MASTER

Point-feature labelling in the 1-Slider model
a fixed-parameter tractable algorithm

de Visser, L.L.

Award date:
2015

Technische Universiteit Eindhoven University of Technology

Graduation Project: Computer Science & Engineering master

Point-Feature Labelling in the 1-Slider Model: A Fixed-Parameter Tractable Algorithm

Author:
Luuk L. de Visser

Supervisor:
Dr. Bart M.P. Jansen

Members of Graduation Comittee:
Dr. Herman J. Haverkort

Dr. Kevin A.B. Verbeek

August 31, 2015

Abstract

The placing of textual labels is a central task to many visualisation applications. The basic variant of this problem, known as the point-feature labelling problem, has been studied extensively in literature. However, as the problem is NP-hard for most variations, the vast majority of known algorithms give an approximate solution. As such, in this paper we present a fixed-parameter tractable algorithm for the point-feature labelling problem in the 1-slider model, returning a provably optimal solution. As parameter k, we use the maximum feedback edge number among all connected components of the graph underlying the problem. Our algorithm has an asymptotic running time of $\mathcal{O}(2^k \cdot n^2)$, where n is the size of the point set used as input. We implemented our algorithm, and present experimental results on real-world examples of data sets. We also tested two techniques that might cause a speed-up in practice, and analyse their effects.

Contents

1 Introduction

The placing of textual labels with their corresponding graphical objects is a central task to many visualisation applications. These include, but are not limited to, engineering drawings, charts such as scatterplots, and, perhaps most significantly, representations of geographical data. The last of these goes back far in the history of human kind, in map making, and has since then been extensively described in the literature. What constitutes the labelling problem is our desire to place labels in such a way, that it is easy for a user of the graphic to determine which label corresponds to which object. This has always been done, and still is, by craftsmen who through years of experience developed an instinctual set of criteria on how to best place the labels. However, as this process is very time intensive, we could greatly benefit from finding automated processes to complete this task. Research into automated label placement was kickstarted in the second half of the last century by papers from Imhof and Yoeli. The former presented a well-defined set of criteria for what constitutes a good labelling, whereas before rules or guidelines were never written down, barring some rare exceptions [15]. The latter wrote one of the first papers on automated label placement, and the basic principles of this problem [25].

1.1 Our Contribution

While the general labelling problem labels any graphical object with another graphical object, we will focus on the point-feature label problem (PFL problem). In the PFL problem, the objects we aim to label are points, often represented by small disks, squares or symbols. The labels (or features) frequently are thought of as rectangles, however this is not exclusively the case. The basic variant of the PFL problem, where the labels are textual and represented by the axis-aligned bounding box of the text, is often divided into two types of labelling models: the fixed-position models and the slider models. In the fixed-position models, we allow for a finite set of fixed positions for the label to be placed around its point. On the contrary, in the slider models the number of possible placements is infinite. This generally means that the label can be placed anywhere, as long as it touches its point with its border according to some non-discrete criteria. Some of the most common optimisation criteria include finding the largest subset of labels we can place such that they are pairwise non-overlapping, or what the largest scaling factor is such that all labels are pairwise non-overlapping. Unfortunately, these optimisation problems have been shown to be NP-hard for most settings [5].

As we will see in the Section 1.2, previous work on the PFL problem only aims to find an approximate solution to the optimisation problems, because of the NP-hardness. Our aim therefore is to construct an algorithm that finds an optimal solution to the PFL problem in the 1-slider model. In the 1-slider model, every label placed must touch its point along its bottom edge. The aim is then to find the smallest subset of labels to be omitted from the drawing, such that all other labels can be placed without pairwise overlaps. In order to do so, we will have to make use of the fixed-parameter tractability framework, as we will show that this problem is NP-complete.

Fixed-parameter tractability, or FPT, is a somewhat recent development in complexity theory that allows us to more finely categorise NP-hard problems. For a parameterised problem, every problem instance x is associated with an integer k, called the parameter, that in some way measures the complexity of the instance. Frequently used parameter choices are the size of the solution, or a complexity measure of the graph associated with the problem instance. A parameterised decision problem is then fixed-parameter tractable if there is an algorithm that, given an input (x, k) of encoding size n, gives the yes/no answer to the problem in time $f(k) \cdot n^{\mathcal{O}(1)}$. Here, $f(k)$ is any function depending exclusively on k. Similarly, a parameterised optimisation problem is fixed-parameter tractable if there is an algorithm that, given an input

instance (x, k) of size n, computes an optimal solution in time $f(k) \cdot n^{\mathcal{O}(1)}$. If k is then 'small' relative to n, the algorithm's run time is a significant improvement over the $\mathcal{O}(n^{k+\mathcal{O}(1)})$ runtime of a naive algorithm.

We will present a fixed-parameter tractable algorithm that takes a measure of the tree-likeness of the graph underlying the PFL problem as parameter. We studied a concept for this algorithm and refined it, after which we formalised the algorithm and proved its correctness. Furthermore, we implemented our algorithm, and used this to experiment on real-world datasets. Our aim for these experiments was to discover what the parameter values of real-world datasets are like, what the relation between the parameter and the runtime of the algorithm is in practice, and how the parameter behaves during the execution of the algorithm. Lastly, we tested the effect of two techniques that could potentially speed up the algorithm.

1.2 Related Work

1.2.1 Point-Feature Labelling

As we have mentioned earlier, for the basic PFL problem we generally distinguish between the fixed-position models and the slider models. In the fixed-position model, Agarwal et al. presented a 1/2-approximation algorithm for rectangular labels of unit height, and a $\mathcal{O}(1/\log n)$-approximation for varying label heights, for finding the maximum subset of labels to place pairwise non-overlapping. That is to say, their algorithms successfully placed at least the approximation-factor times the number of labels that would be placed in the optimal solution. Both approximations can be found in $\mathcal{O}(n \log n)$ time [1]. In a more practical vein, Alvim and Taillard used the POPMUSIC framework, based on an adapted form of tabu search (first presented by Glover [11]), as a heuristic for finding the maximum subset, making use of the combinatorial nature of the problem [2]. Furthermore, a model using eight fixed positions was used by Martnez-Ovando et al. to label maps of metro systems [20].

Kreveld et al. presented approximation algorithms for finding the maximum subset of labels for the 1-, 2- and 4-slider models [23]. In these models, the axis-parallel rectangular label must touch its point with its bottom edge, top or bottom edge, or any of its edges, respectively. For all three models they give a 1/2-approximation in $\mathcal{O}(n \log n)$ time, as well as a polynomial time approximation scheme. Again, their algorithms thus successfully placed at least the approximation-factor times the number of labels placed in the optimal solution. Van Kreveld and Strijk also present practical extensions in a follow-up paper [21].

Doddi et al. published a paper on 'Map Labeling and its Generalisations', presenting amongst other things a polynomial time approximation algorithm for a rectangular labelling model where the labels can have any orientation, as long as they touch their point with their boundary [8]. A concept highly relevant today in digital mapping, Been et al. explore the problem of dynamic map labelling, where the map can be zoomed and translated in real-time, and speed is of higher importance than in the static case [3]. They present a formal model of dynamic map labelling, and dynamic labels. Lastly, Do Nascimento and Eades claim that none of the algorithms in literature fulfill all criteria for good labellings, and as such present an interactive framework where automation and artisan meet. Their 'User Hints' framework supports the cartographer, rather than attempting to replace them [7].

1.2.2 Fixed-Parameter Tractability

The framework of parameterised complexity was first presented by Downey and Fellows [9]. Since then, fixed-parameter tractability has been applied to various, often well-known, problems.

Often the parameter is the size of the solution, such as in vertex cover, one of the most well-explored fixed-parameter tractable problems. Currently the fastest FPT-algorithm for this problem requires $\mathcal{O}(1.2738^k + kn)$ time, as presented by Chen et al. [4]. Finding an undirected feedback vertex set of size at most k has also been shown to be FPT, with k being the parameter. The most recent improvement was given by Kociumaka and Pilipczuk, who gave a $\mathcal{O}((2+\phi)^k \cdot n^{\mathcal{O}(1)})$ time algorithm, where ϕ is the golden ratio [16]. Almost 2-SAT, where the aim is to find a subset of at most k clauses such that a 2-CNF is satisfiable when we remove this subset, was recently shown to allow for a $\mathcal{O}(2.3146^k \cdot n^{\mathcal{O}(1)})$ time algorithm, see [17] by Lokshtanov et al.

However, it is not always the case that the parameter is the size of the solution, as is also the case for our algorithm. For example, deciding whether given a graph G and a graph H, H is a topological subgraph of G, permits an FPT algorithm with the number of vertices of H as parameter, see Grohe et al. [13]. A frequently used parameter is the treewidth of a graph, as defined by Robertson and Seymour [19], even though finding the treewidth of a graph is itself NP-hard. For instance, Gottlob and Tien Lee presented FPT algorithms with treewidth of the input structure as the only parameter for several versions of the multicut problem [12], making use of Courcelle's theorem. This theorem states that, if a graph property of interest can be expressed in monadic second-order logic, then it can be decided in linear FPT time with the treewidth of the graph as parameter, whether or not that property holds for a graph [6].

2 Preliminaries

Before we go more in-depth on our contribution in this paper, we should first present some theoretical concepts and the notation we will use throughout this paper. First, it may be useful to formally state the problem we are aiming to tackle.

Standard Point-Feature Labelling Problem 1-Slider

Input: A set $I = (p_1, l_1), \ldots, (p_n, l_n)$ of n point-label pairs, and an integer k. Each point is given by an x- and y-coordinate, and each label given as an open axis-aligned rectangle.

Question: Is there a set $S \subseteq I$ of size k such that $I - S$ has a *non-overlapping drawing* in the 1-slider model?

A drawing of $I - S$ is non-overlapping if the drawn labels are pairwise non-overlapping. Throughout the remainder of this paper, we speak of this problem when we mention the PFL problem. Furthermore, we will denote the x-coordinate and y-coordinate of a point p as $p.x$ and $p.y$, respectively. Additionally, $width(l)$ will denote the width of label l, and $height(l)$ the height of label l.

Next, let us repeat what composes a fixed-parameter tractable problem:

Definition 2.1. *For a parameterised problem, every problem instance x is associated with an integer k, called the parameter, that in some way measures the complexity of the instance. A parameterised decision problem is then fixed-parameter tractable if there is an algorithm that, given an input (x, k) of encoding size n, gives the yes/no answer to the problem in time $f(k) \cdot n^{\mathcal{O}(1)}$. Here, $f(k)$ is any function depending exclusively on k. Similarly, a parameterised optimisation problem is fixed-parameter tractable if there is an algorithm that, given an input instance (x, k) of size n, computes an optimal solution in time $f(k) \cdot n^{\mathcal{O}(1)}$*

As we will later abstract from the geometric PFL problem, and focus on the underlying graph problem, we will make use of several graph-theoretical concepts and definitions:

- When we speak of a DAG, we mean a Directed Acyclic Graph. A DAG D consists of a set of vertices $V(D)$, and a set of directed edges, or arcs, $A(D)$.
- It is generally known that a DAG always has at least one source (a vertex without incoming arcs), and at least one sink (a vertex without outgoing arcs). We will make use of this property in our branch step.
- When we speak of the feedback edge number of a DAG in this paper, we mean the feedback edge number of the undirected graph underlying the DAG. The same is true for connected components of a DAG, and undirected cycles of a DAG.
- The feedback edge number of an undirected graph G, is the size of a minimum feedback edge set. A minimum feedback edge set is a minimum size set X of edges of G such that $G - X$ is acyclic. The feedback edge number can be given by the following formula: $|E(G)| - (|V(G)| - |C(G)|)$, where $E(G)$ is the edge set of G, $V(G)$ is the vertex set of G, and $C(G)$ is the set of maximally connected components of G.
- If we talk about a leaf l in a DAG D, we mean that $l \in V(D)$ and that its in-degree plus out-degree equals one. The parent p of l is then the only vertex it has an arc to or from.

Now that we have declared the definitions, concepts and notations we will make use of, we will start by proving in the next section that the PFL problem is NP-complete.

3 NP-Completeness of the Standard Point-Feature Labelling Problem in the 1-Slider Model

Before we present our algorithm, we think it is useful to formally prove why it is necessary to use a fixed-parameter tractable approach to the point label placement problem in the 1-slider model. As such, in this section we aim to validate the following theorem:

Theorem 3.1. *Determining whether there is a set of k labels such that, when we omit these labels, all other labels can be drawn pairwise non-overlapping in the 1-slider model, is NP-complete.*

In order to show that a problem is NP-complete, we have to show that it is both in NP, and NP-hard. It is trivial to show that the problem is in NP: Given a solution, we can place all labels not in the solution in a greedy manner in $\mathcal{O}(n^2)$ time (see subsection 4.1). We can then pairwise check for all these labels that their intersections are empty, using a brute force approach, again in $\mathcal{O}(n^2)$ time. What thus remains to be shown is that the point label placement problem in the 1-slider model is NP-hard. In order to prove this, we would like to do a reduction from the planar vertex cover problem. This problem is known to be NP-complete, even for graphs of maximum degree 3 (see Garey and Johnson [10]).

As such, consider a planar graph with maximum degree 3 embedded in the grid. That is to say, each vertex lies on a grid point, and each edge is rectilinear and follows the grid lines. Such an embedding can be obtained in polynomial time, as described by Valiant [22]. We aim to translate such an embedding to an instance of the point label placement problem in the 1-slider model, where a specific 'chain' of labels represents an edge, and some gadget of labels represents a vertex. The label chains representing an edge have to push pressure along the edge, to mimic the requirement of the planar vertex cover problem that every edge should have at least one of its vertices in the solution. However, we cannot push pressure vertically, but we can push pressure in all diagonal directions. As such, we rotate the grid embedding by 45°. Furthermore, to make placing the labels easier later on, we also stretch the rotated grid embedding by a factor 2 along the horizontal axis, see Figure 1.

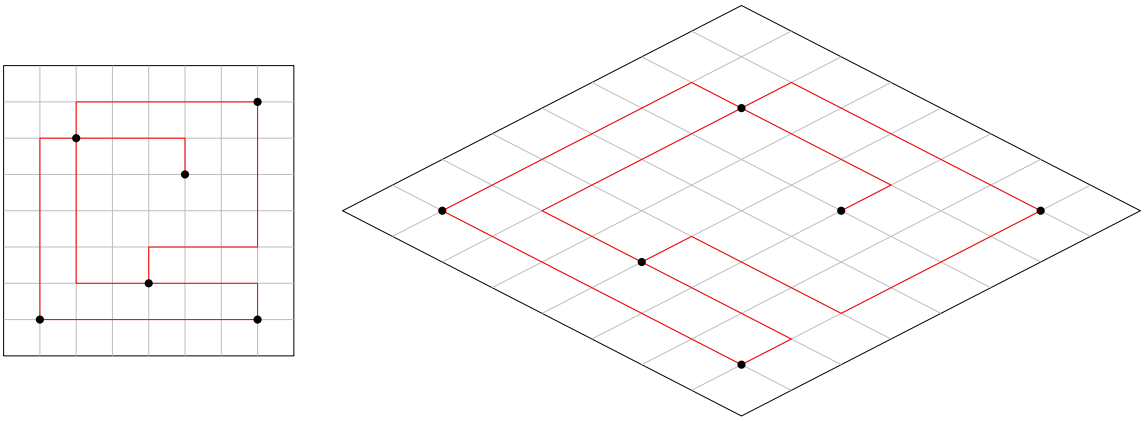


Figure 1: Left: the grid embedding. Right: the grid embedding after rotating and stretching.

From here on, when we mention the embedding, we mean the grid embedding of the planar graph of maximum degree 3 after rotation and stretching. In order to model the bends in the edges of the embedding, we will have to introduce new vertices, as shown in Figure 2. Note that we always place the new vertices on grid points. However this affects the size of the optimal solution, leading us to the following proposition:

Proposition 3.1. *Let $\{v_1, v_2\}$ be an edge in a graph G. Then let G' be graph G when we introduce two new vertices w_1, w_2 and three new edges $\{v_1, w_1\}, \{w_1, w_2\}, \{w_2, v_2\}$, and remove edge $\{v_1, v_2\}$. Then a vertex cover of size at most k exists for G if and only if a vertex cover of size at most $k+1$ exists for G'.*

Proof.

- $\Rightarrow$ Assume S is a vertex cover of size k for G. Then one of v_1, v_2 is in S, otherwise edge $\{v_1, v_2\}$ is not covered. Let us assume without loss of generality, by symmetry, that $v_1 \in S$. Then $S \cup \{w_2\}$ is a vertex cover of size $k+1$ for G', as $v_1 \in S$ covers edge $\{v_1, w_1\}$, w_2 covers edges $\{w_1, w_2\}$ and $\{w_2, v_2\}$, and all other edges are the same as in G.

- $\Leftarrow$ Assume S' is a solution of size $k+1$ for G'. Then one of v_1, w_1, one of w_1, w_2 and one of w_2, v_2 are in S'. Let S then be as follows: if S' contains one of v_1, v_2, $S := S' \setminus \{w_1, w_2\}$, else $S := (S' \setminus \{w_1, w_2\}) \cup \{v_1\}$. We claim that S is a vertex cover for G. Assume, for contradiction, that this is not the case and that $\{u_1, u_2\}$ is an uncovered edge in G. If $\{u_1, u_2\} \neq \{v_1, v_2\}$, then this edge is also in G', thus S' would not be a vertex cover to G', resulting in a contradiction. Furthermore, if $\{u_1, u_2\} = \{v_1, v_2\}$, then by construction of S this edge is covered by S in G, and we again arrive at a contradiction. Indeed S is a vertex cover of G. One of w_1, w_2 must be in S' if it contains v_1 or v_2, and both of w_1, w_2 must be in S' if it does not contain v_1 or v_2, otherwise not all edges are covered by S' in G'. Per construction of S, we thus know that $|S| = k$.

□

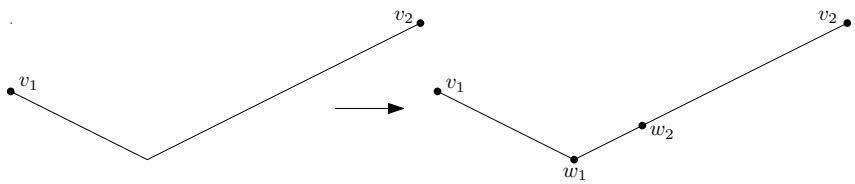
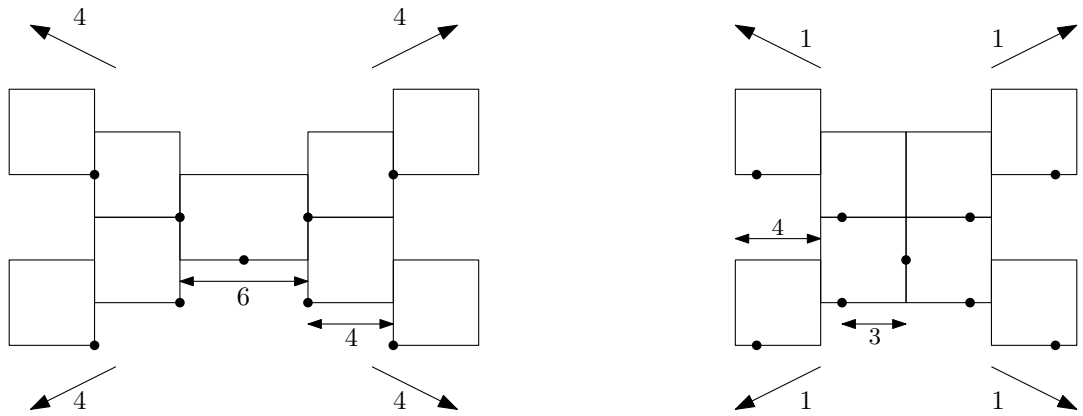


Figure 2: Adding two new points eliminates the mid-edge bend.

It has been shown that planar graphs of maximum degree 3 have grid embeddings of maximum area $O(|V|^2)$ [22]. Since the edges follow the grid lines, this means there is at most a polynomial number of bends, and thus a polynomial number of new points introduced. This is important to ensure we have a polynomial time reduction.

Now that we have finalised our embedding, we place it in a coordinate system such that every grid point has coordinates of the form $(40a, 20b)$. This is to ensure we can precisely specify where we place points of the label placement problem using integer coordinates. We then want to model the vertices of the embedding, including the ones introduced by bends in edges. To do so, we introduce a gadget, such that we create maximum pressure in all four possible directions if we place all labels in the gadget (see Figure 3a), but generate only slight pressure if we do not place its middle label (see Figure 3b).

Now we want to model the edges of the embedding using labels. We do this as shown in Figure 4, so that we can place all labels of the edge gadget if there is only 1 pressure from one or both sides, and all but one if there is 4 pressure from both sides. On an intuitive level, the aim here is that the edge is not covered in the planar vertex cover problem if there is 4 pressure from both sides in the label variant. Figure 4, and its mirror image, together show all possible edge gadget and pressure combinations.



(a) The gadget generates the full 4 pressure to all directions when all its labels are placed.

(b) The vertex gadget generates only 1 pressure when its middle label is not placed.

Figure 3: Two states of the vertex gadget.

As we have now presented our vertex and edge gadgets, what remains is to show where precisely we place them. As mentioned earlier, we place our grid in a coordinate system such that every grid point has coordinates of the form $(40a, 20b)$, where a and b are integers. For every vertex of the embedding with coordinates (x, y), we then place the following points for the label problem to form the vertex gadget. All labels have width 4 and height 4, unless specified otherwise.

- (x, y), this point has a label of width 6.
- $(x-3, y+2), (x-3, y-2), (x+3, y+2), (x+3, y-2)$

For every edge of the embedding between vertices (x, y) and (x', y'), where we assume $x < x'$, we place these points to form the edge gadget, where again all labels have width and height 4:

for every integer i satisfying $1 \leq i \leq 3 + 5(\frac{x'-x}{40} - 1)$:

- if $y < y'$: $(x+3+4i, y+2+2i), (x'-3-4i, y'-2-2i)$, and $(\frac{x+x'}{2}, \frac{y+y'}{2})$
- if $y' > y$: $(x+3+4i, y-2-2i), (x'-3-4i, y'+2+2i)$, and $(\frac{x+x'}{2}, \frac{y+y'}{2})$

Now that we have defined how to transform the embedding to a point label placement problem, we want to show that we can also transform a solution of the planar vertex cover problem on graphs with maximum degree 3 to a solution of the label placement problem in the 1-slider model:

Lemma 3.1. *Let P be an instance of the planar vertex cover with maximum degree 3, after embedding, rotation, stretching and resolving bends, as described. Furthermore, let L be the label placement problem obtained from P by the aforementioned process. Then P has a solution of size at most k if and only if L has a solution of size at most k.*

Proof.

$\Rightarrow$ Assume S is a solution to P of size k. We put the middle label of the vertex gadget in the solution S' for L, for every vertex in S. Then, since S is a solution to the vertex cover problem, every edge gadget of L has full pressure from at most one of its vertex gadgets. As shown in Figure 4), this means we can place all other labels without overlap,

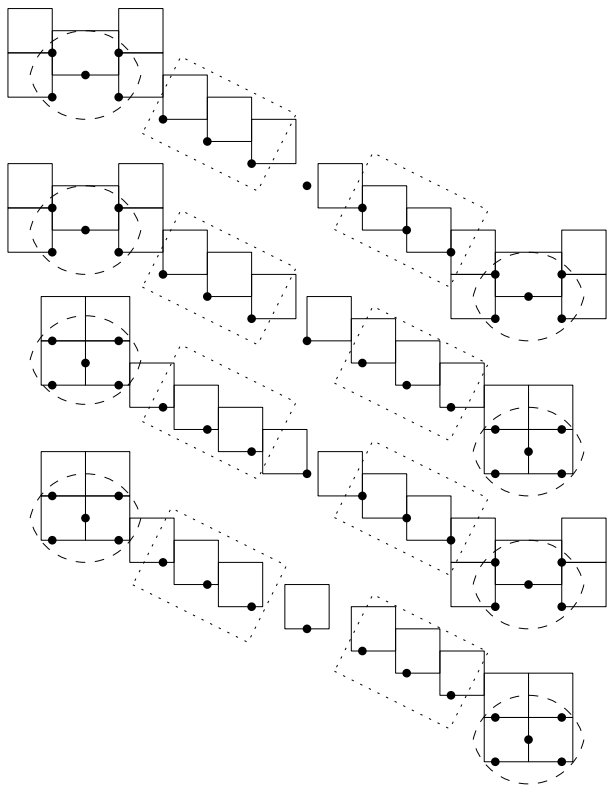


Figure 4: The edge gadget can place all labels with only 1 pressure from at least one side, and all but one label with 4 pressure from both sides. Points in the dashed regions are part of the vertex gadgets. Points in the dotted regions are non-central points of the edge gadget; for every grid point between the two vertices of the edge, each of the point chains in the dotted regions would be 5 points longer.

using the placements presented in that figure. In this figure, there is a chain of three points (dotted regions) between the center point of the edge gadget, and each vertex gadget. If the vertices would have more grid points in between, then as reflected in the coordinates presented earlier, these two chains would simply be extended by five points each per such grid point. The placement of their labels would remain similar to that of Figure 4, however. As such, S' is a solution of size k to L.

$\Leftarrow$ Assume S' is a solution to L of size k. Since S' is a solution to L, we know that for every edge gadget and its two vertex gadgets, at least one of those labels is in S'. For if this were not the case, then all the labels of the vertex gadgets must be placed, which can only be done in one way (see Figure 3a. Assume we have an edge $\{v, u\}$, and assume without loss of generality that $v.x > u.x$ (after rotating there are no strictly vertical edges). The edge gadget of that edge then contains 2 labels for every integer i satisfying $1 \leq i \leq 3+5((v.x-u.x)/40-1)$, per definition of the point coordinates. Additionally, the edge gadget contains a single middle point, for a total of $2 \cdot (3+5((v.x-u.x)/40-1)+1$ labels of width 4. As such, we need $28+10((v.x-u.x)/40-1)$ horizontal space to place all the labels of the edge gadget, since each label of the edge gadget can potentially overlap with its 'neighbouring' labels. Yet, there is $34+40((v.x-u.x)/40-1)$ horizontal space between the corners of the vertex gadgets along the edge. In this case, those are the points with coordinates $(v.x-3, v.y \pm 2)$ and $(u.x+3, u.y \mp 2)$ for $v.y > u.y$, respectively $u.y > v.y$. Thus if we take into account the 4 pressure from each vertex gadget (placed

as in Figure 3a), we are left with $26 + 40((v.x - u.x)/40 - 1)$ space, while we would need $28 + 40((v.x - u.x)/40 - 1)$ to place all labels. This contradicts with our assumption that S' is a solution to L. Thus S' contains at least one label per edge gadget and its two vertex gadgets. Then for each label $l \in S'$, if l is a label of the vertex gadget for vertex v, we put v in S. Otherwise, if l is a label of an edge gadget for edge $\{v, u\}$, add an arbitrary endpoint of that edge to S. As such, we know S contains one endpoint of every edge in P, and is thus a vertex cover of P. Note that $|S| \leq |S'|$, and we conclude P has a solution of size at most k.

□

Combining the ingredients presented in this section into one algorithm, we can take a planar graph G of maximum degree 3 and an integer k as input, and build an instance of the label placement problem L, such that L has a solution of size at most $k + s$ if and only if G has a vertex cover of size at most k. Here s is the number of times we have to subdivide an edge to eliminate a bend. As we mentioned earlier, we can obtain an embedding with area $\mathcal{O}(|V(G)|^2)$ in polynomial time, as shown by Valiant [22]. The area of this embedding also implies the number of bends we have to resolve is polynomial, thus s is of polynomial size. Lastly, the instance of the label placement problem can be constructed efficiently from the embedded, rotated, stretched graph after resolving edge-bends. These results combined give rise to the following corollary:

Corollary 3.1. *The point label placement in the 1-slider model is NP-hard.*

This, combined with our earlier observation that the point label placement in the 1-slider model is in NP, validates Theorem 3.1, and we conclude that this problem is NP-complete.

4 The Label Placement Problem and its Graph Representation

As we have mentioned previously, in the label placement problem we want to find a minimum subset of labels such that we can place all other labels without overlap. The algorithm we will present later will find such a subset using a graph representation of the original problem. Yet once we have such a subset, how precisely do we place the rest of the labels? To this end, we first present a greedy label placing algorithm. We will then, using the approach of this greedy algorithm, determine when a set of labels can be drawn without overlap. Lastly, we will use this information to form a graph representation of the point and label set, which shall later be used as the input to our algorithm.

4.1 Greedy Label Placement

Recall that we view the labels of points as open rectangles with a fixed height and some given width, and that two labels overlap if their intersection is non-empty. A *non-overlapping drawing* is then a drawing where every label of the input is drawn without any two different labels overlapping, while adhering to the 1-slider model.

Given an input (a set of points, the coordinates of those points, and their corresponding label), the greedy algorithm creates a drawing as follows: Iterating over the points from smallest to greatest x-coordinate, draw the label as far left as possible in the 1-slider model, preventing new overlaps if possible. If two points have the same x-coordinate, the point with the greater y-coordinate is iterated over first.

More specifically, let l_n be the label we want to draw, and p_n its corresponding point. If we can draw l_n with its left edge at $x = p_n.x - width(l_n)$ without overlapping any previously drawn labels, then we do so. Otherwise let l_r be the right-most label we would overlap with, and we say that l_r *influences* the placement of l_n. Let x_r be the x-coordinate of the right edge of l_r, then we draw l_n with its left edge at $min(x_r, p_n.x)$. We will refer to drawing the label of a point in this manner as the greedy choice, and the entire drawing as the greedy drawing.

Now that we have described our greedy label drawing method, evidently we want to prove the validity of this method. To this end, we present and prove the following lemma:

Lemma 4.1. *Let I be an input to be drawn. Then the greedy drawing of I contains no overlapping labels if and only if there exists a drawing of I containing no overlapping labels.*

Proof. Clearly if the greedy drawing of I contains no overlapping labels, there exists a non-overlapping drawing of I. We prove the other direction by contradiction: Consider any input I^* that allows at least one drawing containing no overlapping labels, but for which the greedy drawing has at least one overlap. Then consider a non-overlapping drawing of I^* that draws as many labels as possible according to the greedy choice. Let l_d then be the label with the smallest x-coordinate that was not drawn by greedy choice. Then l_d is not drawn as far to the left as possible in the 1-slider model, without overlapping other labels, for that would have been the greedy choice. Note however that instead drawing it using the greedy choice does not give rise to new overlaps. Thus we can safely draw it using the greedy choice, and we obtain a drawing that draws strictly more labels according to the greedy choice. This is a contradiction to our assumption that we started with a non-overlapping drawing that draws as many labels as possible according to the greedy choice. □

Thus, our greedy label drawing method has been verified, and will always succesfully find a non-overlapping drawing if one exists. In order to formulate a graph representation of a

point and label input however, we would like to know precisely when the input has a non-overlapping drawing. As such, consider a sequence P of points $\langle p_1, \ldots, p_e \rangle$, with accompanying labels $l_1, \ldots, l_e$, such that for each pair p_i, p_{i+1} we have:

- $(p_i.x < p_{i+1}.x) \vee (p_i.x = p_{i+1}.x \wedge p_i.y < p_{i+1}.y)$.
- if we draw the labels of only the points of P in order using the greedy algorithm, then l_i influences the placement of l_{i+1}.

Then we claim the following:

Lemma 4.2. *P has a non-overlapping greedy drawing if and only if* $\sum_{i=2}^{e-1} width(l_i) \leq p_e.x - p_1.x$.

Proof.

$\Rightarrow$ Assume P has a non-overlapping greedy drawing. In the worst case, the non-overlapping drawing of P places l_1 with its right edge at $p_1.x$, and l_e with its left edge at $p_e.x$, leaving the horizontal space in between to place all other labels. Since every label l_i influences the placement of label l_{i+1}, l_i can never share a non-empty interval on the x-axis with l_{i+1}, thus placing all labels $l_2, \ldots, l_{e-1}$ takes $\sum_{i=2}^{e-1} width(l_i)$ horizontal space. Therefore if we assume $\sum_{i=2}^{e-1} width(l_i) \leq p_e.x - p_1.x$ does not hold, the labels we place require more horizontal space than is available. This results in an overlap, giving us a contradiction.

$\Leftarrow$ Assume $\sum_{i=2}^{e-1} width(l_i) \leq p_e.x - p_1.x$ holds for a sequence P. We draw l_1 with its right edge at $p_1.x$ and its bottom edge at $p_1.y$. Every subsequent label l_i we draw with its left edge touching the right edge of label l_{i-1}, and its bottom edge at $p_i.y$. This can be done without overlap, as per our assumption that $\sum_{i=2}^{e-1} width(l_i) \leq p_e.x - p_1.x$ holds. Then for some label l_i (with $i > 1$) we know that its left edge lies at $p_1.x + \sum_{j=2}^{i-1} width(l_i)$, and that its right edge lies at $p_1.x + \sum_{j=2}^{i} width(l_i)$. We also know that for the label of point p_{i-1} to influence the label of point p_i, it must be the case that $p_i.x < width(l_i) +$ x-coordinate of the right edge of l_{i-1}. This, combined with the fact that $p_{i-1}.x \leq p_i.x$, gives us that for every point p_i (with $i > 1$) we have that $p_1.x + \sum_{j=2}^{i-1} width(l_j) \leq p_i.x < p_1.x + \sum_{j=2}^{i} width(l_j)$. As such, we know that in our drawing, every label has its corresponding point along its bottom edge. Furthermore, our drawing is in fact the greedy drawing. We can conclude that the greedy drawing of P is non-overlapping.

□

4.2 Graph Representation

Since we have now established that Lemma 4.2 holds, we can use this information to formulate a graph representation of the input. Let our input be a set of points and their accompanying labels (p, l), that is $I = \{(p_1, l_1), \ldots, (p_n, l_n)\}$. Recall that a point p has an x-coordinate $p.x$, and a y-coordinate $p.y$. Then we define our graph representation D as follows:

- For every $(p_i, l_i) \in I$, add v_i to $V(D)$, the set of vertices of D.
- Let $(p_i, l_i), (p_j, l_j) \in I$, such that $p_i.x < p_j.x \ \vee \ (p_i.x = p_j.x \ \wedge \ p_i.y < p_j.y)$. If placing l_i with its left edge at $p_i.x$ and placing l_j with its right edge at p_j would cause them to overlap, add arc (v_i, v_j) to $A(D)$, the arc set of D.

- We create a weight function W_D. In this weight function we store two things: for each vertex $v_i \in V(D)$, we store $w_D(v_i) :=$ width of label l_i; for each arc $(v_i, v_j) \in A(D)$, we store $arcw_D(v_i, v_j) := p_j.x - p_i.x$.
- Lastly, we have a set of *potentials* Π_D. For every vertex $v_i \in V(D)$, Π_D stores the start-potential $\pi_D^-(v_i)$ and end-potential $\pi_D^+(v_i)$. Initially, all potentials are set to be 0.

The idea is to move away from the geometry of the original problem, while capturing the way the greedy approach would place labels. As such, a path in D represents a sequence of points whose labels could potentially influence eachother, and in W_D we store the information we need as shown in Lemma 4.2. Note that because of how we defined the arcs, D is in fact a directed acyclic graph, or DAG, which will be of significance later. The potentials which we store in Π are of no significance in the initial instance of (D, W_D, Π_D), however when we present our algorithm we will see how and when their value will change.

We have now defined how we represent our input as a DAG, but we have not yet discussed what a solution to the label placement problem looks like in the graph representation. The following inequality, applicable to any path $\langle v_1, \ldots, v_e \rangle$ in (D, W_D, Π_D), will be at the core of many things to come:

$$\left(\sum_{i=1}^{e} w_D(v_i)\right) + \pi_D^-(v_1) + \pi_D^+(v_e) \leq \left(\sum_{i=1}^{e-1} arcw_D(v_i, v_{i+1})\right) + w_D(v_1) + w_D(v_e) \tag{I}$$

Note that in the initial DAG, where all potentials are still 0, this inequality says the sum of vertex weights of internal vertices of a path $\langle v_1, \ldots, v_e \rangle$ in (D, W_D, Π_D) must be smaller than or equal to the horizontal distance covered by the path. This is of course very similar to what we showed in Lemma 4.2, where the sequences discussed in the lemma form a subset of all paths in (D, W_D, Π_D). We define a *critical path* to be a path for which inequality I does not hold, and say S is a solution if $(D - S, W_D, \Pi_D)$ contains no critical paths:

Graph Problem

Input:	(D, W_D, Π_D), respectively a DAG, its weight function, and its potentials function. An integer k.
Question:	Is there a set $S \subseteq V(D)$ of size k such that $(D - S, W_D, \Pi_D)$ contains no critical paths?

We will prove that a solution to the graph problem is indeed equivalent to a solution of the original label placement problem:

Lemma 4.3. *Let $I = \{(p_1, l_1), \ldots, (p_n, l_n)\}$ be an instance of the point-feature labelling problem in the 1-slider model, and let (D, W_D, Π_D) be the corresponding instance of the graph problem. Additionally, let $S \subseteq \{1, \ldots, n\}$. Then $\{(p_i, l_i) \mid i \in S\}$ is a solution to the PFL problem if and only if $\{v_i \mid i \in S\}$ is a solution to the graph problem.*

Proof.

$\Rightarrow$ Assume $S \subseteq \{(p_1, l_1), \ldots, (p_n, l_n)\}$ is a solution of I. Observe that the set of minimal critical paths of the corresponding graph problem is a subset of all sequences P of the PFL-problem, as defined in Lemma 4.2. This is the case because a path is a minimal critical path only if, when we place the labels of only points along this path using the greedy approach, each label influences the next label and only the last label can not be

placed without overlap. Since S is a solution to I, S contains at least one point from such sequences P for which the inequality from Lemma 4.2 does not hold. As such, define $S' := \{v_i | p_i \in S\}$. Then $(D - S', W_D, \Pi_D)$ contains no minimal critical paths. Therefore it does not contain any critical paths, and we conclude that S' is a solution to the graph problem.

$\Leftarrow$ Assume $S' \subseteq \{v_1, \ldots, v_n\}$ is a solution of (D, W_D, Π_D). Then there are no paths in $(D - S', W_D, \Pi_D)$ for which inequality I does not hold. Define $S := \{p_i | v_i \in S'\}$. Observe that the path equivalents of all sequences P of $I - S$ as defined in Lemma 4.2 form a subset of all paths of $(D - S', W_D, \Pi_D)$. This is the case because the label of each point of P influences the label of the next point of P, and we add an arc to the graph representation if the label of one point could potentially influence the label of the next point. As such, there are no such sequences P in $I - S$ for which the inequality of Lemma 4.2 does not hold. Therefore we conclude that S is a solution to the PFL problem I.

$\square$

Indeed finding a solution to the graph representation is equivalent to finding a solution of the same size to the original problem. Now that we have established this, we can disregard the original label placement problem from here on, and instead focus on the graph representation for our algorithm.

5 A Fixed-Parameter Tractable Algorithm for Label Placement in the 1-Slider Model

5.1 The Core Algorithm

Having established that finding a set of vertices in the graph representation that hits all critical paths is sufficient to solve the labelling problem in the 1-slider model, we aim to find an algorithm that finds the smallest such set. As we have shown earlier, the point label placement problem in the 1-slider model is NP-complete, and thus has superpolynomial running time. Fortunately however, it is fixed-parameter tractable, using the feedback edge number as parameter. Specifically, we use the maximum feedback edge number among all connected components of the graph. We choose this parameter to measure the tree-likeness of the graph representation, as we believe it is 'easy' to solve the graph problem when the graph is a tree, requiring only the reduction rules we present later. As such, we will present a FPT-algorithm to prove the following theorem:

Theorem 5.1. *Given a problem instance (D, W_D, Π_D), there exists a $\mathcal{O}(2^k \cdot |V(D)|^2)$ time algorithm that returns a minimum vertex set S such that $D - S$ contains no critical paths. Here, k denotes the maximum feedback edge number among all connected components of D.*

In order to prove this theorem, we will present an algorithm that satisfies its criteria. We formulate and prove the correctness of several reduction rules, of which the branch operation forms the backbone of the algorithm. We briefly discuss how these reduction rules form the algorithm, after which we analyse the running time. Recall that a path $\langle v_1, \ldots, v_e \rangle$ is critical in (D, W_D, Π_D) if the following inequality does not hold:

$$\left(\sum_{i=1}^{e} w_D(v_i)\right) + \pi_D^-(v_1) + \pi_D^+(v_e) \leq \left(\sum_{i=1}^{e-1} arcw_D(v_i, v_{i+1})\right) + w_D(v_1) + w_D(v_e) \tag{I}$$

Our algorithm consists of three reduction rules, and takes as input some graph representation (D, W_D, Π_D). The first reduction rule resolves any vertices that are critical, that is to say paths of a single vertex for which inequality I does not hold. We apply the first reduction rule exhaustively before considering the next reduction rule. The second rule 'collapses' a leaf into its parent, and stores information about the leaf in the potential of the parent. Every time we apply this second rule, we then apply the first rule exhaustively again. Once both the first and second reduction rules are no longer applicable, we use a branch step, after which the entire process is repeated.

- Reduction rule 1: Let v be a vertex of D, forming a path by itself for which inequality (I) does not hold. If such a vertex v exists, remove v from D, and add it to the solution.
- Reduction rule 2: Let l be a leaf vertex of D, and p be the parent of l in D. If $(l, p) \in A(D)$, set $\pi_D^-(p) := \max\left(\pi_D^-(p); \pi_D^-(l) + w_D(p) - arcw_D(l, p)\right)$. Else if $(p, l) \in A(D)$, set $\pi_D^+(p) := \max\left(\pi_D^+(p); \pi_D^+(l) + w_D(p) - arcw_D(p, l)\right)$. Then remove l from D.
- Branch: Let s be a source or sink in D. We recurse on two different branch paths. In the first branch, we remove s from D and add it to the solution. In the second branch path, if s is a source, then for every arc $(s, v) \in A(D)$ we set

  $$\pi_D^-(v) := \max\left(\pi_D^-(v); \pi_D^-(s) + w_D(v) - arcw_D(s, v)\right)$$

  If s is a sink, then instead for every arc $(v, s) \in A(D)$ we set

  $$\pi_D^+(v) := \max\left(\pi_D^+(v); \pi_D^+(s) + w_D(v) - arcw_D(v, s)\right)$$

Afterwards, we remove s from D. We then pick the branch that returns the solution with the smallest size, or of the first branch if the solution sizes are equal.

Clearly Reduction Rule 1 is correct: if a vertex by itself forms a critical path, then to hit that critical path we must include this vertex. Before we prove the correctness of Reduction rule 2 and the branch step, we first prove two closely related lemmata which we will need for our correctness proofs.

Lemma 5.1. *Assume (D, W_D, Π_D) is transformed into $(D', W_{D'}, \Pi_{D'})$ by applying Reduction Rule 2 on arc (l, p), where l is the leaf and p is its parent, such that $\pi_D^-(p) \neq \pi_{D'}^-(p)$. Let P' be a path in D' that starts in p, and let P be the path in D that starts with arc (l, p) and then follows P'. Then inequality I holds for P' in $(D', W_{D'}, \Pi_{D'})$ if and only if it holds for P in (D, W_D, Π_D).*

Proof. In order to prove this lemma, we will show that inequality I for path P' in $(D', W_{D'}, \Pi_{D'})$ can be rewritten to inequality I for path P in (D, W_D, Π_D).

$$\big(\sum_{i=1}^{e} w_{D'}(v_i)\big) + \pi_{D'}^-(v_1) + \pi_{D'}^+(v_e) \leq \big(\sum_{i=1}^{e-1} arcw_{D'}(v_i, v_i+1)\big) + w_{D'}(v_1) + w_{D'}(v_e)$$

Inequality I for path P' in $(D', W_{D'}, \Pi_{D'})$, where $v_1 = p$ is the starting vertex of P', and v_e is the ending vertex of P.

$$\big(\sum_{i=1}^{e} w_D(v_i)\big) + \big(\pi_D^-(l) + w_D(v_1) - arcw_D(l,p)\big) + \pi_{D'}^+(v_e) \leq \big(\sum_{i=1}^{e-1} arcw_D(v_i, v_i+1)\big) + w_D(v_1) + w_D(v_e)$$

We insert the start potential of $v_1 = p$ into the inequality. As arc weights and label widths are never changed, $w_{D'}(v) = w_D(v)$ and $arcw_{D'}(v, w) = arcw_D(v, w)$ for all vertices v, w.

$$\big(\sum_{i=1}^{e} w_D(v_i)\big) + \pi_D^-(l) + \pi_{D'}^+(v_e) \leq \big(\sum_{i=1}^{e-1} arcw_D(v_i, v_i+1)\big) + arcw_D(l,p) + w_D(v_e)$$

Term $w_D(v_1)$ on both sides cancels out, and we move $arcw_D(l, p)$ to the right hand side.

$$\big(\sum_{i=1}^{e} w(v_i)\big) + w(l) + \pi^-(l) + \pi^+(v_e) \leq \big(\sum_{i=1}^{e-1} arcw(v_i, v_i+1)\big) + arcw(l,p) + w(l) + w(v_e)$$

Finally we insert $w_D(l)$ on both sides of the inequality. Since it was unchanged, $\pi_{D'}^+(v_e) = \pi_D^+(v_e)$. As such, our rewriting results in inequality I for path P in (D, W_D, Π_D). Of course, we can also rewrite the other way, and as such we have proven that Lemma 5.1 holds. □

The second lemma is similar to the lemma we have just proved, except that the arc between the leaf and parent is reversed in direction, and is now the last arc of the path:

Lemma 5.2. *Assume (D, W_D, Π_D) is transformed into $(D', W_{D'}, \Pi_{D'})$ by applying Reduction Rule 2 on arc (p, l), where l is the leaf and p is its parent, such that $\pi_D^+(p) \neq \pi_{D'}^+(p)$. Let P' be a path in D' that ends in p, and let P be the path in D that follows P' completely and then ends with arc (p, l). Then inequality I holds for P' in $(D', W_{D'}, \Pi_{D'})$ if and only if it holds for P in (D, W_D, Π_D).*

Proof. Again we show that inequality I for path P' in $(D', W_{D'}, \Pi_{D'})$ can be rewritten to inequality I for path P in (D, W_D, Π_D):

$$(\sum_{i=1}^{e} w_{D'}(v_i)) + \pi_{D'}^-(v_1) + \pi_{D'}^+(v_e) \leq (\sum_{i=1}^{e-1} arcw_{D'}(v_i, v_i+1)) + w_{D'}(v_1) + w_{D'}(v_e)$$

Inequality I for path P' in $(D', W_{D'}, \Pi_{D'})$, where v_1 is the starting vertex of P', and $v_e = p$ is the ending vertex of P.

$$(\sum_{i=1}^{e} w_D(v_i)) + \pi^-_{D'}(v_1) + (\pi^+_D(l) + w_D(v_e) - arcw_D(p,l)) \leq (\sum_{i=1}^{e-1} arcw_D(v_i, v_i+1)) + w_D(v_1) + w_D(v_e)$$

We insert the end potential of $v_1 = p$ into the inequality. As arc weights and label widths are never changed, $w_{D'}(v) = w_D(v)$ and $arcw_{D'}(v,w) = arcw_D(v,w)$ for all vertices v, w.

$$(\sum_{i=1}^{e} w_D(v_i)) + \pi^-_{D'}(v_1) + \pi^+_D(l) \leq (\sum_{i=1}^{e-1} arcw_D(v_i, v_i+1)) + arcw_D(p,l) + w_D(v_1)$$

Term $w_D(v_e)$ on both sides cancels out, and we move $arcw_D(p,l)$ to the right hand side.

$$(\sum_{i=1}^{e} w_D(v_i)) + w_D(l) + \pi^-_D(v_1) + \pi^+_D(l) \leq (\sum_{i=1}^{e-1} arcw_D(v_i, v_i+1)) + arcw_D(p,l) + w_D(v_1) + w_D(l)$$

Finally we insert $w_D(l)$ on both sides of the inequality. Since it was unchanged, $\pi^+_{D'}(v_e) = \pi^+_D(v_e)$. As such, our rewriting results in inequality I for path P in (D, W_D, Π_D). Again, we can also rewrite the other way, and as such we have proven that Lemma 5.2 also holds. □

Now that we have proved these two lemmata, we can apply them in our correctness proofs. First we will prove the correctness of Reduction Rule 2, after which we will show that our branch operation is valid. For the following two lemmata, we will assume reduction rule 1 has been applied exhaustively:

Lemma 5.3. *Let (D, W_D, Π_D) be the graph instance before applying Reduction Rule 2, and $(D', W_{D'}, \Pi_{D'})$ be the graph instance after applying Reduction Rule 2 on leaf l with parent p. Then if (D, W_D, Π_D) has a solution of size k, $(D', W_{D'}, \Pi_{D'})$ also has a solution of size k.*

Proof. Suppose S is a solution of size k to (D, W_D, Π_D), and define S' as follows:

- if $l \in S$, then $S' := (S \setminus \{l\}) \cup \{p\}$
- if $l \notin S$, then $S' := S$

Note that in both cases, $|S'| = |S| = k$. We claim S' is a solution to $(D', W_{D'}, \Pi_{D'})$, that is $D' - S'$ contains no critical paths. Assume, for contradiction, that $P' = \langle v_1, \ldots, v_e \rangle$ is a critical path in $D' - S'$. We use a case distinction on P':

- P' does not start or end with p:
  Then all vertices of P' are also in $D - S$, per definition of S'. Furthermore, except for p if $p \in P'$, the potentials of all vertices in P' are also the same in $\Pi_{D'}$ and Π_D. Thus the only thing that could have changed in P' between $D - S$ and $D' - S'$ is the potential of p. However, as p is not the end or start of P', inequality I is the same for P' in $D - S$ and $D' - S'$. Hence P' is also a critical path in $D - S$, yielding a contradiction.

- P' starts with p:
  Since P' is a path in $D' - S'$ and contains p, per definition of S', we have that $l, p \notin S' = S$. Clearly then, all vertices of P' are also in $D - S$. Furthermore note that Reduction Rule 2 must have set $\pi^-_{D'}(p) = \pi^-_D(l) + w_D(p) - arcw_D(l,p)$, for if it did not, the potentials of all vertices in P' would be the same in $D' - S'$ and $D - S$, thus P' would also be a critical path in $D - S$. Observe that this also implies $(l,p) \in A(D)$, otherwise Reduction Rule 2 could not have set $\pi^-_{D'}(p)$. Since P' is a critical path in $D' - S'$, that means inequality I does not hold. However, by Lemma 5.1 this inequality can be rewritten to inequality I for $P = \langle l, v_1 = p, \ldots, v_e \rangle$ in $D - S$. Therefore inequality I also does not hold for P in $D - S$, implying it is a critical path in $D - S$, and we have a contradiction.

- P' ends with p:
  Since P' is a path in $D'-S'$ and contains p, per definition of S', we have that $l, p \notin S' = S$. Clearly then, all vertices of P' are also in $D - S$. Furthermore note that Reduction Rule 2 must have set $\pi^+_{D'}(p) = \pi^+_D(l) + w_D(p) - arcw_D(p, l)$, for if it did not, the potentials of all vertices in P' would be the same in $D' - S'$ and $D - S$, thus P' would also be a critical path in $D - S$. Observe that this also implies $(p, l) \in A(D)$, otherwise Reduction Rule 2 could not have set $\pi^+_{D'}(p)$. Since P' is a critical path in $D' - S'$, that means inequality I does not hold. However, by Lemma 5.2 this inequality can be rewritten to inequality I for $P = \langle v_1, \ldots, v_e = p, l\rangle$ in $D - S$ (see below). Therefore inequality I also does not hold for P in $D - S$, implying it is a critical path in $D - S$, and we have a contradiction.

□

Lemma 5.4. *Let (D, W_D, Π_D) be the graph instance before applying Reduction Rule 2, and $(D', W_{D'}, \Pi_{D'})$ be the graph instance after applying Reduction Rule 2 on leaf l with parent p. Then if S' is a solution to $(D', W_{D'}, \Pi_{D'})$, S' is also a solution to (D, W_D, Π_D).*

Proof. Let S' be a solution to D'. Assume, for contradiction, that $P = \langle v_1, \ldots, v_e\rangle$ is a critical path in $D - S'$. We use a case distinction on P:

- if P does not start or end with l or p:
  Then all vertices of P are also in $D' - S'$. Note that since l is a leaf and P does not start or end with l, $l \notin P$. Thus, except for p if $p \in P$, the potentials of all vertices of P are the same in $D - S'$ and $D' - S'$. Therefore, the only thing that could have changed in P between $D - S'$ and $D' - S'$ is the potential of p. However, since P does not start or end with p, inequality I is the same for P in $D - S'$ and $D' - S'$. Hence, P is also a critical path in $D' - S'$, yielding a contradiction.

- if P starts or ends with p, and does not end or start with l:
  Then again $l \notin P$, and all vertices of P are also in $D' - S'$. Note that, per the definition of Reduction Rule 2, the potentials of p can never decrease when we apply this reduction rule. Furthermore, all other potentials remain the same. Thus if P does not satisfy inequality I in $D - S'$, its potentials did not decrease, and P also does not satisfy the inequality in $D' - S'$. As a result, P is also a critical path in $D' - S'$ and we have a contradiction.

- if P starts with l: Note that l cannot be the only vertex of P, otherwise it would have been resolved before applying the reduction rule. Let $P' = \langle v_2 = p, \ldots, v_e\rangle$. If the reduction rule did not set $\pi^-_{D'}(p)$ to $\pi^-_D(l) + w_D(p) - arcw_D(l, p)$, then $\pi^-_{D'}(p) = \pi^-_D(p) \geq \pi^-(l) + w(p) - arcw(l, p)$. Otherwise the reduction rule set $\pi^-_{D'}(p)$ to $\pi^-_D(l) + w_D(p) + arcw_D(l, p)$. In both cases, per Lemma 5.1 we can rewrite inequality I for P' in $D' - S'$ using $\pi^-_{D'}(p) = \pi^-_D(l) + w_D(p) + arcw_D(l, p)$ to inequality I of P in $D - S'$. Thus inequality I does not hold for P' in $D - S'$, making it a critical path, and we have a contradiction.

- if P ends with l: Similarly to the previous case, l cannot be the only vertex of P, otherwise it would have been resolved before applying the reduction rule. Let $P' = \langle v_1, \ldots, v_{e-1} = p\rangle$. If the reduction rule did not set $\pi^+_{D'}(p)$ to $\pi^+_D(l) + w_D(p) - arcw_D(p, l)$, then $\pi^+_{D'}(p) = \pi^+_D(p) \geq \pi^+_D(l) + w_D(p) - arcw_D(p, l)$. Otherwise the reduction rule set $\pi^+_{D'}(p)$ to $\pi^+_D(l) + w_D(p) + arcw_D(p, l)$. In both cases, per Lemma 5.2 we can rewrite inequality I for P' in $D' - S'$ using $\pi^+_{D'}(p) = \pi^+_D(l) + w_D(p) + arcw_D(p, l)$ to inequality I of P in $D - S'$. Thus inequality I does not hold for P' in $D - S'$, making it a critical path, and we have a contradiction.

□

Lemma 5.3 and Lemma 5.4 together prove the correctness of Reduction Rule 2. Since we chose our parameter because it is a measure of the tree-likeness of the graph, it might be interesting to show that it is indeed easy to solve the graph problem if the parameter is 0:

Lemma 5.5. *If the feedback edge number is 0, and the graph therefore a forest, we can solve the graph problem using only exhaustive application of the reduction rules.*

Proof. Apply Reduction Rule 1 exhaustively. If the graph is then disconnected, we can solve each connected component separately and combine the results. So assume the graph is a tree and Reduction Rule 1 has been applied exhaustively. Then there must be a leaf l with parent p, to which we apply Reduction Rule 2. If l was the last leaf attached to p, then p is now a leaf; otherwise we keep applying Reduction Rule 2 to the leaves of p until we remove the last. If at any point p becomes a critical path by itself due to application of Reduction Rule 2, it is removed by Reduction Rule 1, and all of its leaves will become isolated vertices. For each of these isolated vertices then, either Reduction Rule 1 is applicable to that vertex, or there is no critical path in that connected component of one vertex. In the former case, the vertex is the solution to that connected component consisting only of that vertex; in the latter case the solution of that component is empty. We can iteratively repeat this procedure to solve the entire graph. □

With that out of the way, we can validate the branch step. We do this by again proving two lemmata.

Lemma 5.6. *Let (D, W_D, Π_D) be the graph problem instance after exhaustively applying the reduction rules. Then both branch solutions are a solution to (D, W_D, Π_D).*

Proof. Let v_b be the source/sink on which we branch. We handle both branches seperately:

- Branch where v_b is put into the solution: This case is trivial, clearly if S is a solution to $(D - \{v_b\}, W_D, \Pi_D)$ (the recursive call for this branch), then $S \cup \{v_b\}$ is a solution to (D, W_D, Π_D).
- Branch where v_b is not in the solution: Recall that v_b is a source or a sink. Assume S is a solution to $(D', W_{D'}, \Pi_{D'})$, the instance generated by this branch. For contradiction, assume S is not a solution to (D, W_D, Π_D), and let P be a critical path in $D - S$. Then P must start or end in v_b, for if it did not, P would also be a critical path in $D' - S$; after all, the potentials of the vertices of P are never lower in $\Pi_{D'}$ than they are in Π_D. Similarly to the proof of Lemma 5.4, we can then consider the path P' obtained by removing the arc incident on v_b from P. Let u be the other vertex to which this removed arc was incident. The potential of u in $\Pi_{D'}$ is then equal or greater to the contribution of v_b to inequality I. Again similar to the proof of Lemma 5.4, we can rewrite inequality I for P in $(D - S, W_D, \Pi_D)$ (which does not hold) to that inequality for $(D' - S, W_{D'}, \Pi_{D'})$ such that it also does not hold (per Lemma 5.1 and Lemma 5.2). As such, $D' - S$ would also contain a critical path, giving us a contradiction.

□

Lemma 5.7. *Let (D, W_D, Π_D) be the graph problem instance after exhaustively applying the reduction rules. Furthermore, let v_b be the vertex on which we branch, let $(D - v_b, W_D, \Pi_D)$ be the branch instance where v_b is in the solution, and let (D, W_D, Π'_D) the branch instance where v_b is not in the solution. If (D, W_D, Π_D) has a solution of size at most k, then either $(D - v_b, W_D, \Pi_D)$ has a solution of size $k - 1$, or (D, W_D, Π'_D) has a solution of size k.*

Proof. Assume S is a solution of size k to (D, W_D, Π_D). If $v_b \in S$, then clearly $S \setminus \{v_b\}$ is a solution of size $k-1$ to $(D - v_b, W_D, \Pi_D)$. Thus consider the case where $v_b \notin S$. We claim that S is then also a solution to $(D - v_b, W_D, \Pi'_D)$. Assume, for contradiction, that P' is a critical path in $((D - v_b) - S, W_D, \Pi'_D)$. If P' does not start or end in a neighbour of v_b, then P' is also a critical path in (D, W_D, Π_D), for the potentials of all its vertices are the same in Π_D and Π'_D. So assume P' starts or ends in a neighbour u of v_b. Then let P be the path starting with arc (v_b, u) or ending with arc (u, v_b) in D. As the potential of u in Π'_D is equal or greater to the contribution of v_b in P for inequality I, we can follow an argument similar to what we used in the proof of Lemma 5.3: Since inequality I does not hold for P' in (D, W_D, Π'_D), when we rewrite this to the inequality for P in (D, W_D, Π_D) (using Lemma 5.1 and Lemma 5.2), it also does not hold. As such, P is a critical path in (D, W_D, Π_D), and we obtain a contradiction. □

Thus, with Lemma 5.6 and Lemma 5.7 combined we have shown the correctness of our branch operation. Now we combine the correctness proofs of our operations to show that our algorithm as a whole is correct:

Lemma 5.8. *Given an instance of the graph problem (D, W_D, Π_D), our algorithm finds a solution of minimum size.*

Proof. We prove this by induction. In the base case, the graph is a single vertex. Then the algorithm is correct (that is, returns a solution of minimum size): if the vertex is a critical path by itself, it will be put in the solution by Reduction Rule 1, and otherwise the empty set is returned as solution. We assume the algorithm is correct for graphs of up to $n-1$ vertices, and make the inductive step for the case where the graph D has n vertices. We will consider the three possible actions we can take: appying Reduction Rule 1, Reduction Rule 2, or branch.

- Per correctness of Reduction Rule 1, we know that if we apply this reduction rule to a vertex v of D, then v together with an optimal solution to $D - v$ is an optimal solution to D. By induction, our algorithm finds an optimal solution to $D - v$.
- If we apply Reduction Rule 2, then per Lemma 5.3 we know that the minimum size of a solution to the graph before and after applying the reduction rule is the same. Furthermore, by Lemma 5.4 we know that a solution to the graph after applying Reduction Rule 2 is also a solution to the graph before applying the reduction rule. Again by induction, our algorithm finds a minimum size solution to the graph after applying the reduction rule (since a vertex is removed), and thus also to the graph before applying the reduction rule.
- The last option is to branch. Per Lemma 5.7 we know that one of the branch instances has a solution of minimum size (possibly including the branch vertex itself). From Lemma 5.6 we know that this branch solution is then also a solution to the graph before branching. By induction, our algorithm gives the minimum size solutions to both branch instances, and as such our algorithm also gives the minimum size solution to the graph before branching.

Since the inductive step is correct for all three possible actions, we can conclude that our algorithm will indeed always find a solution of minimum size. □

What remains is to analyse the running time of our algorithm. To make this slightly easier, we first give pseudocode of our algorithm:

CoreAlg (D, W_D, Π_D)

1. Exhaustively apply Reduction Rule 1

if(undirected graph underlying D contains multiple connected components $C_1, \ldots, C_i$)
    Return **CoreAlg**$(C_1, W_D, \Pi_D) \cup \ldots \cup$ **CoreAlg**(C_i, W_D, Π_D)
while(graph contains a leaf)
    Apply Reduction Rule 2 to a leaf
    Exhaustively apply Reduction Rule 1
if(feedback edge number $\neq 0$)
    $S_1 :=$ recursive call for branch 1, $S_2 :=$ recursive call for branch 2
    if$(|S_1| \leq |S_2|)$ add set S_1 and v_b to S, where v_b is the branch vertex
    else add set S_2 to S
Return S

In order to analyse the running time of the algorithm, we first need to verify that it terminates. That is to say, we need to show that the branch operation is guaranteed to decrease the parameter when applied. As such, we want to prove the following lemma:

Lemma 5.9. *Let $f(H)$ denote the maximum feedback edge number among all connected components of some graph H. Let G be a connected, undirected graph without vertices of degree 1, such that G contains at least one cycle. Furthermore, let v be an arbitrary vertex of G. Then $f(G - v) < f(v)$.*

Proof. First observe that a set of edges X is a feedback edge set of G if and only if $G - X$ is acyclic. If $G - X$ is disconnected, then X is not a feedback edge set of minimum size. For consider two different connected components of $G - X$. Then there must be an edge in G and included in X, that connects these two components, since G is connected. As such, if X is a feedback edge set of minimum size, then $G - X$ is a tree. The other way around, if $G - X$ is a tree, then X must be a feedback edge set of minumum size. For consider removing any edge in X: if it is on a cycle in G, then X is no longer a feedback edge st of G. If it is not on a cycle, then X was not of minimum size. Thus X is a feedback edge set of minimum size if and only if $G - X$ is a (connected) tree, or equivalently if and only if $E(G) - X$ is a spanning tree of G. With this in mind, we apply a case distinction on vertex v:

- v lies on a cycle in G: Let u be a neighbour of v such that $\{u, v\}$ lies on a cycle in G. Then the graph G' with $V(G') = V(G)$ and $E(G') = E(G) - \{u, v\}$ is connected. Let T be a spanning tree of G'. Then $E(G') - T$ is a minimum feedback edge set of G'. Since $\{u, v\}$ lies on a cycle in G, T is also a spanning tree of G, and $E(G) - T$ is a minimum feedback edge set of G. Since $|E(G) - T| = |E(G') - T| + 1$ (the former contains $\{u, v\}$ while the latter does not), it follows that the feedback edge number of G' is smaller than that of G. As $G - v$ is a subgraph of G', its feedback edge number cannot exceed that of G', and is therefore smaller than that of G. Furthermore note that the maximum feedback edge number among connected components of $G - v$ is at most equal to that of $G - v$ itself, and thus smaller than that of G. Since G was connected, this means that $f(G - v) < f(G)$.

- v does not lie on a cycle in G: We claim that $G - v$ is then disconnected, and v has exactly one neighbour in every connected component of $G - v$. First we show that $G - v$ is disconnected, by contradiction. Assume $G - v$ is connected, and let u and w be two different neighbours of v. Then there is a path from u to v, since we assumed $G - v$ is connected. If we add the arcs $\{u, v\}$ and $\{w, v\}$ to this path, we obtain a cycle in

G, which contradicts our assumption that v does not lie on a cycle in G. So, $G - v$ is indeed disconnected. Next we show that v has exactly one neighbour in every connected component of $G - v$, again by contradiction. Assume there is a connected component of $G - v$ containing two distinct neighbours u, w of v. Then there is a path between u and w in that component. Similarly to before, this implies that there was a cycle through v, u and w in G, giving us a contradiction. Indeed $G - v$ is disconnected, and v has exactly one neighbour in every connected component of $G - v$.

Now we make another claim: each connected component of $G-v$ contains a cycle. Assume, for contradiction, that C is a connected component of $G-v$ that does not contain a cycle. Then C must contain at least two vertices, for if it would contain only a single vertex, then that vertex would have only had v as neighbour in G and would therefore be a leaf; this contradicts our initial assumption on G. So assume C has at least two vertices and contains no cycles. Observe that C must then have at least two leaves: any maximal length path in C must start and end with a leaf, since C contains no cycles. Since these two leaves were not leaves in G, by our initial assumption on G, both of these leaves must be neighbours of v in G. As such, v would have two neighbours in connected component C, contradicting our previous claim that v has exactly one neighbour in each connected component of $G - v$.

Combining the two claims we have proven, we know that $G - v$ consists of at least two connected components, and each of these components contains a cycle. Let X then be a minimum feedback edge set of G. Then for each connected component C of $G - v$, we have that $X \cap E(C)$ is a feedback edge set for C, else X would not be a feedback edge set for G. Since every connected component of $G - v$ contains at least one cycle, X must contain at least one edge of each of those components. As such, $|X \cap E(C)| < X|$ for each connected component C of $G-v$, and therefore the maximum feedback edge number among all connected components of $G - v$ is strictly smaller than that of G. Since G is connected, we can then conclude that $f(G - v) < f(G)$.

□

Indeed, applying the branch operation decreases the parameter by at least 1. Now we aim to prove the following lemma:

Lemma 5.10. *Given an instance (D, W_D, Π_D) of the graph problem, and k being the maximum feedback edge number among all connected components of D, our algorithm returns a minimum size solution in time $\mathcal{O}(2^k \cdot |V(D)|^2)$.*

Proof. Since we handle each connected component separately, we will first analyse the running time for one such component C. In the theoretical worst case, the parameter is only decreased by the branch operation, and only decreased by one every time we apply this operation. As such, we may have up to 2^k recursive calls, since by Lemma 5.5 we can solve the problem in polynomial time once the parameter is 0. Using a naive approach, each of these recursive calls would take up to $\mathcal{O}(|V(C)|^2)$ time: applying Reduction Rule 1 or 2 once can take up to $|V(C)|$ time, and since we remove a vertex with both rules, together they can be applied a maximum of $|V(D)|$ times. Furthermore, the branch operation itself takes time proportional to the degree of the branch vertex, thus $|A(C)| \leq |V(C)|^2$. As such, we can bound the running time of our algorithm for connected component C as $\mathcal{O}(2^k \cdot |V(C)|^2)$. The overall running time then is the sum of the running times for each connected component, and since the sum of the number of vertices in each component equals the total number of vertices, this can never be more than $\mathcal{O}(2^k \cdot |V(D)|^2)$. From Lemma 5.8 we know that our algorithm returns the minimum size solution, so we can conclude that Lemma 5.10 holds. □

Lemma 5.10 allows us to conclude that Theorem 5.1 indeed holds. As such, using Lemma 4.3 we can say that the graph problem, and thus its PFL problem, are fixed-parameter tractable with the maximum feedback edge number among all connected components of the graph as parameter, and can be solved in time $\mathcal{O}(2^k \cdot n^2)$, where n is the number of vertices or points.

5.2 Additional Options

So far in this section, we have presented our core FPT-algorithm. We were however interested in techniques that could potentially yield a significant speed up of our algorithm in practice. As such, we present a more specific method of finding a source or sink for the branch step, as well as a dynamic programming approach to remove vertices that will never be critical.

In our core algorithm, once we have exhaustively applied our reduction rules, we pick an arbitrary source or sink vertex to branch on. It could, however, very well be that branching on one source or sink will lead to a greater drop in the feedback edge number than another source or sink. We propose to find a source or sink vertex that is a cut point, and also lies on an undirected cycle. A cut point is a vertex that, when removed from the graph, increases the number of weakly connected components. If no such vertex exists, we settle for any cutpoint, or lacking that as well an arbitrary source or sink. What we aim to make use of here is that, when a graph is split into multiple weakly connected components, then the parameter must drop (see Lemma 5.9). Since we attempt to find a vertex that both lies on a cycle and splits the graph into multiple components, we hope this will speed up the algorithm in practice. To find such an alternative source or sink, we compute the set of cut points using the algorithm by Hopcroft and Tarjan [14], and then iterate over this set checking for the desired properties. This takes $\mathcal{O}(|V| + |E|)$ time.

Another option we would like to investigate, is whether or not removing arcs and vertices that can never be part of a critical path gives us a worthwile speed up. In order to find such arcs and vertices, we use a dynamic programming approach to calculate the maximum start- and end-potential that each vertex could possibly ever have. For every arc and every vertex, we then check whether inequality I holds. If it does, we can be sure the arc or vertex can never be part of a critical path, and is therefore irrelevant and can be removed safely.

For our dynamic programming approach, we construct two tables. In one table, we will compute the maximum start potential each vertex can ever have, and in the other table the maximum end potential each vertex can ever have. In order to compute the table for maximum start potentials, we first find a topological ordering of the vertices, such that if $(v, u) \in A(D)$, $v < u$ in the ordering. Then, in the order of this topological ordering, we compute the entry of each vertex:

$$\text{DPT1}[u] := \max\big(\pi_D^-(u); \text{DPT1}[u] + w_D(u) - arcw_D(v,u)\big)\text{for}(v,u) \in A(D)$$

For the second table, we simply invert the entire graph D, and then construct it the same way as the first table, except we use $\pi_D^+(u)$ instead of $\pi_D^-(u)$. Then for every arc (v, u), if $\text{DPT1}[v] + \text{DPT2}[u] \leq arcw_D(v, u)$ does not hold, we know that (v, u) can never be part of a critical path, and we remove it. Furthermore, for each vertex v, if $\text{DPT1}[v] + \text{DPT2}[v] \leq w_D(v)$ holds, then we also know that v can never be part of a critical path, and again we remove it.

Finding the topological ordering takes $\mathcal{O}(|V(D)| + |A(D)|)$ time. In the construction of each table, we compute a single entry per vertex. For each entry we compute, we have to look at all the incoming edges of the vertex, thus overall the construction of each table also takes $\mathcal{O}(|V(D)| + |A(D)|)$ time. Lastly, checking whether or not we should remove each vertex and

edge again takes the same time, resulting in an overall running time of $\mathcal{O}(|V(D)| + |A(D)|)$ for the entire dynamic programming procedure.

6 Experimental Setup

Up to this point, we have discussed only the theoretical side of our FPT-algorithm. However, our aim has always been to discover how it performs in practice. To this end, we have implemented our algorithm, as well as the additional options mentioned in the previous section. We then obtained information on the performance of our algorithm on various data sets. First, we will discuss the details of our implementation, after which we will present our datasets and experiment method, and analyse the parameter values of the data sets.

6.1 Implementation

We implemented our FPT-algorithm in Java, version 1.8.0_25, using the NetBeans 8.0.1 IDE. In our implementation, we used graph data structures as well as graph algorithms from the JGraphT 0.9.1 library [18]. Graphs are presented as a graph data structure from JGraphT (SimpleWeightedDirectedGraph, specifically), and we keep an array of potentials. This array of potentials is copied for every recursive call, such that changing a potential in one recursive call does not affect the potential of another call. Every edge of the graph has a weight, and the vertices are presented as objects containing a label width and a pointer to an entry in the potentials array.

As mentioned in the description of the algorithm, we handle each connected component separately. This way we reduce both the size of the graph and the parameter of any instance of the algorithm. Furthermore, to prevent having to iterate over the entire set of vertices every time we apply Reduction Rule 2, we maintain an arraylist of leaves. For similar reasons, we call Reduction Rule 1 only on vertices that were recently affected by Reduction Rule 2 or the branch operation (that is to say, whose potentials were changed). This makes it such that applying either reduction rule takes only constant time, since we can simply take the first leaf of the leaf list for Reduction Rule 2. Since the degree of a leaf is 1, we only apply Reduction Rule 1 to at most 1 vertex for each time we use Reduction Rule 2. As such, exhaustively applying the reduction rules takes time linear in the number of vertices, compared to the $\mathcal{O}(n^2)$ of the naive approach.

6.2 Datasets and Experiments

For our experiments, we used four datasets. All four data sets consisted of triples containing an x-coordinate, y-coordinate, and text label, and were taken from the General Map Labelling webpage of Alexander Wolff [24].

- German Stations: A dataset of 366 train stations across the entire country of Germany. Every label is the name of a train station.
- Berlin Shops: This dataset consists of 336 antique and arts shops in the city of Berlin. All labels of this dataset are shop names.
- U.S. Cities: A large dataset of 1158 cities in the United States, including Alaska and Hawaii. Each label is a city name followed by the state in which it lies.
- U.S. Cities Abrv.: Similar to the U.S. Cities data set, consisting of 1041 cities in the United States, excluding Alaska and Hawaii. Each label is a three character city code.

As the dominating factor in our theoretical run time is the parameter, we analysed the parameter values of the datasets. As such, we looked at the maximum feedback edge number

Figure 5: Maximum feedback edge number among connected components of the graph problem for the German Stations dataset. Font size on horizontal axis.

Figure 6: Maximum feedback edge number among connected components of the graph problem for the Berlin Shops dataset. Font size on horizontal axis.

Figure 7: Maximum feedback edge number among connected components of the graph problem for the US Cities dataset. Font size on horizontal axis.

Figure 8: Maximum feedback edge number among connected components of the graph problem for the US Cities abrv dataset. Font size on horizontal axis.

among connected components of the graph problem for each dataset. This information can be seen in Figures 5, 6, 7, and 8.

For every dataset, we run experiments for two picture formats: A4 format at 300dpi (3508 x 2480 pixels), and A3 format at 300dpi (4962 x 3508 pixels). We choose the orientation of the picture that best fits the dataset. For each format, we run a set of experiments for each font size from 5 to the max font size for which we could conduct the experiment within reasonable time. These max font sizes were determined by a combination of trial and error, and educated guessing based on some initial testing. The latter revealed we could handle the font size for which the parameter value was up to roughly 110. Each set of experiments consists of four experiments: The core algorithm, the core algorithm with removal of noncriticals, the core algorithm with alternative source/sink method, and the core algorithm with both alternative options.

For each individual experiment, we collect data on the following: the parameter drops throughout the entire algorithm, the parameter drops of the branch path that resulted in the solution, and the time it took to run that experiment. The last is measured from the moment we start reading the data file, until the moment we have computed the solution. Due to time constraints, we only ran each experiment once. All experiments were performed on an Intel® Core™ i7-2670QM, 2.2GHz CPU, operating under the Windows 7 Enterprise 64-bit OS. We will present and analyse the results in the next section.

7 Results and Discussion

7.1 Experiment Results

The tables we present next contain the results of our experiments. Each table is titled with the name of a dataset and the picture format. From left to right, the table columns contain the following information:

- **fsize:** the font size used for the labels.
- **ssize:** the size of the solution given by our algorithm.
- **pdrop:** the average drop in parameter caused by the branch step, using the standard source/sink finding method, over the entire algorithm. That is to say, the difference between the feedback edge number of the current connected graph, and the maximum feedback edge number amongst all connected components after the branch step.
- **pdrop2:** the average drop in parameter caused by the branch step, using the alternative source/sink finding method, over the entire algorithm. Again this is the difference between the feedback edge number of the current connected graph, and the maximum feedback edge number amongst all connected components after the branch step.
- **spdrop:** the average drop in parameter caused by the branch step, using the standard source/sink finding method, for only the branch path yielding the solution.
- **spdrop2:** the average drop in parameter caused by the branch step, using the alternative source/sink finding method, for only the branch path yielding the solution.
- **branch:** the total number of branch steps, over the entire standard algorithm.
- **t:** the execution time of the standard algorithm.
- **t2:** the execution time of the algorithm, with removal of vertices and arcs that can never be part of a critical path.
- **t3:** the execution time of the algorithm, with removal of noncriticals and using alternative source/sink method.

If a table entry is given as '–', this means that we were unable to run the experiment with the settings for that particular entry, but were able to run experiments with different settings for the same font and picture format. Generally this occured due to Java running out of heap space, and/or the experiments taking extraordinarily long.

Germany Stations A4

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	4	1.00	1.00	1.00	1.00	12	16	16	15
15	19	1.92	1.92	1.97	1.97	226	31	31	47
20	34	1.95	1.89	1.94	1.94	584	62	94	109
25	54	2.02	2.02	2.08	2.08	3976	171	172	249
30	79	1.53	1.89	2.22	2.24	143318	3900	1622	3267
33	89	2.02	2.71	2.58	2.63	6323978	199712	19469	11762
35	99	2.18	2.19	2.48	2.45	5369166	178792	36426	87610

Germany Stations A3

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	0	1.00	1.00	1.00	1.00	4	<1	<1	<1
20	18	1.52	1.52	1.74	1.74	196	62	46	47
30	37	1.84	1.80	1.99	1.98	1540	125	140	140
35	52	1.72	1.54	2.02	2.02	5694	203	188	171
37	85	1.71	1.78	2.69	2.66	286488	8393	2059	3573
38	88	1.79	1.90	2.69	2.72	290472	8642	2153	3744
39	93	–	2.31	–	2.66	–	–	37627	27784
40	95	–	1.42	–	2.65	–	–	143640	289100

Table 1: Results for the Germany Stations dataset.

Berlin Shops A4

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	27	1.85	1.87	2.08	2.05	630	92	104	47
6	33	1.88	1.76	2.10	2.08	1200	94	109	63
7	41	2.14	1.61	2.11	2.13	14216	562	187	94
8	43	1.95	1.70	2.25	2.31	38846	1263	468	187
9	59	2.23	2.23	2.64	2.64	2467252	86564	59078	115893
10	65	2.15	2.42	2.84	2.84	14522974	518576	376756	771343

Berlin Shops A3

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	17	1.67	1.67	1.75	1.75	150	16	31	31
7	23	1.89	1.95	1.95	1.80	528	15	16	31
9	35	1.53	1.87	2.10	2.05	5446	219	109	109
11	43	2.05	1.62	2.35	2.39	40226	1279	483	188
13	50	1.41	1.37	2.38	2.38	550200	20139	13572	12761
15	64	1.41	1.41	2.72	2.95	3395462	112086	62119	146375

Table 2: Results for the Berlin Shops dataset.

US Cities A3

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	76	2.47	2.54	1.91	2.00	160380	9033	9298	14477
6	96	2.15	2.05	2.21	2.18	246534	15214	15771	28503

Table 3: Results for the US Cities dataset.

US Cities abrv A4

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	8	1.25	1.25	1.09	1.09	204	141	171	156
7	20	1.54	1.54	1.48	1.48	34444	1576	484	671
9	46	1.93	1.93	1.90	1.90	39780	2028	2028	3073
11	56	2.19	2.20	2.07	2.07	117856	6131	6833	11372
13	69	2.19	2.22	2.16	2.20	213064	11685	11622	18689

US Cities abrv A3

fsize	ssize	pdrop	pdrop2	spdrop	spdrop2	branch	t	t2	t3
5	1	1.00	1.00	1.00	1.00	14	15	31	78
8	8	1.02	1.02	1.15	1.15	964	156	140	171
11	24	1.43	1.43	1.60	1.60	51642	2543	1498	2340
14	47	1.93	1.93	1.97	1.97	96524	4524	2168	3385
16	59	1.84	1.23	1.98	2.00	135444	6583	6178	8782
18	75	1.93	1.93	2.26	2.26	560816	26689	27987	53649

Table 4: Results for the US Cities abrv dataset.

In Figure 9, we show an example of what the label drawing created by our algorithm looks like. This particular picture is A3 format, font size 38, for the German Stations dataset.

Figure 9: Resulting label drawing for A3 format, font size 38, of the German Stations dataset.

7.2 Results Discussion

From the tables with experimental results given just before this, several things can be noticed. First of all, we can see that the execution time of the algorithm with removal of noncriticals is never significantly worse than the execution time of the standard algorithm, but it often outperforms the standard algorithm. As we suspected, using this dyamic programming technique can save significant time in practice, at least for real-world examples. We suspect that, in those cases where the removal of noncriticals yields a similar time to the standard algorithm, there are simply very few vertices and arcs who do not lie on any critical paths.

Secondly, we notice that the alternative source/sink finding method gives mixed results. In some cases, it increases the parameter drop significantly, and also further decreases the execution time combined with the removal of noncriticals (see for example font size 30 and 33, A4 size,

of the German Stations dataset in Table 2). However, on other occasions the opposite is true: the average parameter drop only decreases when we apply the alternative source/sink method, and the execution time is only increased (for example font size 16, A3 size, of the US Cities abrv dataset in Table 4). In the latter case, we think this may be caused by the cut vertex source/sink splitting the graph in one connected component of very small size and one connected component of very large size. If there would then have been a source/sink that is not a cut vertex but which lies on many different undirected cycles, then that source/sink would likely be a better choice. From our results we gather that this happens fairly frequently in practice, negatively influencing the usefulness of our alternative source/sink approach. When we took a closer look at an example where the alternative method caused an improvement (font size 30, A4 size, German Stations), we found that the total number of branches for the alternative approach is only 57142, compared to the 143318 of the standard method. For an example where the alternative method only made things worse (font size 16, A3 size, US Cities abrv), we found that the total number of branches for the alternative method is 88406, compared to the greater 135444 of the standard method. Yet for the former example, the run time improved, and for the latter example it deteriorated. It seems that there is a definite trade-off between time saved from getting fewer branches, and the extra time it costs to choose the alternative source or sink.

Furthermore, we observe that generally speaking, the execution time increases faster and faster as the parameter increases, similarly to what one might expect from an algorithm with theoretical run-time exponential in the parameter. A notable exception is the step from font size 33 to font size 35, in the A4 version of the German Stations experiments (Table 2). Even though the parameter increased, the execution time for font size 33 is larger than the time for font size 35, for the standard algorithm. Notably, the execution times of the algorithm with the removal of noncriticals deviates from this, and does seem to follow the general trend. We are as of yet unsure what may be the cause of this, but since each experiment was only performed once, there is a possibility that there was some interference from a background process. Also, as one might expect, there seems to be a strong relation between the total number of branches and the execution time of the standard algorithm, for larger parameter values. For example, for the A3 size of the US Cities abrv dataset in Table 4, especially for the larger font sizes it seems that the number of branches increases with roughly the same factor as the standard execution time.

Summarising, it appears that applying the additional technique where we remove vertices and arcs that are not part of any critical path can never hurt, and often results in a significant speed-up. However, we do not recommend using the alternative source/sink finding method, as it could have a negative impact, and its influence seems unpredictable. Overall, real-world examples generally (in our case 3 out of 4) seem to have acceptable parameter values, and since a map is usually made to last for a long period of time, the relatively long running times do not seem a major issue. However, for certain datasets the parameter value is simply too high, in which case one of the many approximation algorithms would be the better option. Fortunately this can be discovered in advance, as finding the parameter value of a dataset takes relatively little time and effort.

8 Conclusion

We have given motivation, through examples of practical applications and a look at the existing literature, for the relevancy of parameterising the point-feature labelling problem. We proved that it is NP-complete in the 1-slider model, and as such aimed to find a fixed-parameter tractable algorithm for it. This we have achieved, as we presented a $\mathcal{O}(2^k \cdot |V(D)|)$ algorithm for the parameterisation of the underlying graph problem. Furthermore, we proved the correctness of this algorithm, and the way we present the PFL problem as a graph problem.

We implemented our algorithm, and presented two techniques that we thought could speed up the algorithm in practice. From our experiments on datasets of four real-world examples, we concluded that removing vertices and arcs that are not part of any critical path is well worth the effort, but the alternative source/sink finding method is not. Overall, for data sets where the parameter is not exceedingly high, our algorithm can be worth the extra running time.

There are several questions we have left unasnwered. For example, is the PFL problem also parameterisable for the 2-slider and 4-slider model, and if so can we use a similar approach as the one we used here? We are also interested in practical extensions to the algorithm we have presented. These extensions could include having some additional weight function measuring the importance of the labels, in which case the goal would be to minimise the weight of the labels we have to omit. Another extension could be handling obstacles. Perhaps we would prefer not to place labels over rivers, in a geographical map, for example.

It may also be of interest to see if the problem we have presented in this paper can be parameterised using a parameter other than the feedback edge number, for example the treewidth, or perhaps the size of the solution. Lastly, there may be more efficient techniques than what we have presented to speed up the algorithm in practice.

References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11(3-4):209–218, Dec. 1998.

[2] A. C. Alvim and ric D. Taillard. POPMUSIC for the point feature label placement problem. *European Journal of Operational Research*, 192(2):396 – 413, 2009.

[3] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):773–780, Sept 2006.

[4] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, Sept. 2010.

[5] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics (TOG)*, 14(3):203–232, 1995.

[6] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[7] H. A. Do Nascimento and P. Eades. User hints for map labeling. *Journal of Visual Languages & Computing*, 19(1):39–74, 2008.

[8] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. Moret, and B. Zhou. Map labeling and its generalizations. In *Proc. 8th Ann. ACM/SIAM Symp. Discrete Algs.(SODA97)*, number LCBB-CONF-1997-001, pages 148–157. SIAM Press, 1997.

[9] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer Science & Business Media, 2012.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[11] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[12] G. Gottlob and S. T. Lee. A logical approach to multicut problems. *Information Processing Letters*, 103(4):136–141, 2007.

[13] M. Grohe, K.-i. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 479–488. ACM, 2011.

[14] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[15] E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[16] T. Kociumaka and M. Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556 – 560, 2014.

[17] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, Oct. 2014.

[18] B. Naveh. Jgrapht. `http://www.jgrapht.org`. Last accessed: 08-08-2015.

[19] N. Robertson and P. Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[20] J. Stott, P. Rodgers, J. Martinez-Ovando, and S. Walker. Automatic metro map layout using multicriteria optimization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(1):101–114, Jan 2011.

[21] T. Strijk and M. van Kreveld. Practical extensions of point labeling in the slider model. *GeoInformatica*, 6(2):181–197, 2002.

[22] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 30(2):135–140, 1981.

[23] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry*, 13:21 – 47, 1999.

[24] A. Wolff. General map labelling. `http://i11www.iti.uni-karlsruhe.de/~awolff/map-labeling/general/`. Last accessed: 08-08-2015.

[25] P. Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9(2):99–108, 1972.