University of Bergen

Department of Informatics

Algorithms

Arrangement Problems Parameterized by Neighbourhood Diversity

Student:
Olav Røthe Bakken

Supervisor:
Professor Daniel Lokshtanov
co-Supervisor:
Professor Fedor Fomin

Master Thesis
November 2018

Acknowledgement

I would like to thank Daniel Lokshtanov for giving me such an interesting set of problems to think about and Fedor Fomin for giving me a helping hand in the final stretch. I would also like to thanks my parents for all the support, and my fellow Master students for the wonderful atmosphere in the study hall.

Contents

1 Introduction

We will be investigating the fixed parameter tractability of various arrangement problems parameterized by neighbourhood class, which can be considered as a measure of the number of qualitatively different vertices in a graph. Arrangement problems tasks us with finding an optimal arrangement, i.e ordering of the vertices in a graph with respect to some measure of the quality of the arrangement. The spesific arrangement problems I have been investigating are called BANDWIDTH, CUTWIDTH, IMBALANCE, DISTORTION, and Optimal Linear Arrangement, OLA. We can define these problems as minimizing the following functions, with respect to an ordering $\pi : V \to \{1, 2, ..., n\}$

$$
\begin{aligned}
&BANDWIDTH && f_{bw} = \max_{uv \in E} |\pi(u) - \pi(v)| \\
&CUTWIDTH && f_{cw} = \max_{1 \le i \le n} \sum_{\substack{uv \in E \\ \pi(u) \le i \\ \pi(v) > i}} 1 \\
&IMBALANCE && f_{im} = \sum_{i=1}^{n} |l_\pi(v_i) - r_\pi(v_i)| \\
&DISTORTION && f_{di} = \max \sum_{i=\pi(u)}^{\pi(v)-1} D(v_i, v_i + 1) \\
&OLA && f_{ola} = \sum_{uv \in E} |\pi(u) - \pi(v)|
\end{aligned}
$$

When a problem is fixed parameter tractable, FPT, with respect to a parameter k, formally this means there exists an algorithm solving the problem in time bounded by $f(k) \cdot n^c$, where c is a constant. Intuitively this means the problem is polynomial time solvable for all fixed values of k, and the value of k only affects the constant term multiplier.

The main tool for investigating these problems will be integer linear programming, ILP, and integer quadratic programming, IQP. While neither of these problem-formulations admit polynomial-time algorithms, they are both FPT with respect to the number of variables and the maximum coefficient of each of these variables in the objective function and the constraints which gives us a versatile tool whenever we can give bounded dimension reductions to ILP or IQP.

1.1 Previous work

Arrangement and layout problems on graphs have a wide set of uses as models for things like optimizing parallel computer networks, VLSI design, numerical analysis, computational biology, graph theory, scheduling and archeology.

Some notable examples include BANDWIDTH being equivalent to finding the optimal bandwith of a sparse graph [4] which is important both for the storage and manipulation of sparse matrices, IMBALANCE being useful as a starting point for many algorithms in graph drawing [19, 20, 28, 32, 31], and CUTWIDTH being used as part of a TSP algorithm [17].

BANDWIDTH was first shown to be NP-hard in 1976 [27] using a reduction first from exact 3-SAT to the linear array problem and then to BANDWIDTH. CUTWIDTH in 1977 [14]. A proof for IMBALANCE is given in [1]. Optimal Linear Arrangement in 1974 [13]. DISTORTION is NP-complete on bipartite, co-bipartite and split graphs [15], which of course implies it is NP-complete for general graphs as well.

Many problems have been successfully tackled by considering parameterizations on the treewidth of the input graph, in particular it has been shown that any problem on graphs expressible in the language of Monadic Second Order logic is FPT when parameterized by the treewidth of the input graph [6]. Some problems are not FPT with this parameterization however, in particular BANDWIDTH is NP-complete even when the input graph is a caterpillar with hairlength 3 [26]. For these problems it makes sense to consider parameterizations that enforce more order on the input graph. Previously Fellows, Lokshtanov, Misra, Rosamund and Saurabh have considered the parameterization by the vertex cover number of the input graph, $\text{vc}(G)$, and by using a reduction to integer linear program have shown that BANDWIDTH, IMBALANCE, CUTWIDTH and DISTORTION is FPT when parameterized by $\text{vc}(G)$ [10]. Additionally Lokshtanov has shown that OPTIMAL LINEAR ARRANGEMENT is FPT when parameterized by $\text{vc}(G)$ using a reduction to integer quadratic programmming with bounded dimension and maximum coefficient [25].

The previous results on graphs of bounded neighbourhood diversity includes the original introduction of the parameter [23], where the authors show that problems expressible in MSO1 can be solved on graphs of bounded neighbourhood diversity with a double exponential dependency on the neighbourhood diversity. They also show that Hamiltonian Cycle, Chromatic number and Edge Dominating set is FPT on graphs of bounded neighbourhood diversity. R. Ganian have shown that p-Vertex-Disjoint Paths, Graph Motif and Precoloring Extension problems are FPT with respect to neighbourhood diversity [12]. The proof given for p-Vertex-Disjoint Paths uses integer linear programming with bounded dimension, which we will also rely on for BANDWIDTH and DISTORTION.

We will be considering arrangement problems parameterized by neighbourhood diversity, and have been able to show that BANDWIDTH, IMBALANCE and DISTORTION is FPT when parameterized by neighbourhood

diversity. Additionally we have found structural results for CUTWIDTH similar to the structural results for IMBALANCE. These results extend the results by Lokshtanov et. al [10, 25] for arrangement problems parameterized by vertex cover number.

2 Preliminaries

2.1 Definitions

Some general definitions.

Definition 1. *A graph G = (V,E) is a tuple consisting of a set of points V, and edges* $E \subset V^2$.

Definition 2. *The degree of a vertex v is the number of edges connected to it, denoted* $d(v)$.

Definition 3. *The neighbourhood of a vertex v is the set of vertices connected to v, denoted* $N(s)$. $|N(v)| = d(v)$.

Definition 4. *A walk is an ordered sequence of vertices* $v_1, v_2, \ldots, v_k$, *s.t that* $(v_i - v_{i+1}) \in E, i = \{1, 2, .., k-1\}$. *The length of a walk containing k vertices is* $k - 1$.

Definition 5. *A path is a walk where each vertex appears at most once i.e* $i \neq j \Rightarrow v_i \neq v_j$

Definition 6. *A cycle is a path but the first and last vertex is the same.*

Definition 7. *The distance between two vertices u and v, is the length of the shortest path connecting u to v, denoted* $D(u, v)$

Definition 8. *An arrangement is a permutation* $V \rightarrow \{1, 2, .., |V|\}$ *of the vertices of G.*

Definition 9. *A segment is a sequence of subsequent vertices in an arrangement, from the same neighbourhood class.*

A Turing machine is a theoretical construct designed to be the simplest possible computational model which still has sufficient power to be a legitimate model for actual computational devices. While simpler models exist, they do not have enough power to model real world computers convincingly. A Turing machine consists of a tape with symbols, a read-write head positioned somwhere on the tape, and a finite state machine defining the actions that the tape head will take. Each state in the FST defines the action that the machine will take upon reading a particular symbol from the tape. It will write a symbol to the tape (possibly the same symbol it read), move to a new state (possibly the same state), and move the head either left or right.

A decision problem is the problem of determining whether a particular string belong to a language. A language is defined as a set of strings using

symbols from a particular alphabet, for example $\Sigma = \{0, 1\}$. This can for example be all strings with the same number of 0s and 1s, or all strings which encode yes-instances to VERTEX COVER, where we are given a graph and a number k and asked to decide whether there exists a vertex cover with at most k vertices. If a decision problem can be decided by a Turing machine in a polynomial number of steps, we say the problem belongs to P.

A polynomial time reduction is a reduction from an instance a of a decision problem A to an instance b of a decision problem B, such that b is a yes-instance if and only if a is a yes instance. The reduction must also run in polynomial time. Such a reduction demonstrates that a polynomial algorithm for b would imply a polynomial time algorithm for a, as we can simply reduce a to b in polynomial time, and then solve b instead, also in polynomial time.

2.2 NP

Definition 10. *A decision-problem I is in NP if and only if there exist an algorithm sucessfully deciding I running on a nondeterministic turing machine in polynomial time. Alternatively the decision problem can be supplied with a proof that the answer is yes, and we can verify the proof in polynomial time on a deterministic Turing machine.*

Definition 11. *A decision-problem I is NP-hard if and only if $\forall J \in NP$ there exist a polynomial-time reduction from J to I.*

Definition 12. *A decision problem I is NP-complete if*

1. *$I \in NP$*
2. *I is NP-hard*

While many problems have efficient solutions running in a time polynomially dependent on the size of the input, and are thus contained in the complexity class P. There are also many problems which have efficient solutions, given that we have a "magical" machine which can correctly choose between several different execution paths (This machine obviously does not exist). These problems belong to NP. A long standing open problem in computer science is whether there exists some problem which is part of NP but not in P. Despite mountainous efforts this question remains to be decided, and instead the focus is on so called NP-complete problems which are problems which are at least as hard as every other problem in NP. By at least as hard we mean that there exists a polynomial-time reduction from every

other problem in NP, implying that a polynomial-time algorithm for any NP-complete problem automatically gives polynomial-time algorithms for every other problem in NP. Under the reasonable assumption that P $\neq$ NP these problems do not admit polynomial-time algorithms. (The first problem which was proven to be NP-complete was the boolean satisfiability problem [5]).

Definition 13. *The boolean satisfiability problem, SAT, is a collection of literals* $\{x_1, x_2, ..., x_n\}$*, together with a boolean formula* $\phi(x_1, x_2, ..., x_n)$*, and we are asked to find an assigment of the literals to true or false, such that the boolean formula exists, or to report that no such assignment exists.*

By considering reductions from SAT to other problems several hundreds of problems have subsequently been proven to be NP-complete, suggesting that these problems do not admit polynomial-time algorithms. Notably the problems I am focusing on are NP-complete, that is, BANDWIDTH, CUTWIDTH, IMBALANCE, DISTORTION and OPTIMAL LINEAR ARRANGEMENT. Note that while these problems are optimization problems, we can construct decision problems out of them by simply asking for a solution better than some target value. When we say these problems are NP-complete, we mean that the decision versions fulfill the criteria to be NP-complete.

Since it is unlikely we will be able to find fast algorithms for NP-complete problems it makes sense to look for alternative approaches. The first natural (and quite useful) approach is to give up on finding the optimal solutions and instead look for good enough approximations. Here we would generally like to achieve either constant-factor approximation, where a fast algorithm can guarantee the solution it finds is only a constant factor away from the optimal solution, or polynomial time approximation schemes, where we have an algorithm where we can calibrate the accuracy of our algorithm, giving better accuracy at the cost of a longer runtime.

Definition 14. *A constant-factor approximation algorithm is an algorithm which can find a solution no more than a constant factor away from the optimal solution for that problem. This means* $alg(I) \cdot a \geq opt(I)$ *for maximization problems with constant factor a, and* $alg(I) \leq opt(I) \cdot a$ *for minimization problems.*

Definition 15. *A PTAS is an algorithm scheme producing algorithms which can find an* $(1+\epsilon)$*-approximation in time* $O(f(\epsilon) \cdot n^c)$*, where c is a constant.*

- BANDWIDTH has been shown to be approximable within $O((\log |V|)^{4.5})$ [8], and $O(\sqrt{\frac{|V|}{b}} \cdot \log |V|)$ [2], where b is the bandwidth of the graph.

It is not constant-factor approximable [30] and you cannot obtain an absolute error guarantee of $|V|^{1-\epsilon}$ for $\epsilon > 0$ [22].

- CUTWIDTH is approximable within $O(\log |V| \log \log |V|)$ [7].
- OLA is approximable within $O(\log |V|)$ [29].

2.3 Fixed Parameter Tractability

One fruitful approach to deal with the area of NP-complete problems is parameterized complexity. In this approach we consider additional parameters associated with a problem instance in addition to the size of the instance, and try to build algorithms which exploit this parameterization to achieve better runtime than we can achieve only considering problem size. The main classes of problem we encounter in this approach is FPT and XP, fixed parameter tractable problems, and slicewise polynomial problems. If a problem is in FPT this means we can isolate the exponential portion of the runtime to depend only on our additional parameter, and only polynomialy depend on problem-size. This means if we fix the value of the parameter, and only consider problem instances where this parameter has the value we want, we have in effect a polynomial algorithm. If we have an XP-algorithm on the other hand, the degree of the polynomial dependency on the size is dependent on the additional parameter, meaning that we still get polynomial algorithms by fixing the parameter, but with significantly worse polynomials when k is large, so the runtime will blow up significantly faster. As an example consider the difference between vertex cover and independent set where a relatively naive approach yields a $2^k \cdot n^2$ FPT-algorithm for vertex cover and $k^2 \cdot n^k$ XP-algorithm for independent set parameterized by desired solution size. For a modest graph with 100 vertices and k = 10, this translates to a 10^{12} times slower algorithm for independent set.

Definition 16. *A decision problem is fixed parameter tractable if there exists an algorithm that correctly decides the problem using time bounded by $f(k) * n^c$. The problem class containing all problems that are fixed parameter tractable is called FPT.*

Definition 17. *A decision problem is slicewise polynomial if there exists an algorithm that correctly decides the problem using time bound by $f(k) \cdot n^{g(k)}$. The problem class containing all problems that are slicewise polynomial is called XP.*

Generally speaking there are two types of parameterizations, parameterizations by proposed solution size, and structural parameterizations where

we impose some structure on the input. Neighbourhood diversity is a quite restrictive structural parameterization.

2.4 Integer Linear Programs

A linear program consists of a linear objective function, which should be maximized or minimized, and a set of linear constraints, given as equations and inequalities, which should be satisfied.

Definition 18. *An integer linear program consist of an integer vector c, integer matrix A, and integer vector b, and the objective is to find a vector $x \in \mathbb{Z}^n$ minimizing c, while satisfying all constraints $Ax \leq b$.*

$$\begin{aligned} \min_{x} \quad & c^T x \\ \textit{subject to} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

While general LPs can be solved in (worst case) polynomial time using interior point methods [21], when we restrict variables to integer values we can solve NP-complete problems, f.ex. we can construct an ILP computing MINIMUM VERTEX COVER.

$$\begin{aligned} \min \quad & \sum_{u \in V} x_u \\ \text{s.t} \quad & \forall (u,v) \in E \quad x_u + x_v \geq 1 \\ & x \in \{0,1\} \end{aligned}$$

ILP is therefore NP-complete and we can not expect a polynomial algorithm to solve it.

Lenstra showed that p-variable Integer Linear Programming Feasability is solvable with running time doubly exponential in the number of variables [24], a result wich was later improved by Kannan [18], and Frank and Tardos [11]. Fellows et. al extends this result to the optimization version [10] and gives the following theorem.

Theorem 1 ([10])**.** *p-Opt-ILP can be solved using $O(p^{2.5p+o(p)} \cdot L \cdot \log(MN))$ arithmetic operations and space polynomial in L. Here, L is the number of bits in the input, N is the maximum of the absolute values any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.*

One technical detail. While ILP here is defined using the form $a_i^T x \leq b_i$, we will be using constraints some constraints of the form $a_i^T x \geq b_i$ and $a_j^T x = b_j$. These constraint can be expressed in standard notation as $-a_i^T x \leq -b_i$ and $a_j^T x \leq b_j \wedge a_j^T x \geq b_j$ respectively. The same is true for IQPs.

2.5 Integer Quadratic Programs

Definition 19. *An integer quadratic program is an integer matrix A, symmetric integer matrix Q, and integer vector b, and the objective is to minimize $x^T Q x$ while satisfying the constraints $Ax \leq b$.*

$$\begin{aligned} \min_{x} \quad & xQx^T \\ \textit{subject to} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

We will also be making use of integer quadratic programs, where we allow quadratic terms in the objective function. Lokshtanov has shown that IQP is fixed parameter tractable parameterized by $n + \alpha$ where n is the number of variables, and α is the maximum coefficient in A and Q [25]. The general approach he uses is to show that we can either find an optimal solution in a distance bounded by $n+\alpha$ from an arbitrary (not necessarily integer) feasible solution, or we can branch in a manner which reduces the dimension of the set of feasable solutions by one, and find an optimal solution in one of the branches. The brancing number and branching depth are bounded by $n+\alpha$, implying that the number of nodes in the branching tree is bounded by $n+\alpha$. He gives the following theorem.

Theorem 2 ([25])**.** *There exists an algorithm that given an instance of INTEGER QUADRATIC PROGRAMMING, runs in time $f(n, \alpha)L^{O(1)}$, and outputs a vector $x \in \mathbb{Z}^n$. If the input IQP has a feasible solution then x is feasible, and if the input IQP is not unbounded, then x is an optimal solution.*

2.6 Arrangement Problems

Definition 20. *An arrangement is a bijective mapping $\pi : V \to \{1, 2, ..., n\}$ mapping each vertex in a graph with n vertices to a unique number in the range 1 to n. Intuitively we can consider this an ordering of the vertices.*

In the arrangement problems we will be looking at we are tasked with arranging the vertices of a graph in a linear order, so as to minimize a particular objective function. A simple example of a problem which can formulated

as an arrangement problem, is the problem of finding a topological ordering. Recall that in a topological ordering we have as input a directed graph $G = (V, E)$ and we wish to find an ordering $\pi : V \to \{1, 2, ..., n\}$ of the vertices of the graph such that $uv \in E \Rightarrow \pi(u) < \pi(v)$. This is an example of a polynomial time solvable arrangement problem.

2.7 Parametrization by vertex cover number

Previously Lokshtanov et.al. has showns that the arrangement problems BANDWIDTH, CUTWIDTH, IMBALANCE and DISTORTION parameterized by vertex cover number is FPT using reductions to ILP [10], and that OLA is FPT using a reduction to IQP [25]. A natural question is to consider how this parameterization relates to the parameterization by neighbourhood diversity. Graphs with bounded neighbourhood class can have unbounded vertex cover number, so we can not reduce from problems parameterized by neighbourhood diversity to graphs parameterized by vertex cover number, however the reverse reduction is valid, as a graph with a vertex cover of size k, has neighbourhood diversity at most $2^k + k$.

3 Neighbourhood class

Intuitively, neighbourhood diversity counts the number of qualitatively different vertices in a graph. We say that two or more vertices have the same neighbourhood class if their neighbourhoods are equal. When this is the case the vertices are twins of each other and completely interchangeable. The total number of neighbourhood classes will be referred to as the neighbourhood diversity of the graph.

Definition 21. *We will say that 2 vertices u and v have a neighboorhood relation $u \sim v$ when the following holds: $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

Lemma 1. *The neighbourhood relation is an equivalence relation.*

Proof. For a relation to be an equivalence relation it needs to be reflexive, symmetric and transitive

- $N(u) \setminus \{u\} = N(u) \setminus \{u\}$ clearly holds and the relation is reflexive
- $N(u) \setminus \{v\} = N(v) \setminus \{u\} \Rightarrow N(v) \setminus \{u\} = N(u) \setminus \{v\}$ is also obviously true so the relation is symmetric
- consider three vertices u,v,w st. we have $u \sim v$ and $v \sim w$. First observe that this means that either we have all edges uv, vw and uw, or we have none (otherwise either (u,v) or (v,w) cannot have the same neighbourhood class.We have

$$N(v) \setminus \{u\} = N(u) \setminus \{v\}$$

$$N(w) \setminus \{v\} = N(v) \setminus \{w\}$$

. This implies

$$N(u) \setminus \{u, v, w\} = N(v) \setminus \{u, v, w\}$$

$$N(v) \setminus \{u, v, w\} = N(w) \setminus \{u, v, w\}$$

$$N(u) \setminus \{u, v, w\} = N(u) \setminus \{u, v, w\}$$

. Because we either have all edges uv, vw, uw or none this is equivalent to

$$N(u) \setminus \{w\} = N(w) \setminus \{u\}$$

and the relation is transitive

□

Corollary 1. *Note that the transitive property implies that the vertices in a neighbourhood class are either all connected to each other, or there are no internal edges. This gives two types of classes, which will be referred to as clique classes and independent set classes respectively.*

Definition 22. *If u and v have a neighbourhood relation we will say that they are part of the same neighbourhood class, which I will denote by $C(u)$. We will denote the set of all neighbourhood classes by $\mathcal{C}$ st. $\forall u C(u) \in \mathcal{C}$, $Neighbourhood\ diversity = |\mathcal{C}|$.*

3.1 Neighbourhood Decomposition

Definition 23. *A neighbourhood decomposition $H = ND(G)$ is a weighted graph obtained from G by merging the vertices in each neighbourhood class into a single vertex v, with weight $w(v) = |C(v)|$, and adding selfloops to all the vertices that represent clique-classes.*

Since we can check wether any two vertices is contained in the same neighbourhood class in $|V|$ time, we can relatively easily compute the neighbourhood diversity of a graph in polynomial time [23]. In fact the trivial algorithm runs in time $O(|V|^3)$. In addition it will also be useful to construct a decomposition of the graph. This is necessary if we wish to implement algorithms using neighbourhood diversity. The neighbourhood decomposition works fairly intuitively. We will make a new vertex-weighted graph representing the original graph, where we will have one vertex for each neighbourhood, weighted by the number of vertices in each neighbourhood, and each pair of vertices will have an edge between them if the vertices from their respective neighbourhood classes has edges between them. Vertices that represent clique neighbourhoods will have a selfloop. Note that there may be multiple decompositions of the graph into disjoint cliques and independent sets (consider splitting the vertices of a neighbourhood class into two classes), but the neighbourhood decomposition will be the minimum (and minimal) of these.

Theorem 3 ([23])**.** *We can compute the neighbourhood diversity of a graph in polynomial time.*

Proof. First observe that in order to check wether or not any two vertices are in the same neighbourhood class, we can simply compare their adjacency lists in time $|V|$. The total time to check all pairs of vertices is then $|V|^2 \cdot |V| = |V|^3$. □

I will now give an improved algorithm for computing the neighbourhood decomposition, running in time $O(|V|^2 \cdot \log |V|)$, by the means of polynomial hashing to speed up comparisons.

3.2 Polynomial hash

A hash function is a function which maps a large input-space to a (significantly smaller) output-space, and is can significantly speed up comparisons since we can compare hashes instead of the original object. In the case that the hash is smaller than some fixed number we can perform constant time comparisons, with the tradeoff that there is a (hopefully small) probablity of incorrectly identifying a match. A useful way to hash strings is called polynomial hashes and have a number of interesting properties.

Definition 24. *A polynomial hash is a function h: String → Integer, where x is a number called the hash-multiplier and m is an integer called the modulus.*

$$h(s) = \sum_{i=0}^{len(s)-1} s[i] * x^i \ \textit{modulus}\ m$$

Once we have computed the hash for a string once we can compute the hash for an altered version of the string quickly for a number of different alterations, for our purpose the important one is that we can replace a character anywhere in the string. Assuming we know $h(s_1bs_2)$ we can compute $h(s_1as_2)$.

$$h(s_1as_2) = h(s_1bs_2) + a * x^{|s_2|} - b * x^{|s_2|} \bmod m$$

There are a number of other tricks we can do with with polynomial hashes to speed up comparisons, such as sliding a substring along a longer host string, or fast substring-hashing, allowing us to compute hashes for arbitrary substrings in time $O(\log |s|)$ for each hash, but for our purpose the given trick will suffice.

Note that while comparing hashes of a particular size will generally take time proportional to the number of bits in the hash, for the purpose of comparing objects constant-size hashes will usually suffice, f.ex 64-bit hashes. These can be compared in constant time.

3.3 Neighbourhood decomposition in $O(\log |V| \cdot |V|^2)$

By using polynomial hashing to speed up comparisons the previous algorithm can be improved to $O(\log |V| \cdot |V|^2$. This algorithm will have 3 steps:

1. Compute a hash for the neighbourhood of each vertex
2. Compute the neighbourhood decomposition
3. Check that neighbourhood decomposition is valid

Step 3 is necessary because we are using hashes, which means we have a small probability of hash collisions in every comparison, therefore we need to check that our algorithm has computed the neighbourhood decomposition correctly, otherwise we need to try again with a different hash-multiplier and modulus.

For this algorithm we will use the following polynomial hash-function:

$$h(X \subset V) = \sum_{v_i \in X} x^i \text{ modulus } m$$

By computing the monomials first and then appying the definition we can compute all the hashes we need in time $O(\log m \cdot |V| + |E|)$.

Once we have computed all the hashes and monomials we need we will apply the previous algorithm to compute neighbourhood decomposition but instead of comparing vertices edge by edge we will compare hashes. Our list of hashes refers to $N(u)$ instead of $N(u) \setminus v$ so if v is adjacent to u we will need to remove its contribution to $h(N(u))$. Because we have precomputed the contributions from each vertex, this can be done in time $O(\log m)$.

$$h(N(u) \setminus v_i) = h(N(u)) - x^i \text{ modulus } m$$

Repeat for $N(v) \to N(v) \setminus u$. We can then compare the hashes in time $O(\log m)$ and if they match, we will assume that u and v have the same neighbourhood class. In total we use time $O(\log m \cdot |V|^2)$.

In the last step we will check the neighbourhood decomposition we have computed. Note that hash-comparisons can only produce false positives, not false negatives, so if our neighbourhood decomposition is incorrect, one of the neighbourhood classes will contain an incorrectly placed vertex. We can now use the slower linear compare, since we only need to compare all vertices in a class against a single representative, giving us a total of only $O(|V|)$ comparisons. In total this takes time $O(|V|^2)$.

The probability that step 2 will succesfully compute the neighbourhood decomposition is dependent on the total number of different values the hash can have. The hash can have m different values, so the probablity of a hash collision is $\frac{1}{m}$ for each comparison. If we assume independent probablity (reasonable, and does not affect outcome much) the probability that we have no hash-collisions will be $(1 - \frac{1}{m})^{|V|^2}$. If we pick $m = a \cdot |V|^2$ we have

$$p(\text{no collisions}) = (1 - \frac{1}{a * |V|^2})^{|V|^2} \approx (\frac{1}{e})^{\frac{1}{a}}$$

We can for example pick $a = 100$ which will have a collision-probability of approximately 1%. (Since most of the graphs it makes sense to compute

neighbourhood diversity for will be quite small, this will most of the time fit into an unsigned int (2^{32}), giving effectively constant time comparisons).

Theorem 4. *By picking a hash modulus $m = a \cdot |V|^2$, we can compute the neighbourhood diversity of a graph in time $|V|^2 \cdot \log(a \cdot |V|)$ with error probability $(\frac{1}{e})^{\frac{1}{a}}$. We can also compute the neighbourhood decomposition in the same runtime.*

Note that we can get rid of the possibility of failure by validating every match using a linear scan. While this will make some of the comparisons slower, note that it only does this when we have a significant chance of finding an actual match, and we never need to find more than $|V| - 1$ matches. The runtime will be $O(\log m \cdot |V|^2)$ for true matches, $O(\log m \cdot (1 - \frac{1}{m}) \cdot |V|^2)$ for true non-matches and $O(\log m \cdot \frac{1}{m} \cdot |V|^3)$ for false matches. In the end this will achieve the same asymptotic running time.

3.4 Vertex Cover

Definition 25. *A vertex cover of a graph $G = (V, E)$ is a subset $S \subset V$ such that for every edge $(u, v) \in E$ we have either $u \in S$ or $v \in S$. The vertex cover with minimum cardinality is called the minimum vertex cover.*

As a simple warmup consider MINIMUM VERTEX COVER parameterized by neighbourhood diversity. Observe that for neighbourhood classes that form independent sets, we should either include all the vertices in the cover or none, and for neighbourhood classes that form cliques, we should include either all the vertices or all of the vertices except one. By branching in this manner on each neighbourhood class we obtain a running time of $2^k \cdot n^{O(1)}$ (citation).

Theorem 5. *MINIMUM VERTEX COVER parameterized by neighbourhood diversity can be computed in time $2^k \cdot n^{O(1)}$.*

Proof. The central observation is that we only have a small number of reasonable choices for how many vertices of each type to include, namely, from each class, all of them or none of them, if the class is an independent set, or, all of them or all except one if the class is a clique. To see this, observe that for each edge (except for any self-loops) in the decomposition, either the left or right vertex-class must be fully in the vertex cover in order to cover all of the edges, this means a partially covered class must have only fully covered neighbours, but this means none of the vertices are necessary to cover any of the edges, and we can freely remove the vertices in this class from the vertex

cover, except in the case where the class is a clique, but then it contains all vertices except one anyway (which one is left out is arbitrary).

Therefore, the following algorithm will compute the MINIMUM VERTEX COVER parameterized by neighbourhood class. Check all combinations of including/excluding vertices from each class, and output the smallest valid solution. □

The same general approach will work in many other cases, by including 0, 1, $|C_v| - 1$ or $|C_v|$ vertices from each class. This method yields $4^k \cdot n^{O(1)}$ algorithms for a number of other problems such as:

- MINIMUM DOMINATING SET
- MAXIMUM INDEPENDENT SET
- MINIMUM FEEDBACK VERTEX SET (here we keep all, 2, 1 or 0 vertices from each class)

For arrangement problems the idea will be to attempt to show that there exists an optimal solution where the vertices from each class is placed in a limited number of distinct bags, so that if we can guess the order of bags, we can use an ILP or IQP to figure out how many vertices go in each bag. To that end the following definitions will be useful.

Definition 26. *A maximal contiguous section of vertices from an arrangement, such that all vertices are from the same neighbourhood class is called a zone.*

Definition 27. *The total number of zones of an arrangement is called the zonal dimension of the arrangement. The zonal dimension of a neighbourhood class C_i, is the number of zones the vertices of C_i appear in.*

4 Bandwidth

Definition 28. *Let π be a permution of the vertices of a graph G. Then the bandwidth of the graph $G = (V, E)$ and permutation π is defined as follows.*

$$BW(G, \pi) = \max_{uv \in E} |\pi(u) - \pi(v)| \tag{1}$$

The bandwidth of the graph is defined as the minimum over all permutations

$$BW(G) = \min_{\pi} BW(G, \pi)$$

We let C_i denote the set of vertices from class i, and L_π and R_π denote the set of we get by taking the leftmost and rightmost vertex, with respect to π, in each class.

$$L_\pi = \{u \in V(G) |\ v \in C(u),\ \pi(u) <= \pi(v)\}$$

$$R_\pi = \{u \in V(G) |\ v \in C(u),\ \pi(u) >= \pi(v)\}$$

.

We can observe that in order to compute the bandwidth of π we only need to compare the positions of vertices from $L_\pi \cup R_\pi$.

Lemma 2.

$$\min_{\pi} \max_{uv \in E} |\pi(u) - \pi(v)| = \min_{\pi} \max_{uv \in L_\pi \cup R_\pi} |\pi(u) - \pi(v)|$$

Proof. Suppose that this is not the case and u, v is a vertex-pair with maximum separation with respect to π, but at least one of u or v not in $L_\pi \cup R_\pi$. Assume (without loss of generality) that $u \notin L_\pi \cup R_\pi$. This implies that there are vertices $a, b \in L_\pi \cup R_\pi$ such that $C(u) = C(a) = C(b)$ and $\pi(a) < \pi(u) < \pi(b)$ which implies that either $|\pi(a) - \pi(v)| > |\pi(u) - \pi(v)|$ or $|\pi(b) - \pi(v)| > |\pi(u) - \pi(v)|$. This contradicts u, v being the vertex-pair of maximum distance. □

Lemma 3. *There exists an optimal arrangement π such that $BW(G, \pi) = BW(G)$, where the vertices between two subsequent vertices from $L_\pi \cup R_\pi$ are ordered by neighbourhood class.*

Proof. Suppose ω is an optimal solution, and the above is not satisfied. Then we can find two vertices from $V \setminus L_\pi \cup R_\pi$, which are adjacent in ω, but occur in incorrect order, and swap them. Because this does not change the order of any vertices in $L_\pi \cup R_\pi$ this will not change the optimality of the arrangement. By exhaustively applying this procedure, we will obtain an arrangement with the desired property. □

Since the vertices that are not the leftmost or rightmost in each neighboorhood class can be freely rearranged between vertices which are, we can obtain the optimal solution by guessing the order of the $2k$ vertices which are rightmost and leftmost in each class, and using an integer linear program to calculate how many vertices from each neighboorhood class appear in each segment between two adjacent elements from $L_\pi \cup R_\pi$. We will denote the ordering of the vertices in $L_\pi \cup R_\pi$ by π_N, and the position of vertex $u \in L_\pi \cup R_\pi$ is $\pi_N(u)$. By $E(L_\pi \cup R_\pi)$ we mean the edges connecting vertices from $L_\pi \cup R_\pi$. I will refer to the leftmost and rightmost vertex from each class as delimiter vertices, note that any other vertices from a class can only be placed between its respective delimiters. The positions between two subsequent vertices from $L_\pi \cup R_\pi$ will be referred to as bags.

4.1 Integer Linear Program Formulation

We are now ready to construct an ILP computing the minimum bandwidth, given an ordering of the vertices from $L_\pi \cup R_\pi$. We will have at most $(2 \cdot k - 1) \cdot k$ variables $x_{i,j}$ indicating the number of vertices from neighboorhood class i that will be placed in the bag between vertex u and v from $L_\pi \cup R_\pi$, where $j = \pi_N(u)$ and vertex $j+1 = \pi_N(v)$. Note that vertices from class C_i can only be placed between the delimiters of that class, so if l_i and r_i are the leftmost and rightmost vertices from C_i we have $j < \pi_N(l_i) \vee j \geq \pi_N(r_i) \Rightarrow x_{i,j} = 0$. These variables should be ommited. We will also have a variable c, denoting the maximum separation between two vertices. Clearly the minimum feasible value for c is the bandwidth so our objective function will simply be

$$\text{minimize } c$$

The first constraint will ensure that no pair of connected vertices are further apart then the bandwidth. For each (u, v) in $E(L_\pi \cup R_\pi)$ where $\pi_N(u) < \pi_N(v)$ we have.

$$\pi_N(v) - \pi_N(u) + \sum_{i=1}^{k} \sum_{j=\pi_N(u)}^{\pi_N(v)-1} x_{ij} - c \leq 0$$

We also need to make sure that the variables corresponding to a neighbourhood class sum up to the number of vertices in that class (-2 for the delimiter vertices).

$$\sum_{j=1}^{2*k-1} x_{ij} = |C_i| - 2$$

And of course all the variables must be non-negative and integral. Putting it all together we get the following ILP

$$
\begin{array}{lll}
\text{minimize} & & c \\
\text{such that} & & \\
(u,v) \in E(L \cup R): & \pi_N(v) - \pi_N(u) + \sum_{i=1}^{k} \sum_{j=\pi_N(u)}^{\pi_N(v)-1} x_{ij} \leq & c \qquad \pi_N(u) \leq \pi_N(v) \\
C_i \in \mathcal{C}: & \sum_{j=1}^{2*k-1} x_{ij} = & |C_i| - 2 \\
& x_{i,j} \geq 0 & \\
& x_{i,j} \in \mathbb{Z} &
\end{array}
\tag{2}
$$

Theorem 6. *BANDWIDTH parameterized by neighbourhood diversity k can be computed in time $f(k) \cdot |V|^{O(1)}$.*

Proof. For each permutation of the delimiter vertices, we will construct an ILP computing the optimal bandwidth consistent with that ordering. The ILP constructed when we try a permutation consistent with the actual optimal arrangement will return the bandwidth of G. There are $\frac{(2k)!}{2^k}$ permutations of $L_\pi \cup R_\pi$, and each ILP will run in time $k^{2.5k+o(k)} \cdot L^{O(1)}$ giving a combined runtime of $f(k) \cdot |V|^{O(1)}$. □

4.2 Example

As an example consider the complete bipartite graph with b_1 vertices in the first bipartition and b_2 vertices in the second. This graph has neighbourhood diversity = 2, and if we assume that the optimal order of the delimiters is [p1-left, p2-left, p1-right, p2-right], we will obtain the following ILP (note that this order is not actually optimal, it is merely given as an example).

$$
\begin{aligned}
&\min && c \\
&\text{such that} \\
&&1 + x_{1,1} \leq\ & c \\
&&3 + x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} \leq\ & c \\
&&1 + x_{1,2} + x_{2,2} \leq\ & c \\
&&1 + x_{2,3} \leq\ & c \\
&&x_{1,1} + x_{1,2} =\ & b_1 - 2 \\
&&x_{2,2} + x_{2,3} =\ & b_2 - 2 \\
&&x_{i,j} \geq\ & 0 \\
&&x_{i,j} \in\ & \mathbb{Z}
\end{aligned}
\tag{3}
$$

5 Imbalance

Definition 29. *Let $l_\pi(u)$ denote the number of edges from (u, v) s.t. $\pi(u) > \pi(v)$ and let $r_\pi(u)$ denote the number of edges s.t. $\pi(u) < \pi(v)$ i.e edges to earlier and later vertices respectively.*

Definition 30. *Let π be a permutation of the vertices a graph G. Then the imbalance of the permution is defined as*

$$f_{im}(\pi) = \sum_{i=1}^{n} |l_\pi(v_i) - r_\pi(v_i)|$$

The imbalance of the graph is defined as the minimum over all permutations

$$IM(G) = \min_\pi f_{im}(\pi)$$

In the Imbalance problem we are asked to find an arrangement minimizing IM(π), i.e, compute IM(G). We will show that there exists an optimal arrangement with respect to f_{im}, where each neighbourhood class has been separated into at most two contigous zones. First we will define the *pull* function

$$p_\pi(v) = l_\pi(v_i) - r_\pi(v_i)$$

Notice that $p_\pi(v_i)$ is nondecreasing when evaluated for increasing values of i *over the same Neighbourhood class.*

Lemma 4.

$$\forall u, v \in C(u)\ \pi(u) < \pi(v) \Rightarrow p(u) \leq p(v)$$

Proof. Since u and v comes from the same Neighbourhood class, they have the same incoming edges, the only difference between u and v will be in which direction the various edges pull. We will partition the set of edges into three sets for each vertex, the edges attached to vertices coming before both u and v in the partition u_l and v_l, the edges coming after both u and v, u_r and v_r, and the edges attached to vertices between u and v, u_c and v_c. Since u and v are from the same neighbourhood class we have:

$$u_l = v_l$$

$$u_c = v_c$$

$$u_r = v_r$$

If u and v belong to an independent set class we have:

$$p(u) = l_\pi(u) - r_\pi(u) = u_l - u_c - u_r$$

$$p(v) = l_\pi(v) - r_\pi(v) = v_l + v_c - v_r$$

$$p(u) = u_l - u_c - u_r \leq u_l + u_c - u_r = v_l + v_c - v_r = p(v)$$

$$p(u) \leq p(v)$$

And if u and v belong to a clique class:

$$p(u) = l_\pi(u) - r_\pi(u) - 1 = u_l - u_c - u_r - 1$$

$$p(v) = l_\pi(v) - r_\pi(v) + 1 = v_l + v_c - v_r + 1$$

$$p(u) = u_l - u_c - u_r - 1 \leq u_l + u_c - u_r + 1 = p(v)$$

$$p(u) \leq p(v)$$

□

In order to rearrange the vertices we will also make use of the fact that we can swap a vertex u with its right neighbour without increasing f_{im} provided that $p(u) \leq 0$ after we make the swap, and similarly we can swap with our left neighbour if $p(u) \geq 0$ after the swap.

Lemma 5. *Suppose the arrangement ω can be obtained from σ by swapping a vertex u with its right neighbour v. Then we have*

$$p_\omega(u) \leq 0 \Rightarrow f_{im}(\omega) \leq f_{im}(\sigma)$$

Proof. If u and v are not connected this is obviously true, so assume that $(u, v) \in E$. This implies that

$$p_\sigma(u) = p_\omega(u) - 2$$

Since we have $p_\omega(u) \leq 0$, we have $|p_\omega(u)| = |p_\sigma(u)| - 2$. We also have $|p_\omega(v)| \leq |p_\sigma(v)| + 2$ since that imbalance of vertex v can increase by at most 2. For the remaining vertices in the graph we get $p_\omega(w) = p_\sigma(w)$. The total imbalance after the swap becomes

$$f_{im}(\omega) = \sum_{u \in V} |p_\omega(u)| \leq \sum_{u \in V} |p_\sigma(u)| + 2 - 2 = f_{im}(\sigma)$$

. □

By using the two vertices in each class which have $p(u)$ closest to 0 as guards against moving vertices too far we can safely swap all vertices to the left of the guards to the right, and all vertices to the right of the guards to the left, eventually grouping all the vertices in a class together with their guards. By starting from an arbitrary optimal arrangement, and applying this procedure to each class, we can obtain an optimal arrangement where every class is split into at most two contiguous segments. Furthermore we can do so in a way where $p(u)$ does not flip sign in the middle of a segment (the two segments of a class may be right next to each other). This is not crucial, however it will be easier to formulate a reasonable IQP when we can make this assumption.

Lemma 6. *There exists an optimal arrangement where each Neighbourhood class is partitioned into at most two contiguous segments C_{iL} and C_{iR} such that $u \in C_{iL} \Rightarrow p(u) \leq 0$ and $v \in C_{iR} \Rightarrow p(v) \geq 0$.*

Proof. We will begin by considering an arbitrary optimal arrangement which we will transform into an arrangement having the properties we want. We will do this class by class. First let l_i be the last vertex from some class C_i where $p(l_i) \leq 0$ and r_i be the first where $p(r_i) \geq 0$, notice that l might be equal to r_i and only one of l_i or r_i has to exist.

By lemma 4 all vertices $u \in C(l_i)$ to the left of l_i has $p(u) \leq 0$, and if not adjacent to l_i, it will continue to have $p_(u) \leq 0$ if we swap u with its right neighbour. Therefore, by lemma 5 we can freely swap all the vertices from $C(l_i)$ to the left of l_i with their right neighbour successively until all vertices in $C(l_i)$ to the left of l_i are arranged next to one another immediately to the left of l_i. Similarly we can move all vertices to the right of r_i, so that they end up immediately to the right of r_i.

By applying this procedure to each neighbourhood class $C \in \mathcal{C}$ we will obtain a solution where every class is split into at most 2 contiguous segments, but we still have some work to do to make sure the sign of $p(u)$ does not change inside any of the segments. To fix this observe that if $p(u)$ flips somewhere inside a segment, this means we can group the class even tighter, into a single segment. We can now simple decide where one segment ends and the next begins in order to have all our desired properties. □

5.1 Integer Quadratic Programming Formulation

With these structural results in place we are ready to formulate an FPT algorithm for IMBALANCE parameterized by neighbourhood diversity. For each possible ordering of the segments we will construct an IQP computing the minimum imbalance given the ordering. The IQP will have 2k variables

$x_{1,L} \dots x_{k,L}$ and $x_{1,R} \dots x_{k,R}$ denoting the number of vertices in each of the segments. $\pi_N(x_{i,j})$ denotes the ordering of the segments associated with each variable.

First, we will construct the ojective function, this will be the sum over all segments of [SIZE OF SEGMENT · AVERAGE IMBALANCE], for the left and right segment of each class this looks like

$$\left(\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} \right) \cdot x_{i,L}$$

$$\left(\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} \right) \cdot x_{i,R}$$

Observe here that any internal edges in segment $x_{i,L}$ or $x_{i,R}$ will increase the imbalance of one vertex and decrease the imbalance of another, and so we can treat clique classes the same as independent set classes. The complete objective function becomes

$$\text{OBJ} = \sum_{i=1}^{k} \left(\left(\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} \right) \cdot x_{i,L} + \left(\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} \right) \cdot x_{i,R} \right)$$

For this objective function to be correct we also need to make sure that the sign of the pull function is correct for all the vertices in a segment. This needs to be handled slightly differently for clique classes and independent set classes. For independent set classes the pull for each vertex in the segment is equal to the average pull for the class so we can use the same expressions we have already computed

$$\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} \geq 0$$

$$\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} \geq 0$$

When we have a clique set class we need to account for the internal edges in

the segment as well

$$\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} - x_{i,L} + 1 \geq 0$$

$$\sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} - x_{i,R} + 1 \geq 0$$

Adding the obvious constraint that variables should be nonnegative, and that $x_{i,L} + x_{i,R} = |C_i|$ gives us the following complete IQP.

$$\begin{aligned}
&\text{minimize} && \text{OBJ} \\
&\text{such that} \\
&\qquad \text{for clicques} && \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} - x_{i,L} + 1 \geq 0 \\
&&& \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} - x_{i,R} + 1 \geq 0 \\
&\qquad \text{for independent sets} && \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})>\pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})<\pi_N(x_{i,L})}} x_{u,t} \geq 0 \\
&&& \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,s})<\pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i)\in E \\ \pi_N(x_{u,t})>\pi_N(x_{i,R})}} x_{u,t} \geq 0 \\
&x_{i,L} + x_{i,R} = |C_i| \\
&x_{i,j} \geq 0 \\
&x_{i,j} \in \mathbb{Z}
\end{aligned} \tag{4}$$

Note that this can be preprocessed to get rid of either $x_{i,L}$ or $x_{i,R}$, halving the number of distinct variables.

Theorem 7. *IMBALANCE parameterized by neighbourhood class k can be computed in time $f(k) \cdot |V|^{O(1)}$*

Proof. In order to compute the imbalance of G we try each permutation of the $2k$ segments of G. For each permutation we will construct an IQP finding the optimal arrangement agreeing with this permutation. When we pick the correct permutation, this IQP will return the imbalance of G. We make $\frac{(2k)!}{2^k}$ guesses and construct an IQP with bounded dimension and max coefficient. From theorem 2 we can evaluate the IQPs in time $f(k) \cdot |V|^(O(1)$. Total runtime is bounded by $f(k) \cdot |V|^{O(1)}$. □

5.2 Example

As an example consider again the complete bipartite graph with b_1 vertices in one partition and b_2 vertices in the other, and assume we have guessed to optimal order of segments to be [1,2,1,2]. We then get the following IQP.

$$
\begin{aligned}
\text{minimize} \quad & x_{1L} \cdot (x_{2L} + x_{2R}) + x_{2L} \cdot (x_{1R} - x_{1L}) + x_{1R} \cdot (x_{2L} - x_{2R}) + x_{2R} \cdot (x_{1L} + x_{1R}) \\
\text{such that} \quad & x_{2L} + x_{2R} \geq 0 \\
& x_{1L} + x_{1R} \geq 0 \\
& x_{2L} - x_{2R} \geq 0 \\
& x_{1R} - x_{1L} \geq 0 \\
& x_{1L} + x_{1R} = |C_i| \\
& x_{2L} + x_{2R} = |C_i| \\
& x_{i,s} \geq 0 \\
& x_{i,s} \in \mathbb{Z}
\end{aligned}
$$

6 Distortion

Definition 31. *A mapping $f : V \to \mathbb{R}$ has contraction c_f and expansion e_f if for every pair of vertices $u, v \in V$, the following holds: $|f(u) - f(v)| \cdot \geq D(u, v)$ and $|f(u) - f(v)| \geq e_f \cdot D(u, v)$*

In the (linear) DISTORTION problem we want to find a mapping from the vertices to the real line so that the contraction $c_f \geq 1$, and the expansion e_f is minimized. As has been observed by several authors before [9, 16], this is equivalent to finding an arrangement $\pi : V \to \{1, 2, ..., n\}$ which minimizes the following function:

$$f_{di}(\pi) = \max_{(l,r)\in E} \sum_{i=\pi(l)}^{\pi(r)-1} D(v_i, v_{i+1})$$

If $L \cup R$ is the set of leftmost and rightmost vertices within each neighbourhood class, this is equivalent to

$$f_{di}(\pi) = \max_{(l,r)\in E(L\cup R)} \sum_{i=\pi(l)}^{\pi(r)-1} D(v_i, v_{i+1})$$

In order to find an FPT-algorithm for this problem we will use the following intuition.

1. When computing the distortion for a particular arrangement we are only interested in the vertices which are the furthest apart within each pair of neighbourhood classes, similar to BANDWIDTH.

2. The order of vertices between delimiter vertices can be locally optimized, that is, if we change the order of vertices between adjacent delimiters a and b, this will not change the optimal ordering between adjacent delimiters c and d.

Together these observations gives the following strategy. First we will guess the order of the delimiter vertices, and then we will use an ILP to fill in the remaining vertices. Unlike in BANDWIDTH the order of vertices in each bag does matter, so we will have to be a little bit more clever. Instead of considering the vertices we add to each bag, we will instead consider the ordered and adjacent pairs we add (I will abuse notation and refer to these ordered pairs as edges, but they need not be edges in the original graph). Because the total length of all the edges we add to a bag is simply the sum of

the lengths of each edge, the order of edges does not matter, and we simply need to check that a particular (multi)-set of edges can actually be traversed. Here we can rely on a famous theorem by Euler, stating that a (multi)-graph has a walk traversing each edge exactly once and starting and ending in the same vertex, if and only if the degree of each vertex is an even number, and the graph is connected. When the graph is directed it has a walk traversing each edge exactly once if and only if the in-degree is equal to the out-degree for all vertices and the graph is connected. Connectivity will be enforced by guessing which edges we will use to traverse each bag, and correct degree for each vertex will be enforced by linear equations. Note that since we are interested in $s-t$ walks, not cycles, the first and last vertex will have differing in-/out-degree unless $C_s = C_t$.

Lemma 7. *The order of vertices within a 'bag' can be rearranged so long as the rearranged order corresponds to the same edges.*

Proof. Suppose we have an 2 different arrangements, π and ω, of the vertices in a 'bag' which corresponds to the same edge-(multi-)set. Assume that the left and and right delimiters of the 'bag' is l and r respectively. Let $v_{\pi(i)}$ be the vertex at position i with respect to π, and let $v_{\omega(i)}$ be the vertex at position i with respect to ω. E_l denotes the edge-(multi)set associated with the bag after the delimiter at position l. Then we have

$$\sum_{i=l}^{r-1} D(v_{\pi(i)}, v_{\pi(i+1)}) = \sum_{(u,v)\in E_l} D(u,v) = \sum_{i=l}^{r-1} d(v_{\omega(i)}, v_{\omega(i+1)})$$

□

Because all orderings of the edges have the same length, we only care about how many of each type of edge we include between each pair of subsequent vertices from $L \cup R$, henceforth referred to as a bag.

6.1 Integer Linear Program formulation

We are now ready to formulate an ILP computing the minimum distortion given that we have guessed the order of the delimiter vertices, as well as which edges to include in each bag. We will also guess the first and last edge in some of the bags since these edges might go to and from delimiter vertices which we can only put at the beginning and/or end of the bag. These edges will be included only once. We will have $(2k-1)\cdot k^2$ variables $x_{i,j,l}$ denoting the number of instances of edge (i,j) in bag l. Variables that must have value 0 can be omitted. This includes $x_{i,j,l}$ when bag l is not positioned

between the delimiters for C_i and C_j, except in the case where the left or right delimiter of a bag is excluded from the bag, in which case we guess which edge to start and/or end on. It will also include $x_{i,j,l}$ if we did not choose to include edges of type (i, j) in bag l. We will also have a variable c, denoting the maximum separation of any connected vertices from $L \cup R$. Clearly the distortion $\mathrm{DI}(G) = \min c$ is a suitable objective function.

For every connected pair of vertices u and v from $L \cup R$ we need a constraint enforcing that their separation is less than the maximum separation. This simply amounts to adding up the lengths of all edges between them

$$\sum_{l=\pi_N(u)}^{\pi_N(v)-1} \sum_{i=1}^{k} \sum_{j=1}^{k} D(i,j) \cdot x_{i,j,l} - c \leq 0$$

Next we need constraints ensuring that the edges in each bag can be traversed, i.e. we have an euler-path from the start vertex s to the end vertex t. To check this we need to check that the in-degree is equal to the out-degree for every class, except for the C_s and C_t, which need one extra out-neighbour and one extra in-neighbour, respectively, unless $C_s = C_t$. Connectivity will be enforced by the guessing step. For each neighbourhood class u that is different from C_s and C_t we get.

$$\sum_{i=1}^{k} x_{i,u,l} - \sum_{j=1}^{k} x_{u,j,k} = 0$$

For s and t we get

$$\sum_{i=1}^{k} x_{i,t,l} - \sum_{j=1}^{k} x_{t,j,k} = 1$$

$$\sum_{i=1}^{k} x_{i,s,l} - \sum_{j=1}^{k} x_{s,j,k} = -1$$

If $C_s = C_t$

$$\sum_{i=1}^{k} x_{i,s,l} - \sum_{j=1}^{k} x_{s,j,k} = 0$$

We also need to make sure we use exactly the right amount of vertices from each bag. Note here that apart from the first and last vertex of the arrangement, every vertex is counted twice, once when going to it, and once when going from it. The classes containing the first and last vertex s and t, will

be one short from every edge counting twice if the classes are different, or 2 short if they are the same. So if $C_s \neq C_t$ we have

$$\sum_{l=1}^{2k}\sum_{i=1}^{k}(x_{i,u,l} + x_{u,i,l}) = 2 \cdot |C_u|$$

$$\sum_{l=1}^{2k}\sum_{i=1}^{k}(x_{i,t,l} + x_{t,i,l}) = 2 \cdot |C_t| - 1$$

$$\sum_{l=1}^{2k}\sum_{i=1}^{k}(x_{i,s,l} + x_{s,i,l}) = 2 \cdot |C_s| - 1$$

If $C_t = C_s$ we have

$$\sum_{l=1}^{2k}\sum_{i=1}^{k}(x_{i,u,l} + x_{u,i,l}) = 2 \cdot |C_u|$$

$$\sum_{l=1}^{2k}\sum_{i=1}^{k}(x_{i,s,l} + x_{s,i,l}) = 2 \cdot |C_t| - 2$$

All the edges we have guessed to be included in each bag should be included at least once

$$x_{i,j,l} \geq 1 \text{ if included}$$

$$x_{i,j,l} = 0 \text{ if not included}$$

Finally, if the right or left delimiter of a bag cannot be included in the bag we need to enforce exactly one ocurrence of the first and/or last edge we guessed respectively

$$x_{s,j,l} = 1 \quad \text{when we guess the first edge in bag l is (s,j)}$$

$$x_{i,t,l} = 1 \quad \text{when we guess the last edge in bag l is (i,t)}$$

Putting it all together gives us the following ILP

$$
\begin{array}{lll}
\text{minimize} & & c \\
& \text{such that} & \\
& (u,v) \in E(L \cup R) & \sum_{l=\pi_N(u)}^{\pi_N(v)-1} \sum_{i=1}^{k} \sum_{j=1}^{k} D(i,j) \cdot x_{i,j,l} - c \leq 0 \\
& \text{To enforce Euler path:} & \\
& \text{For each bag l and class u} & \sum_{i=1}^{k} x_{i,u,l} - \sum_{j=1}^{k} x_{u,j,k} = 0 \\
& \text{If } C(s) \neq C(t)\text{:} & \sum_{i=1}^{k} x_{i,s,l} - \sum_{j=1}^{k} x_{s,j,k} = -1 \\
& & \sum_{i=1}^{k} x_{i,t,l} - \sum_{j=1}^{k} x_{t,j,k} = 1 \\
& \text{If C(s) = C(t):} & \sum_{i=1}^{k} x_{i,s,l} - \sum_{j=1}^{k} x_{s,j,k} = 0 \\
& \text{Number of edges matches size of class} & \\
& \text{for each class C(u):} & \sum_{l=1}^{2k} \sum_{i=1}^{k} \left(x_{i,u,l} + x_{u,i,l}\right) = 2 \cdot |C_u| \\
& \text{subtract 1 for class of first and last vertex} & \sum_{l=1}^{2k} \sum_{i=1}^{k} \left(x_{i,u,l} + x_{u,i,l}\right) = 2 \cdot |C_u| - 1 \\
& \text{subtract 2 if class is both first and last} & \sum_{l=1}^{2k} \sum_{i=1}^{k} \left(x_{i,u,l} + x_{u,i,l}\right) = 2 \cdot |C_u| - 2 \\
& \text{edges included} & x_{i,j,l} \geq 1 \\
& \text{edges included once} & x_{i,j,l} = 1 \\
& & x_{i,j,l} \in \mathbb{Z}
\end{array}
\tag{5}
$$

Theorem 8. *DISTORTION parameterized by neighbourhood diversity can be solved in time $f(k) \cdot |V|^{O(1)}$ and polynomial space.*

Proof. In order to compute the distortion of G, we will try each guess of

- The order of delimiters, $\frac{(2k)!}{2^{2k}}$ guesses
- Edges used in each bag, 2^{2k^3} guesses
- First and last edge in each bag, k^{2k} guesses

For each of these guesses we will construct the an ILP computing the minimum distortion consistent with that guess. For the correct guess this will return the distortion of the graph. The total number of guesses is bounded by a function of k, and the total number of variables for each ILP is at most $2k^3$, giving a combined runtime of $f(k) \cdot |V|^{O(1)}$. □

6.2 Example

We will continue to use the complete bipartite graph example, where one partition has b_1 vertices and the other has b_2 vertices. We will guess that the order of delimiters is [1,2,1,2], and that we use edges $(1,1)$ for the first gap, $(1,2)$ and $(2,1)$ for the second, and $(2,2)$ for the last. The last edge in bag 1 has to be $(1,2)$ and the first edge in bag 3 has to be $(1,2)$. We obtain the following ILP:

minimize	c
such that	
paths shorter than c:	$2x_{1,1,1} + x_{1,2,1} - c \leq 0$
	$x_{1,2,2} + x_{2,1,2} - c \leq 0$
	$x_{1,2,3} + 2x_{2,2,3} - c \leq 0$
	$2x_{1,1,1} + x_{1,2,1} + x_{1,2,2} + x_{2,1,2} + x_{1,2,3} + 2x_{2,2,3} - c \leq 0$
paths are euler paths	
class 1 bag 1	$x_{1,1,1} - x_{1,1,1} - x_{1,2,1} = -1$
class 2 bag 1	$x_{1,2,1} = 1$
class 1 bag 2	$x_{2,1,2} - x_{1,2,2} = 1$
class 2 bag 2	$x_{1,2,2} - x_{2,1,2} = -1$
class 1 bag 3	$x_{1,2,3} = 1$
class 2 bag 3	$x_{2,2,3} - x_{2,2,3} - x_{1,2,3} = -1$
classes filled	$x_{1,1,1} + x_{1,2,2} + x_{2,1,2} = 2 \cdot b_1 - 1$
	$x_{1,2,2} + x_{2,1,2} + x_{2,2,2} = 2 \cdot b_2 - 1$
edges used	$x_{i,j,l} \geq 1$
	$x_{1,2,1} = 1$
	$x_{1,2,3} = 1$
	$x_{i,j,l} \in \mathbb{Z}$

7 Cutwidth

Definition 32. *Let π be a permutation of the vertices in a graph G. Then, the Cutwidth of the permutation is*

$$f_{cw}(\pi) = \max_{i=1}^{n} \sum_{\substack{uv \in E \\ \pi(u) < i \\ \pi(v) \geq i}} 1$$

The cutwidth of the graph is

$$CW(G) = \min_{\pi} f_{cw}(\pi)$$

Most of the structural results that are valid for IMBALANCE also hold for CUTWIDTH, and so we can obtain the same simply structure for optimal arrangements. Unfortunately the approach breaks down when you attempt to express the problem as an integer program. Whereas the formulation of IMBALANCE yields an IQP, the same approach to CUTWIDTH results in a set of quadratic inequalities and we cannot rely on theorems about IQPs. We can however establish the following partial results, which implies a simple bruteforce XP-algorithm.

Lemma 8. *Suppose the arrangement ω can be obtained from σ by swapping a vertex u with its right neighbour v. Then we have*

$$p_\omega(u) \leq 0 \Rightarrow f_{cw}(\omega) \leq f_{cw}(\sigma)$$

Proof. Let l_u, r_u, l_v, r_v be the the number of left and right neighbours of u and v respectively (not counting an edge between u and v. Let us first consider the case when u and v are not connected. Note that when swapping two vertices we only change the cut between them. Before we swap this has value $r_u + l_v + a$, where a is the number of edges going over both u and v. After the swap the cut has value $l_u + r_v + a$. However the cut immediately to the right of both vertices has value $r_u + r_v + a \geq l_u + r_v + a$, and the middle cut after the swap cannot be the maximum cut in the arrangement. Since no other cut has changed, this implies the maximum cut cannot have increased, and the swap is safe. If u and v are connected the middle cut goes from $r_u + l_v + a + 1 \rightarrow l_u + r_v + a + 1$. Since $p_\omega(u) \leq 0$ after the swap, we have $p_\omega(u) \leq -2$ before the swap, and $l_u \leq r_u - 2$. Again we get $r_u + r_v + a \geq l_u + r_v + a + 2 \geq l_u + r_v + a + 1$ and the cut between u and v cannot be the maximum cut after the swap. □

By using the same strategy which we used for IMBALANCE, we can group the vertices from each neighbourhood class into at most two zones for each class.

Lemma 9. *There exist an optimal arrangement of G where each class is separated into at most 2 zones.*

For the sake of completeness, I have included the integer program encoding CUTWIDTH in the appendix, note however that it uses quadratic constraints, and cannot be computed using IQP solvers.

By brute-force checking all orderings of the segments and all 2-partitions of each neighbourhood class we can find the optimal CUTWIDTH in time $(2k)!n^k \cdot P(n)$.

Corollary 2. *We can find an optimal arrangement in time* $O(f(k) \cdot n^{g(k)})$.

7.1 Faster XP algorithm using dynamic programming

While the bruteforce algorithm does the job in terms of showing that CUTWIDTH is in XP, it is not a particularly elegant algorithm, and we can do better. By using dynamic programming to check partitions in a more sensible way, we can get rid of the need to order segments, giving us a more practical algorithm with runtime $O(n^k \cdot n^2)$. To achieve this I will demonstrate an $O(2^n)$ algorithm solving CUTWIDTH in the general case [3], which reduces to $O(n^{k+2})$ when parameterized by neighbourhood diversity.

In order to achieve this, observe that if I guess the first s vertices in the arrangement, the remaining $|V| - s$ vertices can be arranged independently of the ordering of the first s vertices. This suggests the following strategy for computing the ccutwidth. First guess the first vertex in the arrangement, and for each guess, call an auxillary function recursively, which will find the optimal arrangement of the remaining vertices. The cutwidth of a subset $X \subset V$, $CW_{aux}(X)$ will either be the minimal cutwidth of $X - u$, $CW_{aux}(X - u)$, or it will be the value of the last cut, which ever is greater. This gives the following recursively defined function, where $CW_{aux}(\emptyset) = 0$.

$$CW_{aux}(X) = \min_{x \in X} \max(CW_{aux}(X - x), \sum_{\substack{uv \in E \\ u \in X \\ v \notin X}} 1)$$

Note that $CW_{aux}(G) = CW(G)$. By utilizing memoization we only need to compute the value of $CW_{aux}(X)$ once for each subset $X \subset V$. Each subproblem can be computed in time $O(n^2)$, and we have 2^n subproblems, giving a total runtime of $O(2^n \cdot n^2)$.

When G has neighbourhood diversity k, the number of subsets can be reduced to n^k, giving a total runtime of $O(n^{k+2})$ instead.

Theorem 9. *CUTWIDTH parameterized by neighbourhood diversity can be computed in time* $O(n^{k+2})$.

8 Optimal linear arrangement

Definition 33. *In the Optimal Linear Arrangement problem we are asked to find the arrangement minimizing the total stretch of all edges.*

$$OLA(G) = \min_{\pi} \sum_{uv \in E} |\pi(u) - \pi(v)|$$

Unfortunately I have been unable to find structural results which enable me to prove that OLA is FPT. I have been able to show it is in XP using dynamic programming, as well as some reasons to believe that integer programming is unlikely to be a successfull approach for solving this problem.

8.1 OLA is in XP

To show that OLA parameterized by neighbourhood diversity is in XP we will go through the route of a DP-algorithm for OLA not parameterized running in time $O(2^n)$ [3], which reduces to a $O(n^k)$ when parameterized, similar to the algorithm for CUTWIDTH. The central idea is that if we can guess which subset comes first in an arrangement, we can independently order the first and second halves of the arrangement. Using the following alternative computation of OLA(G) lends itself to a useful axillary function.

$$OLA(G) = \min_{\pi} \sum_{i=1}^{|V|} \sum_{\substack{uv \in E \\ \pi(u)<i \\ \pi(v)\geq i}} 1$$

If we only consider the part of the edges that is counted "above" a particular subset we obtain the following auxillary function.

Definition 34. *For a graph G = (V,E) and a subset $X \subset V$*

$$OLA_{aux}(X) = \min_{\pi | \pi(x \in X) \leq |X|} \sum_{i=1}^{|X|} \sum_{\substack{uv \in E \\ \pi(u)<i \\ \pi(v)\geq i}} 1$$

Note that $OLA_{aux}(V) = OLA(G)$.

As was the case with CUTWIDTH, we can compute $OLA_{aux}(X)$ recursively by first finding the optimal vertex $v \in X$ to place first, and then adding

the contribution from the edges leaving X.

$$OLA_{aux}(X) = \min_{x \in X} OLA_{aux}(X - x) + \sum_{\substack{uv \in E \\ x \in X \\ v \notin X}} 1$$

The base case is when $X = \emptyset$ and clearly we have $OLA_{aux}(\emptyset) = 0$. Computing OLA_{aux} for all subsets of V takes time $O(2^n \cdot n^2)$.

Lemma 10. *OLA can be computed in time* $O(2^n \cdot n^2)$.

When we solve OLA parameterized by neighbourhood diversity we can move from sets to multisets. Since there are at most n vertices in each neighbourhood class, we have at most n^k multisets to consider, which automatically improves the previous result to $O(n^{k+2})$.

Theorem 10. *We can compute OLA parameterized by neighbourhood diversity in time* $O(n^{k+2})$.

9 Conclusion and Open Problems

We have been able to show that BANDWIDTH, IMBALANCE, and DISTORTION parameterized by neighbourhood diversity is fixed parameter tractable using reductions to ILP and IQP with bounded dimension and bounded coefficients. We also show that OPTIMAL LINEAR ARRANGEMENT and CUTWIDTH is slicewise polynomial when parameterized by neighbourhood diversity.

CUTWIDTH can be reduced to a very simple structure where each neighbourhood class is separated into at most 2 zones, making it very likely that CUTWIDTH is in fact FPT. One possible approach might be to give a deeper analysis of the integer program formulation of CUTWIDTH parameterized by neighbourhood diversity. Regardless it remains to be determined wether CUTWIDTH parameterized by neighbourhood diversity is FPT

Discussing the hardness of OLA I want to consider two points. First, we can prove that OLA parameterized by neighbourhood diversity has unbounded zonal dimension, which does not in and of itself imply that OLA cannot be solved using ILP or IQP, after all DISTORTION works by letting variables represent edges instead of vertices. However it does show that a structure similar to the one used for OLA parameterized by vertex cover [25] is not possible. More notable is the fact that an integer program appears to require a cubic objective function in order to encode OLA. This would certainly be a requirement if we had bounded zonal dimension, and it should also be the case for edge-like encodings similar to what we did for DISTORTION.

Theorem 11. *The minimal zonal dimension of an optimal arrangement is unbounded with respect to k*

Proof. Consider the complete bipartite graph with exactly n vertices in each partition. This graph has neighbourhood diversity 2, yet zonal dimension dependent on n. To see that this graph has unbounded zonal dimension we will consider the optimal arrangement 2 vertices at a time. First observe that the first 2 vertices must be from different partitions, otherwise we could move the first occurance of the second class closer to the beginning, obtaining a strictly better arrangement in the process. However, when the first two vertices are in different partitions, the contributions from their edges to the rest of the arrangement is completely symmetrical and we can ignore them from the remainder of the consideration. This leaves us with finding an optimal arrangement for a bipartite graph with $n - 1$ vertices in each partition.

When all the vertices have been placed in this manner observe that we change class at least n times and so we have zonal dimension at least $n+1$. Since n is not dependent on k the statement is proven. □

For the second reason assume that there could be a simplification of OLA parameterized by neighbourhood diversity which yields a structure with bounded zonal dimension z. This could be encoded by an integer program with at most $O(z)$ variables. By necessity some variables needs to refer to several vertices from the same class, and the total length of all edges associated with a pair of variables will be something like [VARIABLE1] · [VARIABLE2] · [DISTANCE BETWEEN VARIABLES], each of these contributions are linear so the combination will be cubic. In my opinion this strongly suggests that an integer program for solving OLA parameterized by neighbourhood class will be cubic, and that you cannot rely on previous results about ILPs or IQPs.

Nevertheless, it remains to be determined whether OLA parameterized by neighbourhood diversity is FPT.

References

[1] Therese Biedl, Timothy Chan, Yashar Ganjali, Mohammad Taghi Hajiaghayi, and David R Wood. Balanced vertex-orderings of graphs. *Discrete Applied Mathematics*, 148(1):27–48, 2005.

[2] Avrim Blum, Goran Konjevod, R Ravi, and Santosh Vempala. Semidefinite relaxations for minimum bandwidth and other vertex-ordering problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 100–105. ACM, 1998.

[3] Hans L Bodlaender, Fedor V Fomin, Arie MCA Koster, Dieter Kratsch, and Dimitrios M Thilikos. A note on exact algorithms for vertex ordering problems on graphs. *Theory of Computing Systems*, 50(3):420–432, 2012.

[4] Phyllis Z Chinn, Jarmila Chvátalová, Alexander K Dewdney, and Norman E Gibbs. The bandwidth problem for graphs and matricesa survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[5] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

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

[7] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM (JACM)*, 47(4):585–616, 2000.

[8] Uriel Feige and Robert Krauthgamer. Improved performance guarantees for bandwidth minimization heuristics. 1998.

[9] Michael Fellows, Fedor Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances A Rosamond, and Saket Saurabh. Parameterized low-distortion embeddings-graph metrics into lines and trees. *arXiv preprint arXiv:0804.3028*, 2008.

[10] Michael R Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation*, pages 294–305. Springer, 2008.

[11] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1): 49–65, 1987.

[12] Robert Ganian. Using neighborhood diversity to solve hard problems. *arXiv preprint arXiv:1201.3091*, 2012.

[13] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[14] Fanica Gavril. Some np-complete problems on graphs. In *Proc. Conf. on Inform. Sci. and Systems, 1977*, pages 91–95, 1977.

[15] Pinar Heggernes and Daniel Meister. Hardness and approximation of minimum distortion embeddings. *Inf. Process. Lett.*, 110(8-9):312–316, 2010.

[16] Pinar Heggernes, Daniel Meister, and Andrzej Proskurowski. Minimum distortion embeddings into a path of bipartite permutation and threshold graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 331–342. Springer, 2008.

[17] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[18] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[19] Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[20] Goos Kant and Xin He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1-2):175–193, 1997.

[21] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[22] Marek Karpinski, Juergen Wirtgen, et al. On approximation hardness of the bandwidth problem. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 4. Citeseer, 1997.

[23] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

[24] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

[25] Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients. *arXiv preprint arXiv:1511.00310*, 2015.

[26] Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 7(4):505–512, 1986.

[27] Ch H Papadimitriou. The np-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[28] Achilleas Papakostas and Ioannis G Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry*, 9(1-2):83–110, 1998.

[29] Satish Rao and Andréa W Richa. New approximation techniques for some ordering problems. In *SODA*, volume 98, pages 211–219, 1998.

[30] Walter Unger. The complexity of the approximation of the bandwidth problem. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 82–91. IEEE, 1998.

[31] David R Wood. Minimising the number of bends and volume in three-dimensional orthogonal graph drawings with a diagonal vertex layout. 2000.

[32] David R Wood. Optimal three-dimensional orthogonal graph drawing in the general position model. *Theoretical Computer Science*, 299(1-3): 151–178, 2003.

10 Appendix

I will here give an integer program computing the cutwidth, given that we have guessed the order of segments C_{iL} and C_{iR}. The construction is similar to the construction for imbalance, but we will now have an additional variable c denoting the maximum allowed cut. The objective function is

$$\text{minimize } c$$

Because we are constructing a solution where $p(u)$ does not change signs inside segments, the maximum cut will always appear between segments. For each cut between segments i and $i+1$ we get constraints

$$\sum_{\substack{uv \in E(L \cup R) \\ \pi(x_{u,s}) \leq i \\ \pi(x_{v,t}) > i}} x_{u,s} \cdot x_{v,t} - c \leq 0$$

The remaining constraints are the same as for imbalance. The full integer program becomes

$$\begin{aligned}
&\text{minimize} && \text{OBJ} \\
&\text{such that} \\
&\quad \text{for each cut between segment } i \text{ and } i+1 && \sum_{\substack{uv \in E(L \cup R) \\ \pi(x_{u,s}) \leq i \\ \pi(x_{v,t}) > i}} x_{u,s} \cdot x_{v,t} - c \leq 0 \\
&\quad \text{for clicques} && \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,s}) > \pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,t}) < \pi_N(x_{i,L})}} x_{u,t} - x_{i,L} + 1 \geq 0 \\
&&& \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,s}) < \pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,t}) > \pi_N(x_{i,R})}} x_{u,t} - x_{i,R} + 1 \geq 0 \\
&\quad \text{for independent sets} && \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,s}) > \pi_N(x_{i,L})}} x_{u,s} - \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,t}) < \pi_N(x_{i,L})}} x_{u,t} \geq 0 \\
&&& \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,s}) < \pi_N(x_{i,R})}} x_{u,s} - \sum_{\substack{(u,i) \in E \\ \pi_N(x_{u,t}) > \pi_N(x_{i,R})}} x_{u,t} \geq 0 \\
&&& x_{i,L} + x_{i,R} = |C_i| \\
&&& x_{i,j} \geq 0 \\
&&& x_{i,j} \in \mathbb{Z}
\end{aligned} \tag{6}$$