2-link polygons is apx-hard. In *The Thirteenth Canadian Conference on Computational Geometry-CCCG'01, University of Waterloo, ON (13-15 August 2001)*, pages 45–48, 2001.

[6] Jianxin Wang, Wenjun Li, and Jianer Chen. A parameterized algorithm for the hyperplane-cover problem. *Theoretical Computer Science*, 411(44-46):4005–4009, 2010.

[7] Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: The easy kernel is essentially tight. *ACM Transactions On Algorithms (TALG)*, 12(3):1–16, 2016.

[8] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry*. Springer, 2008.

[9] Sándor P Fekete, Kan Huang, Joseph SB Mitchell, Ojas Parekh, and Cynthia A Phillips. Geometric hitting set for segments of few orientations. *Theory of Computing Systems*, 62(2):268–303, 2018.

[10] Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[11] Scott Aaronson. *The Complexity Zoo*. `https://complexityzoo.net/Complexity_Zoo`, 2005.

[12] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.

Appendices

A All result tables

Table 6: Instances with only randomly generated points

Amount of points	k	Answer	Inst Cons.	Kernelization Time	Solving Time	NK Inst Cons.	NK Solving Time
1	1	true	2	00:00:00.0039501	00:00:00.0026792	2	00:00:00.0037121
2	1	true	1	00:00:00.0005511	00:00:00.0001764	4	00:00:00.0006234
3	1	false	1	00:00:00.0013354	00:00:00.0000015	19	00:00:00.0000399
	2	true	5	00:00:00.0003175	00:00:00.0002839	5	00:00:00.0002114
4	1	false	1	00:00:00.0003059	00:00:00.0000018	51	00:00:00.0001661
	2	true	24	00:00:00.0002912	00:00:00.0002326	24	00:00:00.0002223
5	1	false	1	00:00:00.0003137	00:00:00.0000015	124	00:00:00.0003984
	2	false	1	00:00:00.0002780	00:00:00.0000010	448	00:00:00.0014024
	3	true	25	00:00:00.0002955	00:00:00.0002864	25	00:00:00.0002801
6	1	false	1	00:00:00.0003631	00:00:00.0000015	238	00:00:00.0011307
	2	false	1	00:00:00.0002931	00:00:00.0000007	1336	00:00:00.0066684
	3	true	448	00:00:00.0002828	00:00:00.0018577	448	00:00:00.0015498
7	1	false	1	00:00:00.0004511	00:00:00.0000017	441	00:00:00.0019523
	2	false	1	00:00:00.0002792	00:00:00.0000008	3663	00:00:00.0154887
	3	false	16551	00:00:00.0003259	00:00:00.0698876	16551	00:00:00.0639901
	4	true	458	00:00:00.0005554	00:00:00.0023885	458	00:00:00.0020494
8	1	false	1	00:00:00.0006494	00:00:00.0000017	743	00:00:00.0044591
	2	false	1	00:00:00.0004289	00:00:00.0000007	8483	00:00:00.0433195
	3	false	55667	00:00:00.0003398	00:00:00.2723346	55667	00:00:00.2618208
	4	true	16500	00:00:00.0004137	00:00:00.0767540	16500	00:00:00.0720964
9	1	false	1	00:00:00.0007691	00:00:00.0000024	1194	00:00:00.0085376
	2	false	1	00:00:00.0004489	00:00:00.0000008	18114	00:00:00.1114409
	3	false	164010	00:00:00.0002962	00:00:00.9898453	164010	00:00:00.9454811
	4	false	893490	00:00:00.0003612	00:00:05.1269407	893490	00:00:04.9303408
	5	true	16712	00:00:00.0006300	00:00:00.0862939	16712	00:00:00.0856185
10	1	false	1	00:00:00.0009790	00:00:00.0000023	1858	00:00:00.0153109
	2	false	1	00:00:00.0004771	00:00:00.0000008	37846	00:00:00.2782040
	3	false	1	00:00:00.0003156	00:00:00.0000008	473470	00:00:03.1751207
	4	false	3609070	00:00:00.0002941	00:00:23.6425562	3609070	00:00:22.7146726
	5	true	2099526	00:00:00.0004733	00:00:13.0353195	2099526	00:00:12.4341398
11	1	false	1	00:00:00.0015330	00:00:00.0000026	2739	00:00:00.0279661
	2	false	1	00:00:00.0004836	00:00:00.0000008	71031	00:00:00.6442016
	3	false	1	00:00:00.0002925	00:00:00.0000007	1168887	00:00:09.3151399
	4	false	12077487	00:00:00.0003170	00:01:32.4629718	12077487	00:01:28.2357634
	5	false	77529087	00:00:00.0005704	00:09:26.1007534	77529087	00:09:03.1756636
	6	true	1041667	00:00:00.0006890	00:00:06.7658626	1041667	00:00:06.6883672
12	1	false	1	00:00:00.0020632	00:00:00.0000025	3741	00:00:00.0443673
	2	false	1	00:00:00.0002960	00:00:00.0000008	112797	00:00:01.1714270
	3	false	1	00:00:00.0002822	00:00:00.0000007	2216493	00:00:20.6964083
	4	false	28627533	00:00:00.0003087	00:04:15.1631630	28627533	00:04:08.1586494
	5	false	243245133	00:00:00.0006063	00:34:28.9053410	243245133	00:32:56.8814210
	6	true	66493665	00:00:00.0005639	00:08:47.8440194	66493665	00:08:20.5858481

Table 7: Variable random points, fixed length of one reserved line segment

Random Points	k	Answer	Instances Considered	Kernelization Time	Solving Time
1	1	false	1	00:00:03.0792969	00:00:00.0014401
	2	true	2	00:00:02.7430270	00:00:00.0026466
2	1	false	1	00:00:03.0840361	00:00:00.0000216
	2	true	10	00:00:02.7586321	00:00:00.0002956
3	1	false	1	00:00:03.1663195	00:00:00.0000224
	2	false	224	00:00:02.7553824	00:00:00.0013724
	3	true	16	00:00:02.6375758	00:00:00.0007380
4	1	false	1	00:00:03.3157846	00:00:00.0000229
	2	false	554	00:00:02.7356389	00:00:00.0052691
	3	true	330	00:00:02.6035075	00:00:00.0039089
5	1	false	1	00:00:03.4135314	00:00:00.0000245
	2	false	1	00:00:02.7291971	00:00:00.0000020
	3	false	16709	00:00:02.6180239	00:00:00.2322390
	4	true	493	00:00:02.5437175	00:00:00.0133458
6	1	false	1	00:00:03.5571792	00:00:00.0000243
	2	false	1	00:00:02.7343000	00:00:00.0000012
	3	false	91208	00:00:02.6283986	00:00:03.3220419
	4	false	1884738	00:00:02.5189689	00:01:10.2319390
	5	true	2295	00:00:02.3957075	00:00:00.1244239
7	1	false	1	00:00:03.7035864	00:00:00.0000254
	2	false	1	00:00:02.7649282	00:00:00.0000013
	3	false	68168	00:00:02.6563028	00:00:01.3592310
	4	false	1101604	00:00:02.5401152	00:00:21.4912854
	5	true	22989	00:00:02.4126948	00:00:00.4168763
8	1	false	1	00:00:03.8059171	00:00:00.0000264
	2	false	1	00:00:02.8101017	00:00:00.0000016
	3	false	400941	00:00:02.7274087	00:00:29.2645286
	4	false	11348595	00:00:02.5175227	00:13:03.3450598
	5	true	6553737	00:00:02.4461500	00:06:31.6762935
9	1	false	1	00:00:03.9473646	00:00:00.0000274
	2	false	1	00:00:02.7573878	00:00:00.0000019
	3	false	670288	00:00:02.6245817	00:00:56.2546237
	4	false	21636804	00:00:02.5294348	00:28:51.7106491
	5	TIMEOUT			

Table 8: Fixed random points, fixed length of variable amount of reserved line segments

RLS	k	Answer	Instances Considered	Kernelization Time	Solving Time
1	1	false	1	00:00:00.0079234	00:00:00.0016385
	2	false	183	00:00:00.0012320	00:00:00.0035731
	3	true	15	00:00:00.0016307	00:00:00.0006804
2	1	false	1	00:00:00.0229842	00:00:00.0022208
	2	false	1	00:00:00.0032824	00:00:00.0000136
	3	false	525	00:00:00.0023773	00:00:00.0068482
	4	true	27	00:00:00.0014971	00:00:00.0020048
3	1	false	1	00:00:00.0914007	00:00:00.0072018
	2	false	1	00:00:00.0149651	00:00:00.0003459
	3	false	1	00:00:00.0046629	00:00:00.0000146
	4	false	939	00:00:00.0032979	00:00:00.0206848
	5	true	38	00:00:00.0021205	00:00:00.0040271
4	1	false	1	00:00:00.2602144	00:00:00.0207169
	2	false	1	00:00:00.0363300	00:00:00.0006492
	3	false	1	00:00:00.0128750	00:00:00.0001659
	4	false	1	00:00:00.0060164	00:00:00.0000197
	5	false	1413	00:00:00.0038445	00:00:00.0402919
	6	true	49	00:00:00.0037548	00:00:00.0082706
5	1	false	1	00:00:00.6050578	00:00:00.0448781
	2	false	1	00:00:00.0905288	00:00:00.0015132
	3	false	1	00:00:00.0307616	00:00:00.0002700
	4	false	1	00:00:00.0126068	00:00:00.0001108
	5	false	1	00:00:00.0063124	00:00:00.0000267
	6	false	1971	00:00:00.0036278	00:00:00.0725955
	7	true	60	00:00:00.0025836	00:00:00.0128084
6	1	false	1	00:00:01.1571123	00:00:00.0907193
	2	false	1	00:00:00.1867593	00:00:00.0029772
	3	false	1	00:00:00.0576432	00:00:00.0003978
	4	false	1	00:00:00.0264638	00:00:00.0001582
	5	false	1	00:00:00.0123379	00:00:00.0000993
	6	false	1	00:00:00.0059010	00:00:00.0000332
	7	false	2613	00:00:00.0045035	00:00:00.1190375
	8	true	71	00:00:00.0039961	00:00:00.0201920
7	1	false	1	00:00:02.0722400	00:00:00.1534155
	2	false	1	00:00:00.3306776	00:00:00.0055732
	3	false	1	00:00:00.1105116	00:00:00.0007673
	4	false	1	00:00:00.0491124	00:00:00.0002306
	5	false	1	00:00:00.0240337	00:00:00.0001311
	6	false	1	00:00:00.0112902	00:00:00.0000902
	7	false	1	00:00:00.0061791	00:00:00.0000428
	8	false	3339	00:00:00.0034499	00:00:00.1739312
	9	true	82	00:00:00.0032018	00:00:00.0250007
8	1	false	1	00:00:03.5307030	00:00:00.2602055
	2	false	1	00:00:00.5423969	00:00:00.0087571
	3	false	1	00:00:00.1885382	00:00:00.0013511
	4	false	1	00:00:00.0833153	00:00:00.0003018
	5	false	1	00:00:00.0415730	00:00:00.0001700
	6	false	1	00:00:00.0211080	00:00:00.0001110
	7	false	1	00:00:00.0099273	00:00:00.0000960
	8	false	1	00:00:00.0050074	00:00:00.0000535
	9	false	4149	00:00:00.0060971	00:00:00.2708256
	10	true	93	00:00:00.0041590	00:00:00.0350020

Table 9: Fixed random points, variable length of a single reserved line segment

Points on RSL	k	Answer	Inst. Cons.	Kernelization Time	Solving Time	NK Inst. Cons.	NK Solving Time
20	1	false	1	00:00:00.0326929	00:00:00.0014378	35297	00:00:01.4171314
	2	false	1	00:00:00.0135501	00:00:00.0000046	2978561	00:01:48.0381747
	3	true	198	00:00:00.0116515	00:00:00.0055761	2448915	00:01:18.2698268
40	1	false	1	00:00:00.4804957	00:00:00.0000123	369797	00:01:05.6552182
	2	false	1	00:00:00.4101423	00:00:00.0000181	96637979	04:04:23.3483077
	3	true	255	00:00:00.3613198	00:00:00.0061483		
60	1	false	1	00:00:03.2531131	00:00:00.0000222		
	2	false	1	00:00:02.7174209	00:00:00.0000169		
	3	true	255	00:00:02.6304784	00:00:00.0052304		
80	1	false	1	00:00:12.7366406	00:00:00.0000397		
	2	false	1	00:00:11.4571063	00:00:00.0000180		
	3	true	255	00:00:11.1007506	00:00:00.0061422		
100	1	false	1	00:00:37.6238518	00:00:00.0000553		
	2	false	1	00:00:34.1135616	00:00:00.0000358		
	3	true	255	00:00:33.4502654	00:00:00.0058553		

Table 10: Instances with added length constraint

Instance	k	l	Answer	Inst. Cons.	Kernelization Time	Solving Time
6 points	3	0	false	3928	00:00:00.0040800	00:00:00.0189259
	3	float.MaxValue	true	448	00:00:00.0004098	00:00:00.0019966
	3	738	true	1576	00:00:00.0004391	00:00:00.0058513
3 points, 1 RSL of 60 points	3	0	false	4614	00:00:10.9776450	00:00:00.0638206
	3	float.MaxValue	true	255	00:00:10.8589252	00:00:00.0051503
3 points, 6 RSL of 10 points	8	0	false	19456	00:00:00.0092781	00:00:00.7590744
	8	float.MaxValue	true	71	00:00:00.0027998	00:00:00.0203413
6 points, 1 RSL of 40 points	5	0	false	35000712	00:00:02.3212540	00:21:52.9389384
	5	float.MaxValue	true	2295	00:00:02.3130146	00:00:00.1220677
	5	1058	true	84893	00:00:02.3450554	00:00:02.8894118