A mathematician is a device for turning coffee into theorems.

– Alfréd Rényi

Math is hard.

– Barbie

University of Alberta

MINIMUM DEGREE SPANNING TREES ON BIPARTITE PERMUTATION GRAPHS

by

Jacqueline Smith

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2011
Edmonton, Alberta

To my parents, Jim and Jennifer.

Abstract

The minimum degree spanning tree problem is a widely studied NP-hard variation of the minimum spanning tree problem, and a generalization of the Hamiltonian path problem. Most of the work done on the minimum degree spanning tree problem has been on approximation algorithms, and very little work has been done studying graph classes where this problem may be polynomial time solvable. The Hamiltonian path problem has been widely studied on graph classes, and we use classes with polynomial time results for the Hamiltonian path problem as a starting point for graph class results for the minimum degree spanning tree problem. We show the minimum degree spanning tree problem is polynomial time solvable for chain graphs. We then show this problem is polynomial time solvable on bipartite permutation graphs, and that there exist minimum degree spanning trees of these graphs that are caterpillars, and that have other particular structural properties.

Acknowledgements

First, I would like to express my gratitude to my supervisor, Lorna Stewart, for her support and encouragement. Her guidance led to my successful completion of this thesis, and to my development as a researcher. I would also like to thank my committee members, Joe Culberson and Gerald Cliff, for their useful feedback on this work.

Thanks to Marc Bellemare for being my sounding board, for being able to do algebra, and for keeping me motivated through the hard parts. Many thanks also go to Rick Valenzano for preserving my sanity, and helping me keep things in perspective.

Finally, I am forever grateful to my friends and family for their love and support, and for tolerating me trying to teach them about graphs.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

A spanning tree of a graph is a minimum set of edges that connect all the vertices of the graph. A minimum degree spanning tree is a spanning tree of a graph with the restriction that the maximum degree of a vertex in the tree is minimized. The problem of finding the minimum maximum degree of a spanning tree is NP-hard in general. Thus, it is believed that there is no polynomial time algorithm for the problem. The best approximation algorithm finds a solution that is within one of the optimal value.

For a given NP-hard problem, there may exist restricted graph classes with particular properties that allow a polynomial time solution to be found to solve the problem on that class. The minimum degree spanning tree problem is a generalization of the problem of finding a Hamiltonian path in a graph, which is NP-hard in general. The Hamiltonian path problem is well-studied on graph classes and the problem is known to be polynomial time solvable on many classes.

In this thesis, we first motivate and define the minimum degree spanning tree problem, present some background and related work, and then show that the problem can be solved in polynomial time on chain graphs and bipartite permutation graphs. We also give a number of structural results for the longest path problem and the minimum degree spanning tree problem on bipartite permutation graphs.

1.2 Motivation

While there is a good approximation algorithm for this problem, it is still worthwhile to study this problem on graph classes as the complexity of the problem was previously unknown on most classes. The minimum degree spanning tree problem on graph classes is interesting from a theoretical standpoint to discover where the boundary is between classes where the problem is NP-hard and where it is polynomial time solvable. It is also interesting to compare the boundary for this problem to the boundary for the Hamiltonian path and longest path problems.

There is only a small amount of work done on solving the minimum degree spanning tree problem on graphs with particular properties. Czumaj and Strothmann [15] studied k-connected and planar graphs and gave some complexity results on finding a spanning tree in these graphs with maximum degree less than some bound, when certain degree conditions were satisfied. To the best of our knowledge, there does not exist any other work on the minimum degree spanning tree problem for restricted graph classes. Our work begins to fill in that gap.

The Hamiltonian path problem is a restricted version of the minimum degree spanning tree problem, and the longest path problem is a generalization of the Hamiltonian path problem that is related to the minimum degree spanning tree problem. These problems have polynomial time solutions on many graph classes. Both of these problems are linear time solvable on bipartite permutation graphs.

The class of bipartite permutation graphs is the graph class we focus on most in this thesis. Bipartite permutation graphs have a strong ordering of their vertices. This ordering has adjacency and enclosure properties that allow for polynomial time solutions to problems that are NP-hard in general. In addition to the Hamiltonian path problem and the longest path problem, a number of other polynomial time algorithms for generally hard problems exist for this class. Examples of other problems solved for bipartite permutation graphs include the path cover problem and the edge domination problem.

Fürer and Raghavachari [25] mention some practical applications of the minimum degree spanning tree problem in areas such as non-critical network broadcast-

ing. Reducing the maximum work done by each node in a network in propagating a message is critical in networks with constraints on resources like power. Additionally, there is a cost associated with splitting a message from one node to several, and one might want to reduce the number of splits made by a node. The minimum maximum degree of a spanning tree of a graph G plays an important role in the edge reconstruction conjecture. Graphs with at least $cn \log \Delta_G^*$ edges, where Δ_G^* is the minimum maximum degree of a spanning tree of G, are edge-reconstructible [8].

Although few polynomial time exact algorithms for graph classes are known, the minimum degree spanning tree problem and its variations have been widely studied in other contexts. In Chapter 2, we discuss work done in this area.

1.3 Definitions and Graph Classes

A graph G is a pair of sets (V, E), where V is a set of vertices and E is a set of unordered pairs (u, v), called *edges*, such that $u, v \in V$. Let $n = |V|$ and $m = |E|$. We may refer to V and E as $V(G)$ and $E(G)$ if the context is not clear. The *order* of a graph G is the number of vertices, n. A graph $G = (V, E, c)$ is a *weighted graph*, with cost function c from E to $\mathbb{R}$. A graph $G = (V, F)$ is *directed* if F is a set of ordered pairs (u, v), $u, v \in V$, instead of unordered pairs. We will deal only with undirected graphs in this thesis. An edge is *incident* on a vertex v if the edge is (u, v) for some u.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. A *spanning subgraph* of a graph G is a subgraph of G that contains all vertices of G. An *induced subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' = \{(u, v) | u, v \in V' \text{ and } (u, v) \in E\}$.

The *complement* of a graph $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$ such that $(u, v) \in \overline{E}$ if and only if $(u, v) \notin E$, for all $u, v \in V$.

A *path* is a sequence of vertices $v_1, v_2, ..., v_k$ with edges (v_i, v_{i+1}) for $1 \leq i < k$ and with $v_i \neq v_j$ for all $i \neq j$. The *size* of the path P is k, the number of vertices in the path, denoted $|P|$. The *length* of the path is $k - 1$, the number of edges in the path. Vertices v_1 and v_k are *endpoints* of the path. A (u, v)-path is a path with u and

v as endpoints. A *cycle* is a sequence of vertices $v_1, v_2, ..., v_k$ with edges (v_i, v_{i+1}) and (v_k, v_1), for $1 \leq i < k$ and with $v_i \neq v_j$, for all $i \neq j$. A *walk* is a sequence of vertices $v_1, v_2, ..., v_k$ with edges (v_i, v_{i+1}) for $1 \leq i < k$. Vertices may be repeated. A walk is a *closed walk* if $v_1 = v_k$.

A graph is *connected* if for every pair of vertices $u, v \in V$ there exists a path in G from u to v. A *connected component* of a graph is a maximal connected subgraph of G; $c(G)$ is the number of connected components in G. A *clique* in a graph is a set of vertices K such that every pair of vertices in K is connected by an edge. An *independent set* is a set of vertices $I \subseteq V$ such that for all pairs $u, v \in I$, $(u, v) \notin E$. The *independence number* of a graph G, denoted $\alpha(G)$, is the size of the largest independent set in G. Let S be a set of vertices such that $S \subseteq V$. We define $G - S$ to be the subgraph of G constructed by removing the vertices in S and all their incident edges. A *cutset* of a connected graph $G = (V, E)$ is a set of vertices $S \subset V$ such that $G - S$ has more than one connected component. A graph is *k-connected* if it remains connected if less than k vertices are deleted from the graph. Equivalently, a graph is k-connected if its smallest cutset is of size k.

Vertices u and v are *adjacent* if $(u, v) \in E$. The *neighbourhood* of v is the set of all vertices adjacent to v in G, and is denoted $N_G(v)$. The *degree* of a vertex v in G is the size of $N_G(v)$, denoted $deg_G(v)$. The subscript for neighbourhood and degree may be omitted if the context is clear. $\Delta(G)$ is the *maximum degree* of a vertex in G, and $\delta(G)$ is the *minimum degree* of a vertex in G. A vertex v *dominates* a vertex set S if v is adjacent to every vertex in S. That is, v dominates S if $S \subseteq N(v)$. A vertex v in a graph G is a *leaf* if $deg_G(v) = 1$.

A *tree* $T = (V, E)$ is a connected acyclic graph. More properties of trees will be discussed in Section 2.6.

A *spanning tree* of a graph G is a spanning subgraph of G that is a tree. A graph that is not connected does not have a spanning tree. Win defines a *k-tree* T of a graph G to be a spanning tree of G with $\Delta(T) = k$ [57]. In this thesis, we will use k-tree in this way. A k-tree is also commonly defined recursively as follows: A clique with $k+1$ vertices is a k-tree. Given a k-tree T_n with n vertices, a k-tree with $n+1$ vertices can be constructed by making a new vertex adjacent to the vertices of

a k-clique in T_n [3]. However, our references to k-trees will be to the first definition.

The complexity class P is the set of decision problems that can be solved in polynomial time. The class NP is the set of decision problems that are verifiable in polynomial time. That is, given a certificate for a solution to the problem, we can verify that the certificate is correct in polynomial time. The class NP-complete is the set of decision problems that are in NP, and are as hard as any problem in NP. A problem is NP-hard if and only if there exists an NP-complete problem that is polynomial time reducible to it. If an optimization problem has a decision version that is NP-complete, then the optimization problem is NP-hard. For more information on complexity classes, see [12] or [27].

Problems that are known to be hard in general sometimes have polynomial time solutions if the problem input is restricted to have certain properties. A graph class is a family of graphs that share some property. We now define a number of graph classes that have properties that are helpful in finding polynomial time solutions to the minimum degree spanning tree problem, and other related problems. These definitions can be found in [7] and [29].

A *hereditary graph class* is a graph class where any induced subgraph of a graph in the class is also in the graph class.

A *bipartite graph* has $V = X \cup Y$ where X and Y are disjoint, independent sets and is denoted $G = (X, Y, E)$. Equivalently, a bipartite graph contains no odd cycles. Instead of using n when discussing the order of a bipartite graph, we use $p = |X|$ and $q = |Y|$.

Chordal graphs are graphs where each cycle of length greater than three has a chord. A *chord* is an edge between two vertices in a cycle that is not part of the cycle.

Chordal bipartite graphs are bipartite graphs where any cycle of length greater than four has a chord. Strictly speaking, these graphs are not chordal as there can be induced cycles of length four.

Convex graphs are bipartite graphs $G = (X, Y, E)$ where there exists an ordering of X such that for every $y \in Y$, $N(y)$ is consecutive in the ordering. This ordering is a *convex* ordering. Such an ordering does not need to exist for both X

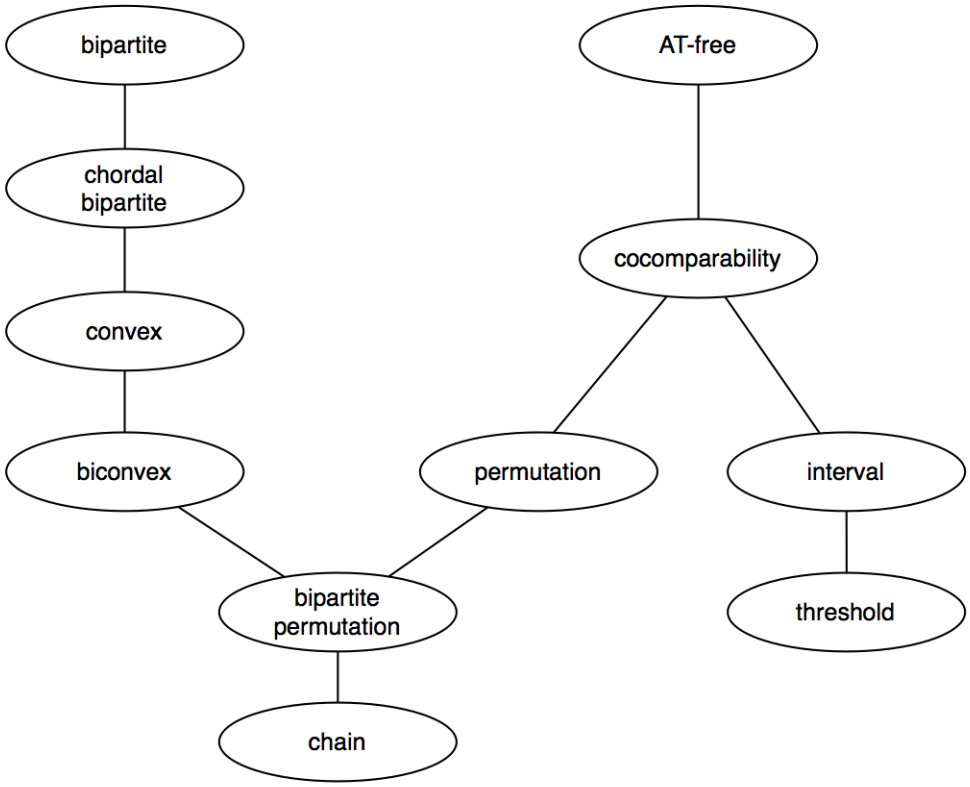

Figure 1.1: A hierarchy of the graph classes discussed in this thesis. For each class, its superclasses are above it and subclasses are below.

and Y for G to be a convex graph.

Biconvex graphs are bipartite graphs $G = (X, Y, E)$ where there exists a convex ordering of both X and Y.

Given a permutation $\pi = (\pi_1, \pi_2, ..., \pi_n)$ of the numbers 1 to n, the *permutation* graph generated by π is a graph with one vertex corresponding to each number in the permutation, with an edge between two vertices if and only if the two numbers they represent are in reverse order in the permutation. A graph is a permutation graph if and only if it is generated by some permutation.

A graph is a *bipartite permutation graph* if it is both a bipartite graph and a permutation graph. A biconvex graph is a bipartite permutation graph if and only if $(x_1, y_1), (x_p, y_q) \in E$ [1]. A bipartite permutation graph has a strong ordering, which will be defined in the next chapter.

A bipartite graph is a *chain graph* if it contains no induced $2K_2$ [58]. Therefore,

an induced subgraph of a chain graph is also a chain graph. A chain graph has a nested neighbourhood ordering, which will be defined in the next chapter.

An *asteroidal triple* is a set of three vertices such that two vertices are joined by a path that does not visit the neighbourhood of the third [14]. A graph with no asteroidal triple is called an *asteroidal-triple free (AT-free)* graph. AT-free graphs contain interval, permutation and cocomparability graphs. Bipartite permutation graphs are exactly those AT-free graphs that are also bipartite [7].

A *transitive orientation* is an assignment of a direction to each edge in a graph to produce a directed graph (V, F) that satisfies the following condition [29]:

$(a, b) \in F$ and $(b, c) \in F$ implies $(a, c) \in F$, for all $a, b, c \in V$.

A graph with such an orientation is said to be transitively orientable, and is called a *comparability graph*. *Cocomparability graphs* are graphs with transitively orientable complements. They contain both permutation graphs and interval graphs.

A graph $G = (V, E)$ is an *interval graph* if it is the intersection graph of intervals on a line. Each vertex $\{v_1, ..., v_n\} \in V$ corresponds to some interval $I_1, ..., I_n$ and $(v_i, v_j) \in E$ if and only if $I_i \cap I_j \neq \emptyset$. The interval graphs are the graphs that are AT-free and chordal.

A graph $G = (V, E)$ is a *threshold graph* if there exists a partition of V into a clique and an independent set, I, with an ordering of I, $v_1, v_2, ..., v_r$, $r = |I|$, such that $N_G(v_i) \subseteq N_G(v_{i+1})$ for $i = 1...r - 1$.

Trees are also a graph class. A subclass of trees that we will look at in this thesis is the class of caterpillars. A *caterpillar* is a tree T where the subtree P formed by removing all leaf vertices from T is a path. P is called the *spine* of T. Note that $V(T) - V(P)$ is an independent set. Equivalently, a tree is a caterpillar if and only if it does not contain the subgraph in Figure 1.2. It is worth noting that caterpillars are exactly trees that are permutation graphs [6].

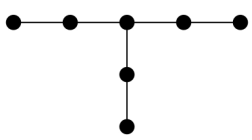

Figure 1.2: A tree is a caterpillar if and only if it does not contain this subgraph.

1.4 Problem Definitions

The minimum spanning tree problem is a classic graph theory problem, initially presented by Czech mathematician Otakar Borůvka [5]. The problem is, given a weighted graph $G = (V, E, c)$, find a spanning tree T of G such that the total cost of the edges in T is minimized over all spanning trees of G.

Problem 1.1. ***Minimum Spanning Tree:*** *Given a weighted graph $G = (V, E, c)$ find T such that $w(T) = \sum_{(u,v)\in T} w(u, v)$ is minimized.*

Borůvka's interest in the problem began when tasked with finding the most economical construction of the electric power network in southern Moravia [30]. While Borůvka was the first to explicitly define a minimum spanning tree of a graph, discussions that came close to defining the problem also existed in work on anthropological classification in the early 1900s [30].

A simplified version of the algorithm presented by Borůvka is this: given a graph G, and a tree $T = (V(G), \emptyset)$, for every connected component of T, add the shortest edge to connect that component to another connected component to $E(T)$. This algorithm assumes that each edge weight is distinct.

The two most famous solutions for the minimum spanning tree problem are Kruskal's algorithm and Prim's algorithm. Kruskal's algorithm builds a minimum spanning tree T for graph G as follows: let $T = (V(G), \emptyset)$. Then, while T is not a spanning tree of G, take the lowest weight edge $(u, v) \in G$ that is not in T such that it does not create a cycle in T, and add it to T [35]. Kruskal also gives a dual of this algorithm, which constructs T from G by at each step removing the highest weighted edge that does not disconnect the graph.

Prim's algorithm chooses an arbitrary initial vertex x, and sets $T = (V(G), \emptyset)$. Then, while T is not a spanning tree, it takes the lowest weight edge (u, v) such that u is in the same connected component as x and v is not, and adds it to T [46].

Prim published this algorithm in 1957, and it was also independently discovered by Dijkstra in 1959 [18]. However, this solution was first discovered in 1930 by

Jarník [33], in response to Borůvka's paper: the subtitle of his publication was "On a letter to O. Borůvka" [30]. It appears that Kruskal was the first to discover the solution that bears his name.

Each of these algorithms runs in polynomial time, and so the minimum spanning tree problem can be solved in polynomial time. Currently the fastest algorithm for solving the minimum spanning tree problem runs in $O(m\,\alpha(m,n))$ time on an n-vertex, m-edge graph, where α is the inverse Ackermann function, and is due to Chazelle [10].

A *minimum degree spanning tree (MDST)* of a connected graph G is a spanning tree T of G such that $\Delta(T)$ is minimum over all spanning trees of G. We denote the minimum $\Delta(T)$ of all spanning trees T of a graph G as Δ^*_G. The subscript may be omitted if the context is clear. An MDST of G is a k-tree with $k = \Delta^*_G$.

We consider the following problems for unweighted, undirected graphs.

Problem 1.2. ***Degree Constrained Spanning Tree:*** *Given a graph G and an integer k, does G have a spanning tree T such that $\Delta(T) \leq k$?*

The degree constrained spanning tree problem is NP-complete, and remains NP-complete when k is restricted to two [27]. When $k = 2$, this is the Hamiltonian path problem.

Problem 1.3. ***Minimum Degree Spanning Tree:*** *Given a graph G, compute the value of Δ^*_G.*

The minimum degree spanning tree problem is the optimization version of the degree constrained spanning tree problem, looking for the minimum k such that G has a spanning tree with $\Delta(T) = k$. This minimum k is Δ^*_G. Therefore, the minimum degree spanning tree problem is NP-hard.

Problem 1.4. ***Minimum Degree Spanning Tree Construction:*** *Given a graph G, find a spanning tree of T of G such that $\Delta(T) = \Delta^*_G$.*

1.5 Contributions

In this thesis, we investigate two graph classes, chain graphs and bipartite permutation graphs. In Chapter 3, we present a polynomial time algorithm for solving the minimum degree spanning tree problem and finding an MDST in a chain graph, as well as a formula for Δ_G^* for a chain graph G.

In Chapter 4, we give some structural results for longest paths in bipartite permutation graphs, and show that there exists an MDST of a given bipartite permutation graph that contains no edge crossings, as well as one that contains a longest path in the given graph. We combine these results to show that for a bipartite permutation graph G, there exists an MDST of G that is a crossing-free caterpillar containing a longest path. We also present a dynamic programming algorithm that computes Δ_G^* for a bipartite permutation graph G and finds an MDST of G, in polynomial time.

Chapter 2

Background

In this section, we discuss background work related to the minimum degree spanning tree problem in general and on specific graph classes.

In the first part of this chapter, we present background work on the minimum degree spanning tree problem. First, we give the complexity of problems of the minimum degree spanning tree problem and of some related problems. Since the minimum degree spanning tree problem is NP-hard in general, we then give some attention to work done on approximation algorithms for the problem and some of its variations. This review will only discuss work on undirected graphs. We also review background work on sufficient conditions for a graph to have a k-tree, along with the conditions for a graph to have a Hamiltonian path that inspired them.

The second part of this chapter discusses previous work that we use to assist us in obtaining our results. We present some vertex orderings and other properties of the bipartite graph classes introduced in the first chapter. We discuss what is known about the complexities of problems related to the minimum degree spanning tree problem on certain graph classes. Lastly, we introduce a number of properties of trees that will be used in this thesis.

2.1 Complexity, Related Problems, and Variations on the Minimum Degree Spanning Tree Problem

We now discuss some problems related to the minimum degree spanning tree problem and their complexities.

A Hamiltonian path in a graph G is a path that contains each vertex of G exactly once.

Problem 2.1. ***Hamiltonian Path:*** *Given a graph G, does G have a Hamiltonian path?*

Consider the degree constrained spanning tree problem with $k = 2$. This is equivalent to the Hamiltonian path problem, and so the minimum degree spanning tree problem is a generalization of the Hamiltonian path problem. The Hamiltonian cycle problem asks if there exists a cycle that spans G. Both the Hamiltonian path and the Hamiltonian cycle problems are NP-complete [27]. This implies the NP-completeness of the degree constrained spanning tree problem. In addition, Garey and Johnson [27] give the following reduction from the Hamiltonian path problem to the degree constrained spanning tree problem where the parameter k is a fixed constant. Let G be a graph with at least three vertices. Construct G' by adding $k-2$ leaves to each vertex of G. G' has a spanning tree with maximum degree k if and only if G has a Hamiltonian path. Thus, the degree constrained spanning tree problem is NP-complete for any fixed $k \geq 2$.

Problem 2.2. ***Longest Path (decision):*** *Given a graph G and a positive integer k, does G contain a path with k or more edges?*

Problem 2.3. ***Longest Path (optimization):*** *Given a graph G, what is the largest k such that G contains a path with k edges?*

If we let $k = |V| - 1$ in the longest path decision problem, this is the same as the Hamiltonian path problem, and so the longest path decision problem is another generalization of the Hamiltonian path problem. The longest path decision problem is NP-complete [27], and so the longest path optimization problem is NP-hard. For the purposes of this thesis, we will refer to the longest path optimization problem as the longest path problem.

For any graph class, if the Hamiltonian path problem is NP-complete, then its generalizations, the decision versions of the longest path problem and the degree constrained spanning tree problem are also NP-complete. The longest path problem and the minimum degree spanning tree problem are NP-hard on such a class.

We will see that the Hamiltonian path problem is polynomial time solvable on graph classes such as interval graphs, bipartite permutation graphs, and chain graphs. These classes are a good starting point for finding polynomial time results for the minimum degree spanning tree problem.

We now look at some generalizations of the minimum degree spanning tree problem on weighted graphs.

Given a weighted graph G, there are two constraints to be optimized: the maximum degree of any vertex in a spanning tree, and the weight of the spanning tree. Since there may not be a minimum weight tree with the minimum maximum degree, variations of this problem must involve some compromise between the weight of the spanning tree and the maximum degree of any vertex.

Problem 2.4. ***Minimum Degree Minimum Spanning Tree (MDMST):*** *Given a weighted graph G, find the minimum maximum degree of a minimum weight spanning tree of G.*

The MDMST problem finds the minimum weight of a spanning tree, but the minimum maximum degree of a minimum weight spanning tree may be higher than is possible in a spanning tree with higher weight. The MDMST problem compromises on the optimality of the maximum degree of a vertex in order to achieve optimality on the weight of the tree. In the unweighted case, the MDMST problem is the minimum degree spanning tree problem, and so it is NP-hard.

Problem 2.5. ***Bounded Degree Minimum Spanning Tree (BDMST):*** *Given a weighted graph G and a set of integer degree bounds, find a spanning tree that satisfies the degree bounds, and has smallest possible weight.*

For the BDMST problem, these degree bounds may be some constant upper-bound B for all vertices, or each vertex v can have its own upper bound B_v. There may also be a specified lower bound A_v for each vertex v as well. If no lower bound is specified, $A_v = 1$ for all v is implied, since this is required in any spanning tree. Thus, the BDMST problem finds a spanning tree T such that $A_v \leq deg_T(v) \leq B_v$ for each vertex v and the weight of T is minimized over all spanning trees satisfying these degree bounds. The BDMST problem is also NP-hard.

There are a number of other problems that are related to the minimum degree spanning tree problem, but are not generalizations or specifications of the problem. The following problems related to Minimum Degree Spanning Tree are also NP-complete [27]: Maximum Leaf Spanning Tree, Steiner Tree in Graphs, Partition into Hamiltonian Subgraphs, and Degree Bounded Connected Subgraph.

A minimum degree Steiner tree for some $D \subseteq V$ is a minimum degree tree in G that spans at least the vertices in D. The minimum degree spanning tree problem is a special case of the minimum degree Steiner tree problem, when $D = V$. This problem is used to obtain some of the approximation results presented in this chapter.

The spanning tree problem is a classic problem in graph theory, and so many variations of the problem have been studied. We will mention some problem variations here, but we will not focus on them in detail. For an extensive survey of spanning tree problems, see [45].

The problem of finding a minimum spanning tree of a graph that has a degree constraint on exactly one vertex in the graph is linear-time reducible to the minimum spanning tree problem [26]. Additionally, if only some vertices in the graph have degree constraints, and these vertices form an independent set, the problem is solvable in polynomial time [39].

While the minimum degree spanning tree problem is NP-hard, finding a spanning tree of a graph with maximum $\Delta(T)$ can be solved efficiently with a greedy algorithm. For example, for a graph $G = (V, E)$, choose $v \in V$ such that $deg_G(v) = \Delta(G)$. Begin constructing a maximum degree spanning tree T by adding all edges in G that are incident on v. Continue to add edges to T until it is a spanning tree. We have that $\Delta(T) = \Delta(G)$, and it cannot be any greater.

However, the maximization versions of other related problems are not all easy. Finding a spanning tree of a graph with the maximum possible number of leaves is NP-hard [22].

2.2 Approximations

Since the minimum degree spanning tree problem is NP-hard, much of the work done on this problem has been in the area of approximation algorithms. We discuss some approximation algorithms here; for a more complete review of this area, see [48].

In this section, we review approximation algorithms for two variations of the minimum spanning tree problem, the MDMST problem and the BDMST problem. In the unweighted case, the MDMST problem is equivalent to the minimum degree spanning tree problem.

2.2.1 Unweighted Approximations

We first discuss work on approximation algorithms for the minimum degree spanning tree problem, which is defined for unweighted graphs.

Fürer and Raghavachari [24] first gave a polynomial time algorithm that returns a spanning tree T with $\Delta(T) \in O(\Delta^* + \log n)$. The algorithm reduces the minimum degree spanning tree problem to a maximal matching in auxiliary graphs.

In [25], Fürer and Raghavachari present a better polynomial time approximation algorithm for the minimum degree spanning tree problem, and more particularly, for the minimum degree Steiner tree problem, that produces a Steiner tree T such that $\Delta(T) \leq \Delta^*_G + 1$.

The algorithm presented is shown to produce a spanning tree T of G that has $\Delta(T) \leq \Delta^*_G + 1$. Finding an upper bound on Δ^*_G is easy; simply produce any spanning tree T of G, and we know that $\Delta^*_G \leq \Delta(T)$. The proof of correctness for this algorithm relies on a *witness set* to give a lower bound on Δ^*_G and show that Δ^*_G is within one of optimal. A witness set is a subset of the vertices of G, $W \subseteq V$ with $|W| = w$, and the removal of W from G disconnects G into t components. Let $d = \lceil \frac{w+t-1}{w} \rceil$. Then [25] shows that $\Delta^*_G \geq d$.

The algorithm for producing this spanning tree of G with $\Delta(T) \leq \Delta^*_G + 1$ relies on a series of improvements to an initial arbitrary spanning tree T. At each iteration of the algorithm, a forest F is created from T by removing the set of vertices with

degree $\Delta(T)$ and $\Delta(T) - 1$ and their incident edges. If there is no edge in $E(G) - E(T)$ that is between two components of F, the algorithm terminates. Otherwise, choose $(u, v) \in E(G) - E(T)$ such that u and v are in different components of F, and consider the cycle C created by adding (u, v) to T. If there exists some vertex $w \in C$ such that $deg_T(w) = \Delta(T)$, and if $deg_T(u) \leq \Delta(T) - 2$ and $deg_T(v) \leq \Delta(T) - 2$ then w can be *improved* by adding (u, v) to T and removing an edge of C that is incident on w. Each such improvement reduces the number of vertices of T of degree $\Delta(T)$, and can potentially decrease $\Delta(T)$. If no further improvements can be made, the set of vertices of degree $\Delta(T)$ and $\Delta(T) - 1$ is a witness set that shows $\Delta(T) \leq \Delta^*_G + 1$. This algorithm runs in $O(mn\,\alpha(m, n)\log n)$, where α is the inverse Ackermann function.

2.2.2 Weighted Approximations

We now discuss approximations for some variations of the minimum degree spanning tree problem on weighted graphs.

Two results for a more general version of the minimum degree spanning tree problem are presented by Fischer in [23]. Here, we consider the minimum maximum degree over all spanning trees of a graph, Δ^*, and the minimum cost of a spanning tree with maximum degree Δ^*, $cost^*$. The first is a generalization of Fürer and Raghavachari's work in [25] to weighted graphs that produces a spanning tree T such that $cost(T) \leq cost^*$ and $\Delta(T) \leq k \cdot (\Delta^* + 1)$, where k is the number of distinct edge weights in the graph.

The second result in [23] generalizes [24] to weighted graphs as well. It produces a spanning tree T such that $cost(T) \leq cost^*$ and $\Delta(T) \in O(\Delta^* \cdot \log\ n)$. These algorithms rely on the notion of *tree rank* of a tree T, which is an array $\{t_{n-1}, ..., t_1\}$ where t_k is the number of vertices in T with degree k. These rankings are considered in lexicographic order. Then, a series of neutral weight edge swaps (swapping an edge of weight c in T for another edge of weight c not in T) are performed. A tree is considered *locally optimal* if there does not exist a neutral weight edge swap that decreases the tree rank. If T is locally optimal, then $\Delta(T) \leq b \cdot \Delta^* + \lceil \log_b n \rceil$ for some constant $b > 1$. The time complexity of this

algorithm is $O(n^{4+\frac{1}{\ln b}})$.

Goemans conjectured that, given a graph G and a constant upper bound k for all vertex degrees in a spanning tree of G, there exists a polynomial time algorithm to find a spanning tree T such that $\Delta(T) \leq k+1$ and the cost of T is at most $OPT(k)$, where $OPT(k)$ is the optimal cost of a spanning tree that satisfies the upper bound k. Later, in [28], Goemans presented an algorithm for a version of the weighted BDMST problem that produces a solution within two of optimal. This algorithm considers weighted graphs with an upper and lower bound on the degree of each vertex v. It settles a slightly weaker version of the conjecture, finding a spanning tree with cost at most $OPT(k)$ and $\Delta(T) \leq k + 2$ [28].

Ravi and Singh [47] give a generalization of Fürer and Raghavachari's $\Delta^* + 1$ approximation algorithm for the BDMST problem. Their polynomial time algorithm either shows that the degree bounds given in the problem statement are infeasible, or returns a spanning tree T such that $deg_T(v) \leq B_v + k$ where k is the number of distinct edge costs in T. Previous approximation algorithms for these problems used witness sets to find a lower bound on the optimal solution. Ravi and Singh [47] instead use linear programming to give a stronger lower bound.

More recently, Singh and Lau [50] presented an algorithm for the BDMST problem that produces a spanning tree T such that $weight(T) \leq OPT$ and $A_v - 1 \leq deg_T(v) \leq B_v + 1$, for all v, where OPT is the weight of an optimal solution and A_v, B_v are the given bounds on the degree of vertex v. This result generalizes the result in [25] to weighted graphs and proves Goemans' conjecture correct. It is also essentially the best possible result in polynomial time for the BDMST problem, unless $P = NP$.

The BDMST problem is again considered in [9], using the methodology applied by Edmonds to the weighted matching problem. Their first algorithm finds a tree T of optimal cost with $\Delta(T) \leq \frac{b}{2-b}B + \log_b n$, $b \in (1, 2)$, where B is the upper bound for all vertex degrees in T, in polynomial time. Their second algorithm improves on this by finding a spanning tree T of optimal cost with $\Delta(T) \leq B + O(\frac{\log n}{\log \log n})$ in quasi-polynomial time. A quasi-polynomial time algorithm is slower than a polynomial time algorithm, but still faster than exponential time. This second

algorithm gets a huge improvement on the accuracy of $\Delta(T)$, but at the expense of being slower than polynomial time. This improvement in error on the maximum degree is achieved using an analogy to bipartite matching, and by using augmenting path techniques.

These approximation algorithms are for general weighted graphs which may be considered as complete graphs with some edges having infinite weight. There has also been work done on finding low degree, low weight spanning trees for Euclidean graphs, where each vertex represents a point in the plane, and each pair of points is connected by an edge with weight equal to the distance between the points. The goal is to find a minimum weight spanning tree T such that the degree of each vertex in bounded above by some k. This problem is explored in [37], [49], [21], and [34]. The most recent results from [34] state that the problem is NP-hard for $2 \leq k \leq 3$ and polynomial time solvable for $k \geq 5$. The complexity is still unknown for $k = 4$, and so [34] gives an approximation algorithm that shows there always exists a spanning tree in a such a collection of points in the plane, where the maximum degree is 4, and the cost of this tree is at most $\frac{\sqrt{2}+2}{3}$ times the optimal cost.

2.3 Hamiltonicity and k-trees

We discuss conditions for Hamiltonicity and for a graph to have a k-tree related to the minimum degree of a vertex in a graph, the independence number, and the connectivity.

One of the first sufficient conditions for Hamiltonicity of a graph is Dirac's theorem, which states that for a given graph G, if $\delta(G) \geq \frac{n}{2}$, then G has a Hamiltonian cycle [19]. Ore showed that if $deg_G(x) + deg_G(y) \geq n - 1$, for all $x, y \in V$ such that $(x, y) \notin E$, then G has a Hamiltonian cycle [44]. This result introduces the theme of independent sets in results for sufficient conditions for Hamiltonicity of graphs.

Chvátal and Erdös presented further results related to independent sets and Hamiltonicity in [11]. Let G be a graph with $n \geq 3$. If G is k-connected and

has no independent set of size greater than k, then G has a Hamiltonian cycle. Similarly, if G is k-connected and has no independent set of size greater than $k+1$, then G has a Hamiltonian path [11].

Bondy and Chvátal showed, for a connected graph $G = (V, E)$ with some $(u, v) \notin E$ such that $deg_G(u) = deg_G(v) \geq n - 1$, G has a Hamiltonian path if and only if $G + (u, v)$ has a Hamiltonian path [4].

We now discuss some generalizations of these conditions for Hamiltonicity. These generalizations give a sufficient condition for a graph to have a k-tree.

Win generalized Ore's result to show that if a graph G satisfies the condition in Equation 2.1 for a given k, then G has a spanning tree T with $\Delta(T) \leq k$ [56].

Win's Condition [56]:

$$\sum_{x \in I} deg(x) \geq n - 1, \text{for every } k\text{-element independent set } I \subset V \tag{2.1}$$

Win's upper bound k on Δ^*_G of a graph G also gives a lower bound on the degrees of vertices in G. That is, $\delta(G) \geq (n-1)/k$, a generalization of Dirac's theorem.

Win proved a conjecture of Las Vergnas in [55], which generalizes the result of Chvátal and Erdös: every k-connected graph with independence number $\alpha \leq k + c$ contains a spanning tree with no more than $c + 1$ terminal vertices. Neumann-Lara and Rivera-Campo generalized this result to bounded degree spanning trees, showing that if G is k-connected with $\alpha \leq 1 + ks$ for $s \geq 1$, then G has a spanning tree with no vertices with degree larger than $s + 1$ [43]. Another result, also from [43], gives a less tight bound on the maximum degree of the tree, but also provides information about the number of vertices in the tree that have that maximum degree: let G be a k-connected graph, and $s \geq 3$, $0 \leq c \leq k$. If $\alpha \leq 1 + k(s-1) + c$ then G has a spanning tree T with no vertices with degree larger than $s + 1$, and with at most c vertices in T having degree $s + 1$.

These results are interesting on dense graphs, but sparser graphs that have Hamiltonian paths are less likely to satisfy these conditions. Czumaj and Strothmann [15] looked at algorithms for a degree bounded spanning tree problem on k-connected graphs, and also compiled results for the problem on planar graphs,

which are sparse. They presented a table of results which gives the complexity of finding a spanning tree with maximum degree less than a bound Δ_T, given the connectivity of a graph, k, and its maximum degree $\Delta(G)$. These results are given in Figure 2.1.

General Graphs					
k=2		k=3		k ≥ 4	
$\Delta(G) \leq 2\,\Delta_T - 2$	$\Delta(G) > 2\,\Delta_T - 2$	$\Delta(G) \leq 3\,\Delta_T - 4$	$\Delta(G) > 3\,\Delta_T - 4$	$\Delta(G) \leq k(\Delta_T - 2)+2$	$\Delta(G) \geq k(\Delta_T - 1)$
$O(m+n^{3/2})$	NP-complete	$O(n^2\,\alpha(n,n)\log n)$	NP-complete	$O(n^2 k\alpha(n,n)\log n)$	NP-complete

Planar Graphs					
k=1	k=2		k=3		k=4 or k=5
$\Delta(G) > \Delta_T$	$\Delta(G) \leq 2\,\Delta_T - 2$	$\Delta(G) > 2\,\Delta_T - 2$	$\Delta_T = 2$	$\Delta_T \geq 3$	$\Delta_T \geq 2$
NP-complete	$O(n \log n)$	NP-complete	NP-complete	$O(n)$	$O(n)$

Figure 2.1: Complexities of finding a spanning tree T with $\Delta(T) \leq \Delta_T$ in a k-connected general or planar graph G with $\Delta(G)$ bounded. [15]

Another generalization of Win's work is by Zhenhong and Baoguang, who presented results related to the edge-connectivity of a graph. They defined a d_k-tree as a tree T such that for all $v \in V(T)$, $deg_T(v) \leq \lfloor \frac{deg_G(v)+k-1}{k} \rfloor + c$, where c depends on k [59]. Then, if G is a k-edge-connected graph, with $k \geq 2$, G has a d_k-tree with $c = 1$ for $k = 2$ and $c = 2$ for $k \geq 3$ [59].

Win's condition for $k \geq 2$, Equation 2.1, was explored further by Czygrinow et al in [16]. They also explored the structure of spanning trees with small maximum degree. Their results state that given a graph G that satisfies Equation 2.1 for some k, then either:

1. G has a spanning tree T with $\Delta(T) < k$, or

2. G is a graph made up of k cliques, $G_1, G_2, ..., G_k$ with $|V(G_i)| > 1$ for at least 3 distinct i, and one dominating vertex adjacent to all vertices of $G_1, ..., G_k$, or

3. for every maximum length path P in G, there exists a spanning tree T of G such that

   (a) T is a caterpillar,

   (b) $\Delta(T) = k$,

(c) the spine of T is P,

(d) the set $\{v \in V | deg_T(v) \geq 3\}$ is an independent set in T.

There exists a k that satisfies Equation 2.1 for all graphs, although this k does not necessary give a good bound on Δ_G^*. Consider a graph G that is a clique K_r, $r \geq 3$, with r leaves, each adjacent to exactly one vertex in the clique. The size of the largest independent set in G is r, but the sum of the degrees of this set is $r < 2r-1$. Then $k = r + 1$ as there is no independent set of size $r + 1$. On the other hand, regardless of the size of G, $\Delta_G^* = 3$. For large r, Equation 2.1 gives a poor bound on Δ_G^*. This bound is tighter for denser graphs, and worse for sparse graphs and graphs with many leaves.

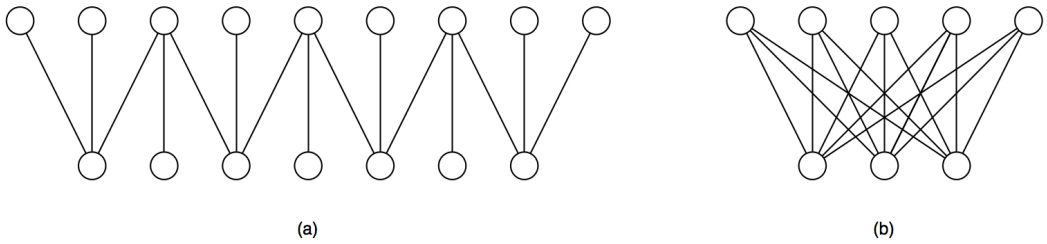

Figure 2.2: Two bipartite permutation graphs where Win's condition, Equation 2.1, for a k-tree is not tight and tight, respectively.

Figure 2.2 shows two bipartite permutation graphs where Win's condition has different accuracy. In Figure 2.2 (a), $n = 16$ and the condition is satisfied at $k = 10$, as there is no independent set of size 10, and there exists an independent set of size 9 with degree sum less than $n - 1$ (the set of all leaf vertices, for example). But the graph itself is a caterpillar and so $\Delta_G^* = \Delta(G) = 3$. So Win's condition is not accurate here. The graph in Figure 2.2 (b) has $n = 8$, and satisfies Win's condition at $k = 3$. For this graph, $\Delta_G^* = 3$, so Win's condition is tight. These examples show that these results of [16] will not necessarily give much information about k-trees in bipartite permutation graphs.

Bondy and Chvátal's work was generalized by Kishimoto and Kano for j-connected graphs. They present three results in [38]. Let G be a j-connected graph, for some $j \geq 1$, and let $k \geq 2$. First, for some $(u, v) \notin E$ such that

$deg_G(u) + deg_G(v) \geq n - 1 - (k-2)j$, then G has a spanning k-tree if and only if $G + (u, v)$ has a spanning k-tree. Next, for some $(u, v) \notin E$ such that $deg_G(u) + deg_G(v) \geq n - k + 1$, then G has a spanning k-tree if and only if $G + (u, v)$ has a spanning k-tree. Finally, if for all $(u, v) \notin E$, we have $deg_G(u) + deg_G(v) \geq n - 1 - (k-2)j$, then G has a spanning k-tree.

2.4 Vertex Orderings and Properties of Bipartite Graph Classes

We now discuss vertex orderings for bipartite graph classes.

A *convex ordering* of one vertex set of a bipartite graph (assume X) is an ordering of the vertices of X such that for every $y \in Y$, $N(y)$ is consecutive in the ordering of X. As previously mentioned, a bipartite graph $G = (X, Y, E)$ is convex if there exists a convex ordering of X or Y, and is biconvex if there exists a convex ordering of both X and Y.

An ordering of the vertices X of a bipartite graph $G = (X, Y, E)$ has the *adjacency property* if for each vertex $y \in Y$, the vertices of $N(y)$ are consecutive in the ordering of X, and has the *enclosure property* if for every pair of vertices $u, v \in Y$, if $N(u) \subset N(v)$, then $N(v) - N(u)$ is consecutive in the ordering of X.

A *strong ordering* of a bipartite graph $G = (X, Y, E)$ is an ordering $x_1, ..., x_p$ of the vertices in X and $y_1, ..., y_q$ of the vertices in Y, such that for all (x_{i_1}, y_{j_1}), $(x_{i_2}, y_{j_2}) \in E$ with $x_{i_1} < x_{i_2}$ and $y_{j_1} > y_{j_2}$, (x_{i_1}, y_{j_2}), $(x_{i_2}, y_{j_1}) \in E$. A strong ordering is symmetric. That is, if $x_1, x_2, ..., x_p, y_1, y_2, ..., y_q$ is a strong ordering of a bipartite graph $G = (X, Y, E)$, then $x_p, x_{p-1}, ..., x_1, y_q, y_{q-1}, ..., y_1$ is also a strong ordering of G.

Figure 2.3 shows a bipartite permutation graph with vertices ordered as in a strong ordering of G.

Theorem 2.6. *The following are equivalent for a bipartite graph $G = (X, Y, E)$ [51]:*

1. *G is a bipartite permutation graph.*

2. *There exists a strong ordering of $X \cup Y$.*

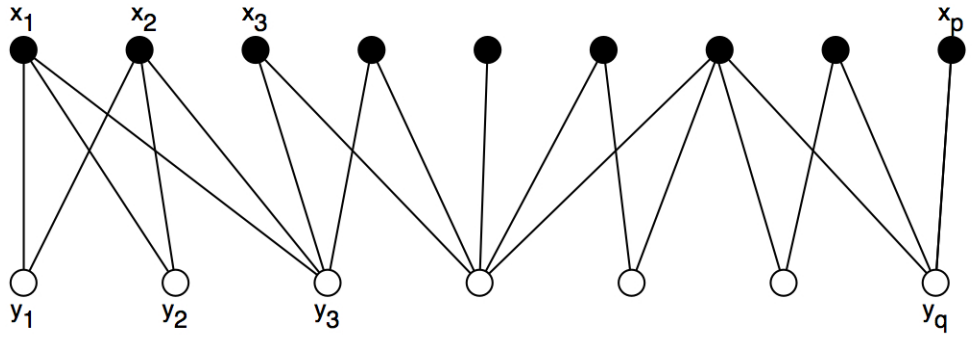

Figure 2.3: A bipartite graph with X and Y ordered as in a strong ordering of G.

3. *There exists an ordering of X that has both the adjacency and the enclosure properties.*

A *nested neighbourhood* ordering of a bipartite graph is an ordering $x_1, ..., x_p$, $y_1, ..., y_q$ of the vertices such that $N(x_i) \supseteq N(x_j)$, for $i < j$, and $N(y_i) \supseteq N(y_j)$, for $i < j$. In a nested neighbourhood ordering, the highest degree vertices have the lowest index, and the lowest degree vertices have the highest index. An additional property of a nested neighbourhood ordering is if $i < j$, then $deg(x_i) \geq deg(x_j)$ and $deg(y_i) \geq deg(y_j)$. A graph is a chain graph if it has a nested neighbourhood ordering.

Figure 2.4 shows a chain graph with vertices ordered as in a nested neighbourhood ordering.

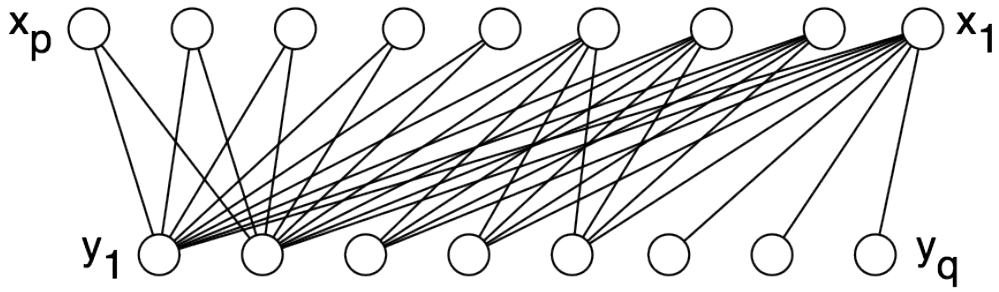

Figure 2.4: A graph with a nested neighbourhood ordering.

Note that a nested neighbourhood ordering is not a strong ordering, but reversing

the ordering of either X or Y gives a strong ordering. That is, if $x_1, ..., x_p, y_1, ..., y_q$ is a nested neighbourhood ordering, then $x_p, ..., x_1, y_1, ..., y_q$ is a strong ordering.

We use the strong ordering property of some bipartite graphs to introduce more definitions and properties for these graphs.

A path P in a bipartite graph $G = (X, Y, E)$ with strong ordering $x_1, ..., x_p$ and $y_1, ...y_q$ is a *zig-zag* if P can be written as $\{x_{a_1}, y_{b_1}, x_{a_2}, y_{b_2}, ..., x_{a_i}, y_{b_i}, (x_{a_{i+1}})\}$, where $a_j < a_k$ and $b_j < b_k$ for $j < k$. From [52], we know that every bipartite permutation graph has a longest path that is a zig-zag. Figure 2.5 shows in bold two longest paths in the strongly ordered bipartite graph from Figure 2.3. The path in Figure 2.5 (a) is not a zig zag, while the path in Figure 2.5 (b) is a zig zag.

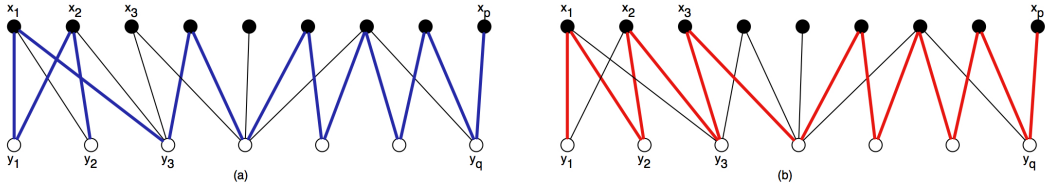

Figure 2.5: Two longest paths in a bipartite graph: (a) is not a zig zag, while (b) is a zig zag.

An *edge crossing* in a bipartite graph $G = (X, Y, E)$ with ordering $x_1, ..., x_p$ and $y_1, ...y_q$ is a pair of edges $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}) \in E$ where $i_1 < i_2$ and $j_1 > j_2$. For example, the edges $(x_1, y_3), (x_2, y_2)$ in Figure 2.3 are an edge crossing.

Note that a path in a bipartite graph is a zig-zag if and only if it has no edge crossings.

2.5 Algorithms and Complexity for Hamiltonicity and other Related Problems on Graph Classes

Examining algorithms for solving the Hamiltonian path and longest path problems on restricted graph classes may provide insight into how to solve the minimum degree spanning tree problem. The main focus of this thesis is on bipartite permutation graphs and their superclasses, subclasses, and other related graph classes, as in Figure 1.1. We discuss what is known about the complexity of Hamiltonian path

and longest path on the classes shown in Figure 1.1.

We start with one of the highest level superclasses of bipartite permutation graphs as shown in Figure 1.1, bipartite graphs. Moon and Moser explored the Hamiltonicity of bipartite graphs in [41], and showed both the Hamiltonian cycle and Hamiltonian path problems remain NP-complete on bipartite graphs.

The most basic necessary condition for a bipartite graph $G = (X, Y, E)$ to have a Hamiltonian cycle is that $p = q$. For a Hamiltonian path to exist in a bipartite graph, we must have $|p - q| \leq 1$. Moon and Moser [41] present some sufficient conditions for the existence of a Hamiltonian cycle in a bipartite graph. Let $p = |X| = |Y|$ for the following results. A bipartite graph $G = (X, Y, E)$ has a Hamiltonian cycle if for any non-empty subset F of $X \cup Y$ with $|F| = k \leq \frac{p}{2}$, such that for every $x_i \in F$, $deg_G(x_i) \leq k$, then every vertex $y_j \in Y$ with $deg_G(y_j) \leq p - k$, y_j is adjacent in G to some vertex in F. The same holds with X and Y reversed. A corollary to this is that for a bipartite graph $G = (X, Y, E)$, if for each k, $1 \leq k \leq \frac{p}{2}$, the number of vertices $x \in X$ such that $deg_G(x) \leq k$ is less than k, and the same for all $y \in Y$, then G has a Hamiltonian cycle. This can also be stated as, if G has $deg_G(x) + deg_G(y) > p$ for all $x, y \in G$ such that $(x, y) \notin E$, then G has a Hamiltonian cycle.

Chordal bipartite graphs are contained in bipartite graphs, and contain bipartite permutation graphs. Müller [42] gives a reduction from SAT to both Hamiltonian cycle and Hamiltonian path in chordal bipartite graphs.

A Hamiltonian cycle can be found in a convex graph in polynomial time, as shown in [42]. Keil showed that there exists a linear time algorithm for finding a Hamiltonian cycle in an interval graph [36], and [42] presents an $O(n^2)$ reduction from a convex bipartite graph $G = (X, Y, E)$ to an interval graph G' such that there exists a Hamiltonian cycle in G' if and only if there exists a Hamiltonian cycle in G. The reduction is as follows: assume X is convex. Let $G' = (X \cup Y, E \cup \{(y_1, y_2) | N_G(y_1) \cap N_G(y_2) \neq \emptyset\})$. G' is now an interval graph and Keil's algorithm can find a Hamiltonian cycle C in linear time. Since X is an independent set, all edges in the cycle must be in E. Therefore C is also a Hamiltonian cycle in G.

Müller shows that the Hamiltonian path problem can be solved in $O(n^2)$ time

for convex and biconvex graphs. The complexity of the longest path problem is unknown for these classes [52].

A number of polynomial time results for problems related to the minimum degree spanning tree problem are known for bipartite permutation graphs. Spinrad, Brandstädt and Stewart [51] present conditions for a bipartite permutation graph $G = (X, Y, E)$ to have a Hamiltonian path or Hamiltonian cycle.

Theorem 2.7. *[51] Let $G = (X, Y, E)$ be a bipartite permutation graph containing a Hamiltonian path beginning at $x \in X$. Let $x_1, ..., x_p$ and $y_1, ...y_q$ be a strong ordering of G. Then $(x_1, y_1, x_2, y_2, ..., x_p, y_p)$ or $(x_1, y_1, x_2, y_2, ..., x_{p-1}, y_q, x_p)$ is also a Hamiltonian path in G, where $x_1, x_2, ..., x_p$, $y_1, y_2, ..., y_q$ is a strong ordering of G. Similarly, if G has a Hamiltonian path beginning at some $y \in Y$, then $(y_1, x_1, y_2, x_2, ..., y_q, x_q)$ or $(y_1, x_1, y_2, x_2, ..., y_{q-1}, x_p, y_q)$ is also a Hamiltonian path.*

Therefore, determining if a bipartite permutation graph has a Hamiltonian path reduces to just determining if it contains one of the paths in Theorem 2.7. Uehara and Uno give a linear time algorithm to solve the longest path problem on bipartite permutation graphs [53].

The results for the Hamiltonian path problem on bipartite permutation graphs apply to chain graphs, which are a subclass of bipartite permutation graphs. The longest path problem can be solved in $O(n)$ time on chain graphs, as in [53].

The complexity of the minimum degree spanning tree problem was not known for any graph class between chain graphs and convex bipartite graphs, prior to our work.

The other highest level superclass of bipartite permutation graphs that we will discuss here are AT-free graphs. The complexity of the Hamiltonian cycle and Hamiltonian path problems on AT-free graphs is unknown [2]. It is also unknown for the minimum degree spanning tree problem, and for longest path.

Both the Hamiltonian path problem is solvable in polynomial time for cocomparability graphs [17]. Recently, Mertzios and Corneil [40] showed that the longest path problem is also polynomial time solvable for cocomparability graphs. Therefore, the cocomparability graphs and their subclasses provide an interesting starting

point for exploring the minimum degree spanning tree problem. The Hamiltonian path problem and longest path problem can be solved in polynomial time for permutation graphs, since all permutation graphs are cocomparability graphs.

It is also useful to consider graph classes such as interval graphs and threshold graphs, although they are not directly related to bipartite permutation graphs. As we have seen when discussing the Hamiltonicity of convex bipartite graphs, there is a connection between convex bipartite graphs and interval graphs. Threshold graphs are very similar to chain graphs, and can be thought of as a chain graph with edges added to either X or Y of a chain graph to form a clique. Both of these classes also have polynomial time results for problems related to the minimum degree spanning tree problem.

As previously mentioned, Keil showed there is a linear time solution for the Hamiltonian path problem on interval graphs [36]. Uehara and Uno showed that the longest path problem on interval graphs can be reduced to the longest path problem on convex bipartite graphs [52]. Recently, Ioannidou et al. gave a $O(n^4)$ algorithm to solve the longest path problem on interval graphs [32].

Harary and Peled give a necessary and sufficient condition for a threshold graph to have a Hamiltonian cycle [31]. This condition gives a lower bound on the degree of each vertex in G that holds if and only if G has a Hamiltonian cycle. A condition for a Hamiltonian path is not given, but it is easy to see how the Hamiltonian cycle condition might be altered for the Hamiltonian path problem. Uehara and Uno also showed the longest path problem has a linear time solution for threshold graphs [52].

The results discussed in this section are summarized in Table 2.1. Note that there is no citation for the NP-hard results for the longest path problem where the Hamiltonian path problem is known to be NP-complete since a solution to the longest path problem gives a solution to the Hamiltonian path problem.

Graph Class	Hamiltonian Path	Longest Path
bipartite	NP-c [41]	NP-hard
chordal bipartite	NP-c [42]	NP-hard
convex	$O(n^2)$ [42]	unknown
biconvex	$O(n^2)$ [42]	unknown
bipartite permutation	$O(n)$ [51]	$O(n)$ [53]
chain	$O(n)$ [51]	$O(n)$ [53]
AT-free	unknown	unknown
cocomparability	$O(n^2)$ [17]	$O(n^4)$ [40]
permutation	$O(n^2)$ [17]	$O(n^4)$ [40]
interval	$O(n)$ [36]	$O(n^4)$ [32]
threshold	$O(n)$ [31]	$O(n)$ [52]

Table 2.1: Complexity results for the Hamiltonian path and longest path problems for certain graph classes

2.6 Properties of Trees

We make use of properties of trees in many of the proofs in this thesis. In particular, we use these properties to show that trees we construct from other trees using particular techniques retain the properties of the initial trees.

Theorem 2.8 gives several characteristics of trees.

Theorem 2.8. *For a graph G, the following are equivalent [54]:*

1. *G is connected and has no cycles.*
2. *G is connected and has $n - 1$ edges.*
3. *G has $n - 1$ edges and no cycles.*
4. *G has no cycles and has, for each $u, v \in V(G)$, exactly one (u, v)-path.*
5. *G is a tree.*

There are many other properties of trees and spanning trees that we make use of in this thesis. For these and other properties, see [54].

We know some specific things about the connectivity and colouring of trees. Adding one edge anywhere in a tree forms a cycle. Every non-leaf vertex in a tree

is a cut vertex, and every edge in a tree is a cut edge. Since trees have no cycles, they are bipartite.

Every connected graph has a spanning tree, and a graph that is a tree has exactly one spanning tree. A path is a tree with maximum degree two.

Let T and T' be two distinct spanning trees of a connected graph G. Let e be an edge in $E(T) - E(T')$. Then there exists an edge e' in $E(T') - E(T)$ such that $T - e + e'$ is a spanning tree of G.

Chapter 3

Chain Graphs

As previously defined, a bipartite graph $G = (X, Y, E)$ is a chain graph if it has no induced $2K_2$, and a chain graph has an ordering of its vertices $x_1, ..., x_p$, $y_1, ..., y_q$ that is a nested neighbourhood ordering. That is, $N(x_i) \supseteq N(x_j)$, for $i < j$, and $N(y_i) \supseteq N(y_j)$, for $i < j$. In this chapter, we assume that $|X| \geq |Y|$; that is, that X is the larger side of the graph.

Let $G = (X, Y, E)$ be a chain graph with nested neighbourhood ordering $x_1, ..., x_p$ and $y_1, ..., y_q$. Recall that therefore $x_p, ..., x_1$, $y_1, ..., y_q$ is a strong ordering of G. By the definition of the nested neighbourhood ordering, x_1 is the high degree vertex in X and y_1 is the high degree vertex in Y. The nested neighbourhood ordering also has the property that if $i < j$, then $deg(x_i) \geq deg(x_j)$ and $deg(y_i) \geq deg(y_j)$. Since G is a chain graph, x_1 dominates Y and y_1 dominates X. The vertices in X will be drawn with x_p on the left and x_1 on the right. The vertices in Y will be drawn with y_1 on the left and y_q on the right.

For the purposes of this work, we will only consider chain graphs that are connected. A disconnected graph does not have an MDST.

In this chapter, we give a degree condition for a chain graph to have a Hamiltonian path, and present an algorithm to find a spanning tree with a certain maximum degree in a chain graph. We then show that this spanning tree is an MDST of G.

3.1 A Degree Condition for Hamiltonian Chain Graphs

We give an exact condition for a chain graph to have a Hamiltonian path. Equivalently, the condition holds if and only if a chain graph has an MDST with $\Delta_G^* = 2$.

Chain graphs are a subclass of bipartite permutation graphs, and we make use of Theorem 2.7 in our proof.

Theorem 3.1. *Let $G = (X, Y, E)$ be a chain graph with nested neighbourhood ordering $x_1, ..., x_p$ and $y_1, ...y_q$. Then G has a Hamiltonian path if and only if one of the following conditions holds:*

(1) $p = q$, *and*
$deg_G(x_i) \geq p - i + 1$, *for all* $x_i \in X$, *and*
$deg_G(y_j) \geq p - j + 1$, *for all* $y_j \in Y$.

(2) $p = q + 1$, *and*
$deg_G(x_i) \geq p - i + 1$, $2 \leq i \leq p$, $deg_G(x_1) = q$, *and*
$deg_G(y_j) \geq q - j + 2$, *for all* $y_j \in Y$.

Proof. We first show that these conditions are sufficient for G to have a Hamiltonian path.

Observation: Since G is a chain graph, we know x_1 dominates Y and y_1 dominates X, and the neighbourhood of each vertex is consecutive in the nested neighbourhood ordering. We observe that, if $deg_G(x_i) \geq d_x$, then $N_G(x_i) \supseteq \{y_1, ..., y_{d_x}\}$. Similarly, if $deg_G(y_j) \geq d_y$, then $N_G(y_j) \supseteq \{x_1, ..., x_{d_y}\}$.

Recall that if $x_1, ..., x_p,\ y_1, ..., y_q$ is a nested neighbourhood ordering, then $x_p, ..., x_1, y_1, ..., y_q$ is a strong ordering.

Assume condition (1) holds. We show that G has the Hamiltonian path x_p, y_1, x_{p-1}, y_2, ..., x_1, y_p. We now show that condition (1) guarantees we have this path.

We first show that condition (1) guarantees we have each (x_i, y_j) edge in this path. Note that we can rewrite the indices in that path x_p, y_1, x_{p-1}, y_2, ..., x_1, y_p to be x_p, y_{p-p+1}, x_{p-1}, $y_{p-(p-1)+1}$, ..., x_1, y_{p-1+1}. Thus, we want to show that for each x_i, the edge (x_i, y_{p-i+1}) is in E. By our assumption that condition (1) holds, then

$deg_G(x_i) \geq p-i+1$, and by our earlier observation, then $N_G(x_i) \supseteq \{y_1, ..., y_{p-i+1}\}$. Therefore, $(x_i, y_{p-i+1}) \in E$ for $1 \leq i \leq p$.

It remains to be shown that $(y_j, x_{p-j}) \in E$ for all $1 \leq j < p$. By the assumption that condition (1) holds, $deg_G(y_j) \geq p - j + 1$, and by our earlier observation, then $N_G(y_j) \supseteq \{x_1, ..., x_{p-j+1}\}$, and so $(y_j, x_{p-j}) \in E$ for $1 \leq j < p$.

Therefore, if condition (1) holds, then G has a Hamiltonian path.

Now assume condition (2) holds. Using the same argument as in the proof for condition (1), we see that $(x_i, y_{p-i+1}) \in E$ for $2 \leq i \leq p$, and $(y_j, x_{p-j}) \in E$ for $1 \leq j \leq q$. Therefore, G has the Hamiltonian path x_p, y_1, x_{p-1}, y_2, ..., x_2, y_{p-1}, x_1.

We now show that these conditions are necessary for G to have a Hamiltonian path.

Assume $p = q$ and condition (1) does not hold for all vertices. By Theorem 2.7, if neither x_p, y_1, x_{p-1}, y_2, ..., x_1, y_p nor y_1, x_p, y_2, x_{p-1}, ..., y_p, x_1 is a Hamiltonian path of G, then G has no Hamiltonian path. If condition (1) does not hold, then there exists a vertex x_i such that $deg_G(x_i) < p - i + 1$, or a vertex y_j such that $deg_G(y_j) < p - j + 1$.

If $deg_G(x_i) < p-i+1$, then $N_G(x_i) \subseteq \{y_1, ..., y_{p-i}\}$. Therefore, $(x_i, y_{p-i+1}) \notin E$. This edge is in both x_p, y_1, x_{p-1}, y_2, ..., x_1, y_p and y_1, x_p, y_2, x_{p-1}, ..., y_p, x_1, and therefore G does not have a Hamiltonian path. An equivalent argument can be made if $deg_G(y_j) < p - j + 1$.

We can also, by the same argument, see that if $p = q + 1$ and condition (2) is not satisfied for some vertex, then G does not have a Hamiltonian path.

Therefore, the conditions are necessary and sufficient, and so the theorem holds.

□

3.2 A Minimum Degree Spanning Tree Construction Algorithm for Chain Graphs

We present an algorithm that constructs a spanning tree with a certain maximum degree on a chain graph. Later, we will show that this algorithm solves the minimum degree spanning tree problem and the minimum degree spanning tree construction

problem on chain graphs.

3.2.1 Algorithm

Let $G = (X, Y, E)$ be a chain graph with a nested neighbourhood ordering $x_1, ..., x_p$ and $y_1, ..., y_q$. We present an algorithm to construct T, a spanning tree of G.

The edges of T will be determined by partitioning G into subgraphs, constructing spanning trees on the subgraphs, and then connecting the resulting components. We define a *part* to be one of these subgraphs. The parts will be chosen using a formula that finds a set of vertices in either X or Y of the unpartitioned portion of G such that each vertex in this set has degree less than or equal to some value, and the size of this set divided by the size of its neighbourhood is maximum.

As we construct the partition of G, each part will consist of consecutive sets of vertices and will be chosen from one end of the unpartitioned portion of G. The maximum part in the unpartitioned portion of G may have its larger vertex set from either X or Y. Since every induced subgraph of a chain graph is also a chain graph, each part is a chain graph. As stated at the beginning of this chapter, we assume $|X| \geq |Y|$. Therefore we will assume that $|X(C)| > |Y(C)|$, and so there may exist a part C of G where $X(C) \subset Y(G)$ and $Y(C) \subset X(G)$. Through the rest of the chapter, we will refer to the X and Y sides of each part. Note that these labels in the part do not necessarily correspond to X and Y in G. For a given part C, $X(C)$ is the larger side of C.

We use four algorithms to construct this spanning tree for a chain graph G. ChainGraphMDST is Algorithm 1, Partition is Algorithm 2, PartMDST is Algorithm 3, and ComputeRatios is Algorithm 4. The algorithms can be found at the end of this subsection, and are further explained and proved correct in Subsection 3.2.2. Later in the chapter, we show these algorithms solve the minimum degree spanning tree problem.

We define the following values and ratios for use in the algorithms and in Theorem 3.5:

For a chain graph $G = (X, Y, E)$, let k be the number of distinct vertex degrees in X and let l be the number of distinct vertex degrees in Y. Let $a_1, ..., a_k$ be the

distinct degrees of vertices in X with $a_1 < a_2 < ... < a_k$, and let α_i be the number of vertices in X with degree $\leq a_i$. Let $b_1, ..., b_l$ be the distinct degrees of vertices in Y with $b_1 < b_2 < ... < b_l$, and let β_i be the number of vertices in Y with degree $\leq b_i$. Note that the vertices with degree $\leq a_i$ will be $x_p, ..., x_{p-\alpha_i+1}$ and the vertices with degree $\leq b_i$ will be $y_q, ..., y_{q-\beta_i+1}$. That is, these vertices will be sequential starting from the highest indexed vertices in X or Y.

The ratios below are used to determine how to partition the chain graph to construct the MDST.

For all i, $1 \leq i \leq k$,

$$f_X(i, G) = \begin{cases} 0 & \text{if for some } x \in X \text{ with } deg_G(x) = a_i, \exists\ y \in N_G(x) \text{ s.t. } deg_G(y) = 1 \\ \frac{\alpha_i}{a_i} & \text{otherwise} \end{cases}$$

For all i, $1 \leq i \leq l$,

$$f_Y(i, G) = \begin{cases} 0 & \text{if for some } y \in Y \text{ with } deg_G(y) = b_i, \exists\ x \in N_G(y) \text{ s.t. } deg_G(x) = 1 \\ \frac{\beta_i}{b_i} & \text{otherwise} \end{cases}$$

The ComputeRatios algorithm shows how to compute these values.

We now show a detailed example of how these algorithms work before presenting them formally.

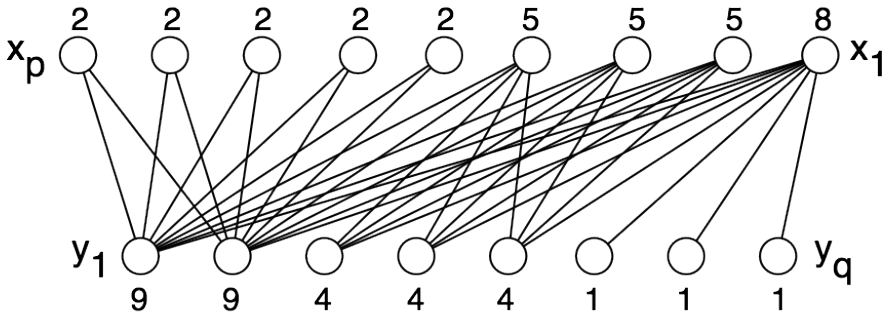

Figure 3.1: A chain graph with degrees, a_i and b_i, labeled.

Given the chain graph in Figure 3.1, we have $a_1 = 2$, $a_2 = 5$, $a_3 = 8$, $b_1 = 1$, $b_2 = 4$, and $b_3 = 9$. Therefore, we have $\alpha_1 = 5$, $\alpha_2 = 8$, $\alpha_3 = 9$, $\beta_1 = 3$, $\beta_2 = 6$, and $\beta_3 = 8$.

Using these values, we get the following for f_X and f_Y: $f_X(1,G) = \frac{5}{2}$, $f_X(2,G) = \frac{8}{5}$, $f_X(3,G) = 0$, $f_Y(1,G) = \frac{3}{1}$, $f_Y(2,G) = \frac{6}{4}$, and $f_Y(3,G) = \frac{8}{9}$. These values are computed by the ComputeRatios algorithm.

Now that we have the values of f_X and f_Y, we use these constraints to partition the graph into a number of parts, starting with the largest ratio. Each part consists of a set of vertices that must be connected to the other set in the part in a spanning tree of the graph. We will see that these parts can be chosen from either end of the graph, since it is not known whether the highest constraint will come from f_X or f_Y. These parts will be denoted C_i, with $X(C_i)$ being the larger side and $Y(C_i)$ being the smaller side. These parts are constructed by the Partition algorithm.

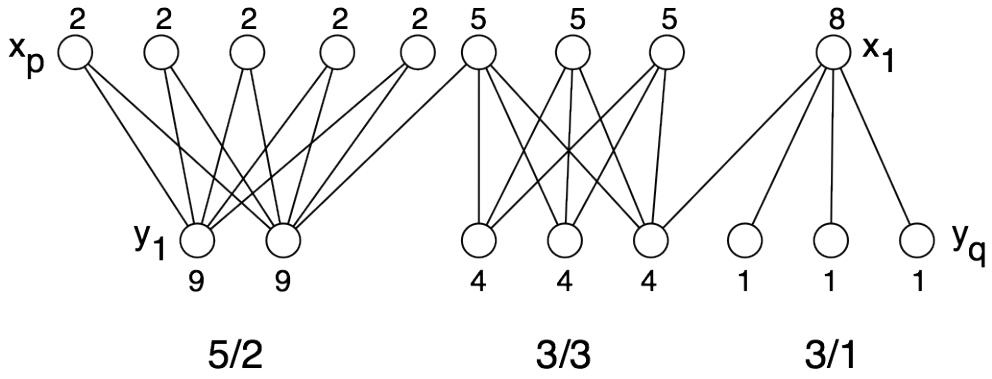

Figure 3.2: A chain graph partitioned. From left to right, the parts are C_2, C_3, and C_1.

In Figure 3.2, we see the partitions of this graph as indicated by the values of f_X and f_Y. The value of $f_Y(1,G)$ was largest, and so the first part, C_1, consists of the highest indexed β_1 vertices in $Y(G)$ and their neighbourhood.

Next, we want to construct spanning trees on each part. We do this in a greedy manner, by assigning each vertex in $X(C_i)$ to a vertex in $Y(C_i)$ and adding an edge between them, for each C_i. The way the parts are chosen allows us to distribute the edges over $Y(C_i)$ in such a way that the highest degree and the lowest degree of a vertex here differs by at most one.

In Figure 3.3, we see the first step in creating a spanning tree on a part. Each vertex in $X(C_i)$ has degree one and degrees of vertices in $Y(C_i)$ differ by at most

one.

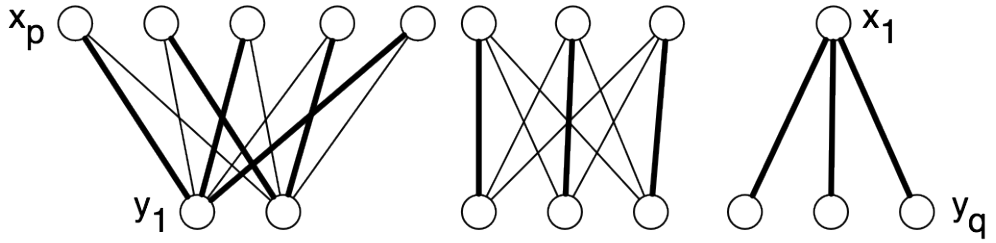

Figure 3.3: Constructing spanning trees on each part.

The second step in creating spanning trees on parts is to connect the components. There is one component for each vertex in $Y(C_i)$, and we denote this number as q_i. We now greedily add the $q_i - 1$ edges required to connect the spanning forest of each part, and maintain the difference between highest and lowest degree of vertices in $Y(C_i)$ to be no more than one. The highest degree of a vertex in $Y(C_i)$ also increases by at most one.

Figure 3.4 shows the spanning trees constructed on each part. These spanning trees on parts are constructed by the PartMDST algorithm.

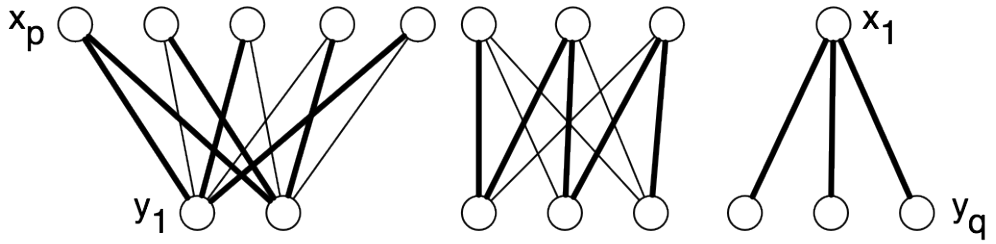

Figure 3.4: A spanning tree is constructed on each part in the partition of a chain graph.

Once a spanning tree has been constructed on each part, we must connect each part to create a spanning tree of the whole graph. We add the edge crossing each gap between parts from the lower indexed vertex in $Y(G)$ to the lower indexed vertex in $X(G)$. These edges are indicated in bold in Figure 3.5. The dashed edges cannot be used to connect the parts since they do not exist in the graph.

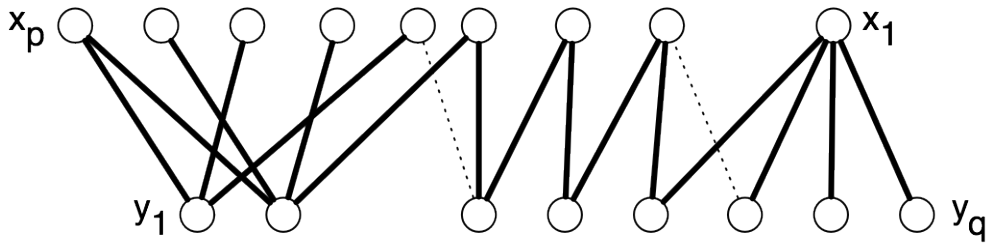

Figure 3.5: Spanning trees on each part are connected to form a spanning tree for the graph.

We have created a spanning tree for a chain graph with a certain maximum degree. The maximum degree of the spanning tree is determined by the ratio, f_X or f_Y, associated with C_1. In our example, the maximum degree of the spanning tree is four. Looking at C_1, we can see why we have this maximum degree: we must add edges from each of the three vertices with degree one in $Y(G)$ to their neighbour in $X(G)$. Then, we must connect this component to the rest of the spanning tree by adding another edge to this one vertex.

Since the ratio $\frac{3}{1}$ is larger than any other ratio in the graph, no other part will produce a maximum degree any larger than that of C_1. Therefore, we can see that C_1 gives the maximum degree of a vertex in this spanning tree. These claims will be proven formally later in this chapter.

We now present the algorithms formally.

Algorithm 1: ChainGraphMDST

Input: A chain graph $G = (X, Y, E)$ with $|X| \geq |Y|$
Output: A spanning tree T of G

// initialize T to contain the vertices of G and no edges
$T = (X, Y, \emptyset)$;
// G' is the unpartitioned portion of G
$G' \leftarrow G$;
$i \leftarrow 1$;
while $G' \neq \emptyset$ **do**
    // C_i is the next part of G'
    $C_i \leftarrow$ Partition(G');
    // build an MDST for C_i and add these edges to T
    $T \leftarrow T \cup$ PartMDST(C_i);
    $G' \leftarrow G' - C_i$;
    $i \leftarrow i + 1$;
end
$r \leftarrow i - 1$;
// connect the components of T from left to right in a drawing of G
$\mathbf{C} =$ an ordering of the parts C_i such that $C_j < C_k$ if C_j is to the left of C_k in a drawing of G;
// j is the index of a part in the ordering $\mathbf{C}$
foreach $j = 1...r - 1$ **do**
    $a \leftarrow \min\{k \mid x_k \in X(G) \cap \mathbf{C}[j]\}$;
    $b \leftarrow \max\{k \mid y_k \in Y(G) \cap \mathbf{C}[j]\}$;
    $E(T) \leftarrow E(T) \cup \{(x_{a-1}, y_b)\}$;
end

Algorithm 2: Partition

Input: A chain graph $G' = (X', Y', E')$ with $|X'| \geq |Y'|$

Output: A part C that is the subgraph of G' constructed by finding the maximum value of f_X and f_Y, using this ratio to choose the vertices for $X(C)$, and adding their neighbours in G' as $Y(C)$

a_i, α_i, b_i, β_i are as defined earlier in this section;
$c, d =$ ComputeRatios(G');
`// construct part` C `based on the values of` c `and` d
if $\frac{\alpha_c}{a_c} \geq \frac{\beta_d}{b_d}$ **then**
  $W = \{x \,|\, deg_{X'}(x) \leq a_c\}$;
  $N(W) = \{y \,|\, y \in N_{G'}(x) \text{ for some } x \in W\}$;
else
  $W = \{y \,|\, deg_{Y'}(y) \leq b_d\}$;
  $N(W) = \{x \,|\, x \in N_{G'}(y) \text{ for some } y \in W\}$;
end
$C = (W, N(W), \{(u, v) \,|\, u, v \in W \cup N(W), (u, v) \in E'\})$;
return C;

Algorithm 3: PartMDST

Input: A chain graph $C_i = (X_i, Y_i, E_i)$ with $|X_i| \geq |Y_i|$

Output: A spanning tree $T_i = (X_i, Y_i, F_i)$ of C_i

foreach $x = x_{p_i}...x_1 \in X_i$ **do**
  choose $y_j \in N_{C_i}(x)$ s.t. $deg_{T_i}(y_j)$ minimized, break ties by lowest index;
  $F_i \leftarrow F_i \cup \{(x, y_j)\}$;
end
choose $y_j \in Y_i$ s.t. $deg_{T_i}(y_j)$ is minimized, break ties by lowest index;
`// we connect the` q_i `components of` T_i`, starting at` y_j
`// we add an edge from` $q_i - 1$ `vertices in` Y_i `to their highest indexed neighbour in` X_i
$\mathbb{C}(v) =$ the component of T_i a vertex v is in;
for $k = j...(j + q_i - 2) \bmod q_i$ **do**
  $l = \max_h$ s.t. $\mathbb{C}(x_h) \neq \mathbb{C}(y_k)$, $deg_{T_i}(x_h) = 1$, and $(x_h, y_k) \in E$;
  $F_i \leftarrow F_i \cup \{(x_l, y_k)\}$;
end
return T_i;

Algorithm 4: ComputeRatios

Input: A chain graph $G' = (X', Y', E')$

Output: The indices c and d that represent the maximum values of $f_X(i, G')$ and $f_Y(i, G')$ respectively.

$index = |X'|$, $c = 0$, $d = 0$, $f_c = 1$, $f_d = 1$;
while $index > 0$ **do**
  while $deg_{G'}(x_{index}) \equiv deg_{G'}(x_{index-1})$ **do**
    $index - -$;
  end
  if $deg_{G'}(y) = 1$ *for some* $y \in N_{G'}(x_{index})$ **then**
    $thisRatio = 0$;
  else
    $thisRatio = \frac{index}{deg_{G'}(x_{index})}$;
  end
  if $f_c \leq thisRatio$ **then**
    $f_c = thisRatio$;
    $c = i$ s.t. $\alpha_i = index$;
  end
  $index - -$;
end
$index = |Y'|$;
while $index > 0$ **do**
  while $deg_{G'}(y_{index}) \equiv deg_{G'}(y_{index-1})$ **do**
    $index - -$;
  end
  if $deg_{G'}(x) = 1$ *for some* $x \in N_{G'}(y_{index})$ **then**
    $thisRatio = 0$;
  else
    $thisRatio = \frac{index}{deg_{G'}(y_{index})}$;
  end
  if $f_d \leq thisRatio$ **then**
    $f_d = thisRatio$;
    $d = i$ s.t. $\beta_i = index$;
  end
  $index - -$;
end
return c, d;

3.2.2 Proof of Correctness

We first give some further explanation of the algorithms presented in the previous subsection. ComputeRatios computes the values of f_X and f_Y to be used by Pariti-ton.

Partition finds the most constraining part of a chain graph, that is the part of the graph that has the largest ratio f_X or f_Y. PartMDST adds edges from the larger, constraining side of a part to the smaller side, pairing each vertex in X_i with a vertex in Y_i, where i is the part index. (We note that X_i and Y_i in a part may be reversed from X and Y in G, depending on how the partition was chosen.) This results in a forest of q_i components. Another $q_i - 1$ edges must be added to finish a spanning tree on the partition.

ChainGraphMDST takes the spanning trees constructed by PartMDST, which result in r components, where r is the number of parts in the final partition of G. The spanning trees for each part are then connected to their adjacent parts, in a way to avoid increasing $\Delta(T)$ unless necessary.

Since the definitions of $f_X(i, G)$ and $f_Y(i, G)$ ensure the part with the maximum ratio does not contain leaves in its smaller side, we can assume that for each part C_i, there are no vertices in $y \in Y_i$ with $deg_{C_i}(y) = 1$.

We now show that these algorithms construct a spanning tree with a particular maximum degree of a chain graph G.

Lemma 3.2. *Immediately following the first loop of the PartMDST algorithm and before the second loop, there exists some* k, $1 \leq k < q_i$, *such that* $deg_{T_i}(y_j) = \lceil \frac{p_i}{q_i} \rceil$ *for all* $1 \leq j \leq k$ *and* $deg_{T_i}(y_j) = \lfloor \frac{p_i}{q_i} \rfloor$ *for all* $k < j \leq q_i$.

Proof. We first show that $deg_{T_i}(y_j) > deg_{T_i}(y_k)$ implies $j < k$. Since G is a chain graph, $N_G(y_j) \supseteq N_G(y_k)$ if $j < k$. If $deg_{T_i}(y_k) > deg_{T_i}(y_j)$, then there was some vertex in $N_G(y_k)$ that was paired to y_k instead of y_j. But since we break ties by lowest index, and $j < k$, then we should have made that vertex adjacent to y_j instead, a contradiction. Therefore $deg_{T_i}(y_j) > deg_{T_i}(y_k)$ implies $j < k$.

By the generalized pigeonhole principle, if we are connecting p_i vertices of X_i to q_i vertices of Y_i, then the maximum degree of a vertex in Y_i is at least $\lceil \frac{p_i}{q_i} \rceil$. In

fact, the maximum degree of a vertex in Y_i is at least the ceiling of the maximum ratio inside this part. Since C_i was chosen as the part that maximized the ratio, this is $\lceil \frac{p_i}{q_i} \rceil$. So the maximum vertex degree in Y_i in T_i after the pairing will be at least $\lceil \frac{p_i}{q_i} \rceil$.

Assume the maximum degree of some vertex in Y_i of T_i is larger than $\lceil \frac{p_i}{q_i} \rceil$. That is, there exists $y \in Y_i$ s.t. $deg_{T_i}(y) \geq \lceil \frac{p_i}{q_i} \rceil + 1 > \lceil \frac{p_i}{q_i} \rceil$. When an edge is added from a vertex $x \in X_i$ to a vertex in Y_i whose degree is already $\lceil \frac{p_i}{q_i} \rceil$, then all other neighbours of x in Y_i have degree at least $\lceil \frac{p_i}{q_i} \rceil$ as well.

When we try to add this new edge, the total degree of $N_{T_i}(x)$ is $\geq |N_{T_i}(x)| * \lceil \frac{p_i}{q_i} \rceil$, and, since we have a vertex $x \in X$ that current does not have a neighbour in T_i, $|N_{T_i}(x)| * \lceil \frac{p_i}{q_i} \rceil < |N_{T_i}(N_{T_i}(x))|$. Equivalently, $\frac{|N_{T_i}(N_{T_i}(x))|}{|N_{T_i}(x)|} > \lceil \frac{p_i}{q_i} \rceil$. Therefore there is a larger ratio in C_i and the part would have been chosen differently. But since this part was chosen, we have a contradiction. Therefore, the maximum degree of any vertex in Y_i of T_i is $\lceil \frac{p_i}{q_i} \rceil$.

Now we show that the minimum degree of any vertex in Y_i of T_i at this point is $\lfloor \frac{p_i}{q_i} \rfloor$. Assume there exists a vertex with degree in T_i less than $\lfloor \frac{p_i}{q_i} \rfloor$. Since we know that higher indexed vertices in Y_i will have degrees no larger than the lower indexed vertices, we can say that $deg_{T_i}(y_{q_i}) < \lfloor \frac{p_i}{q_i} \rfloor$. Let D_{q_i} be the set of vertices in Y_i that have the same degree in G as y_{q_i} (including y_{q_i}). Since G is a chain graph, we know that $N_{C_i}(D_{q_i}) = N_{C_i}(y_{q_i})$.

Since we paired each vertex in X_i of T_i to its lowest degree neighbour in C_i, breaking ties by lowest index, if there is a vertex in D_{q_i} that has degree less than $\lfloor \frac{p_i}{q_i} \rfloor$ in T_i, then the maximum degree in T_i of a vertex in D_{q_i} differs from the minimum degree by exactly one. Since the degree in T_i of y_{q_i} is strictly less than $\lfloor \frac{p_i}{q_i} \rfloor$, then the maximum degree in T_i of a vertex in D_{q_i} is $\leq \lfloor \frac{p_i}{q_i} \rfloor$.

Since there is at least one vertex with degree strictly less than $\lfloor \frac{p_i}{q_i} \rfloor$, we get $|N_{T_i}(D_{q_i})| < |D_{q_i}| * \lfloor \frac{p_i}{q_i} \rfloor$. Therefore, $\frac{|N_{T_i}(D_{q_i})|}{|D_{q_i}|} < \lfloor \frac{p_i}{q_i} \rfloor$.

For simplicity, let $n' = |N_{T_i}(D_{q_i})|$ and $d' = |D_{q_i}|$. We have that $\frac{n'}{d'} < \frac{p_i}{q_i}$. We now show that this implies $\frac{p_i}{q_i} < \frac{p_i - n'}{q_i - d'}$ and so we would have chosen our part to not include D_{q_i} and its neighbourhood in T_i.

From $\frac{n'}{d'} < \frac{p_i}{q_i}$, we get that $-n' > \frac{-d' p_i}{q_i}$.

$$\frac{p_i - n'}{q_i - d'} > \frac{p_i - d'\frac{p_i}{q_i}}{q_i - d'} = \frac{p_i(1 - \frac{d'}{q_i})}{q_i - d'} = \frac{p_i \frac{1}{q_i}(q_i - d')}{q_i - d'} = \frac{p_i}{q_i} \tag{3.1}$$

Therefore, the part C_i would not have been chosen to contain the vertices in D_{q_i} and $N_{T_i}(D_{q_i})$, and so the minimum degree of any vertex in Y_i of T_i is $\lfloor \frac{p_i}{q_i} \rfloor$.

Therefore, the lemma holds. □

Lemma 3.3. *The PartMDST algorithm returns a spanning tree T_i of C_i with*

$$\Delta(T_i) = \begin{cases} \lceil \frac{p_i}{q_i} \rceil & \text{if } \lceil \frac{p_i}{q_i} \rceil \times q_i = p_i + q_i - 1\,, \\ \lceil \frac{p_i}{q_i} \rceil + 1 & \text{otherwise}\,. \end{cases}$$

Proof. We will first show that T_i is a spanning tree of C_i. In PartMDST, T_i is defined to contain the same vertices as C_i. The first loop adds one edge for each $x \in X_i$: p_i edges. The second loop adds $j + q_i - 2 - j + 1 = q_i - 1$ edges, totalling $p_i + q_i - 1$ edges.

After the first loop, there are q_i components in T_i. For each iteration of the second loop, an edge is added from one component to another, decreasing the number of components by one. This happens $q_i - 1$ times, so we get $q_i - (q_i - 1) = 1$ components. Therefore, T_i is connected, and is a spanning tree of C_i.

Now, we will show that $\Delta(T_i)$ is equivalent to the value stated in the lemma. By Lemma 3.2, after the first loop all vertices in T_i have degree $\lceil \frac{p_i}{q_i} \rceil$ or $\lfloor \frac{p_i}{q_i} \rfloor$. So after PartMDST, $\Delta(T_i) \geq \lceil \frac{p_i}{q_i} \rceil$.

We have that $deg_{T_i}(x) \in \{1, 2\}$ for all $x \in X_i$, and since any spanning tree of C_i must have maximum degree ≥ 2, any increase in the maximum degree of T_i must come from a vertex in Y_i.

Case 1: Exactly one vertex in Y_i after the first loop has degree $\lceil \frac{p_i}{q_i} \rceil$, and $\lceil \frac{p_i}{q_i} \rceil \neq \lfloor \frac{p_i}{q_i} \rfloor$. Thus, $p_i \equiv 1 \mod q_i$, and so $\lceil \frac{p_i}{q_i} \rceil \times q_i = p_i + q_i - 1$.

By Lemma 3.2, we know that $deg_{T_i}(y_1) = \lceil \frac{p_i}{q_i} \rceil$, and $deg_{T_i}(y_k) = \lfloor \frac{p_i}{q_i} \rfloor, 1 < k \leq q_i$. The second loop of PartMDST will then start at y_2 and will add one edge incident to each of $y_2, ..., y_{q_i}$. This gives $deg_{T_i}(y) = \lceil \frac{p_i}{q_i} \rceil$ for all $y \in Y_i$. Since we have shown $\Delta(T_i) \geq \lceil \frac{p_i}{q_i} \rceil$, and now that no vertex in Y_i will have degree $< \lceil \frac{p_i}{q_i} \rceil$, we have $\Delta(T_i) = \lceil \frac{p_i}{q_i} \rceil$.

Case 2: More than one vertex in Y_i has degree $\lceil \frac{p_i}{q_i} \rceil$ in T_i, or $\lceil \frac{p_i}{q_i} \rceil = \lfloor \frac{p_i}{q_i} \rfloor$.

We must add $q_i - 1$ edges to the vertices in Y_i. By the generalized pigeonhole principle, we see that we must force at least one vertex with degree $\lceil \frac{p_i}{q_i} \rceil$ to increase to $\lceil \frac{p_i}{q_i} \rceil + 1$. But since C_i is a chain graph, we do not have to add more than 1 edge to each $y \in Y_i$. Therefore, $\Delta(T_i) = \lceil \frac{p_i}{q_i} \rceil + 1$.

By the generalized pigeonhole principle, $\Delta(T_i)$ must be at least $\lceil \frac{p_i}{q_i} \rceil$. In Case 1, we have this value exactly, and in Case 2, we are forced to increase $\Delta(T_i)$ by one in order to get a connected tree.

Therefore, the lemma holds. □

Theorem 3.4. *The ChainGraphMDST algorithm constructs a spanning tree T of a chain graph G with*

$$\Delta(T) = \begin{cases} \lceil \frac{p_1}{q_1} \rceil & \textit{if } p_1 = p, q_1 = q, \lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1\,, \\ \lceil \frac{p_1}{q_1} \rceil + 1 & \textit{otherwise}\,. \end{cases}$$

Proof. Let $T = \text{ChainGraphMDST}(G)$. We need to show that T is a spanning tree of G, and that $\Delta(T) = \lceil \frac{p_1}{q_1} \rceil$ or $\lceil \frac{p_1}{q_1} \rceil + 1$.

First, we will show that T is a spanning tree of G. ChainGraphMDST assigns T to contain the vertices of G, and, if r is the number of parts made by the algorithm, we have $\sum_{i=1}^{r} |E(T_i)| + r - 1 = \sum_{i=1}^{r} (p_i + q_i - 1) + r - 1 = \sum_{i=1}^{r} (p_i + q_i) - r + r - 1 = p + q - 1$ edges in T. Each T_i is the spanning tree constructed for each part C_i of G, and the second loop of the ChainGraphMDST algorithm adds $r - 1$ edges to connect these parts. Note that if $r = 1$, then $C_1 = G$ and T_1 is a spanning tree of G. We now assume $r > 1$.

We know that each T_i is connected, so we just need to show that this second loop connects all T_i, and that the edges used to connect exist in G.

Consider G drawn in the way we described for chain graphs in the beginning of this chapter. See Figure 3.6 as an example. Recall that for some C_i, we may have that $X(C_i) \subset X(G)$ and $Y(C_i) \subset Y(G)$, and for other C_i we may have that $X(C_i) \subset Y(G)$ and $Y(C_i) \subset X(G)$.

This example is partitioned in Figure 3.7, and each T_i has been constructed. The vertices in $X(G)$, $x_1, ..., x_p = Y_1, Y_3, X_4, X_2$ and the vertices in $Y(G)$, $y_1, ..., y_q = Y_2, Y_4, X_3, X_1$, because of how the parts were chosen in this particular chain graph.

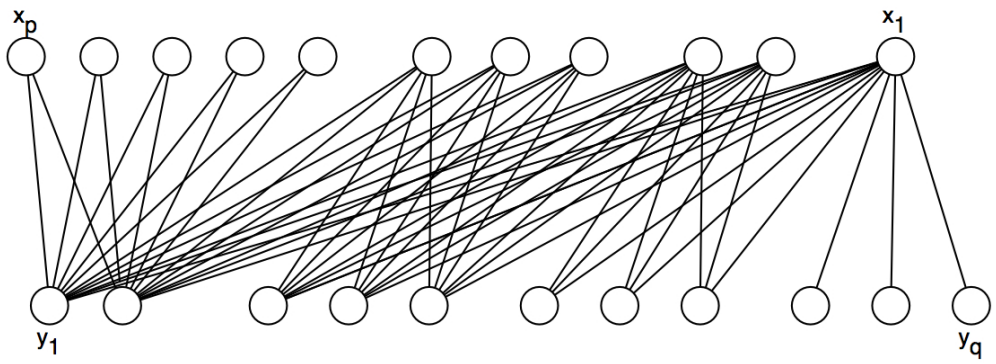

Figure 3.6: A chain graph.

Consider the vertices from each distinct part separately. The edges added between parts are from the highest indexed vertex in $Y(G) \cap C_i$ to the vertex indexed one lower in $X(G)$ than the lowest indexed vertex in $X(G) \cap C_i$. We define an ordering $\mathbf{C}$ to be the ordering of parts from left to right. We denote the part at position j in $\mathbf{C}$ as $\mathbf{C}[j]$. For Figure 3.7, $\mathbf{C} = C_2, C_4, C_3, C_1$, and $\mathbf{C}[1] = C_2$.

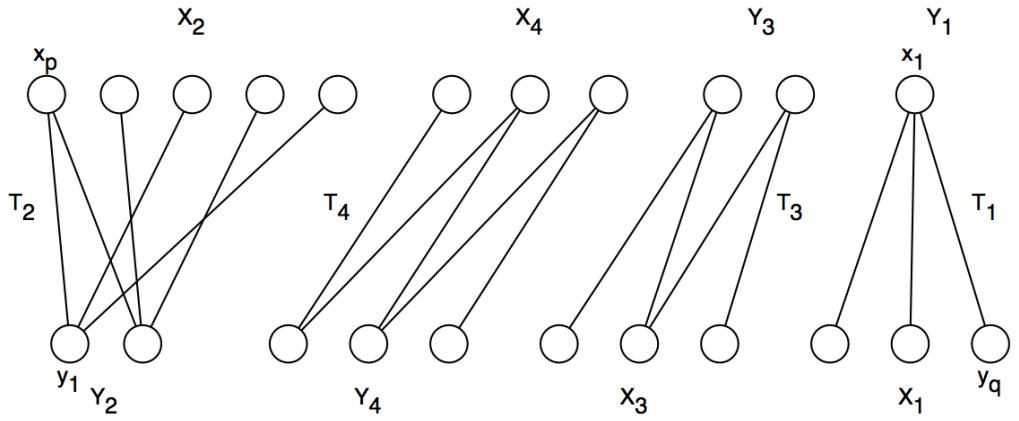

Figure 3.7: The chain graph from Figure 3.6, partitioned and with each T_i constructed as in PartMDST.

In Figure 3.8, we can see that each edge added in the final step (the bold edges) will cross exactly one gap between parts, so each adjacent $\mathbf{C}[j]$ and $\mathbf{C}[j+1]$ will be connected by an edge with one vertex in each part. So T will be connected with $p + q - 1$ edges and therefore is a spanning tree of G.

As defined in ChainMDST, for a given C_i, x_a is the lowest indexed vertex in

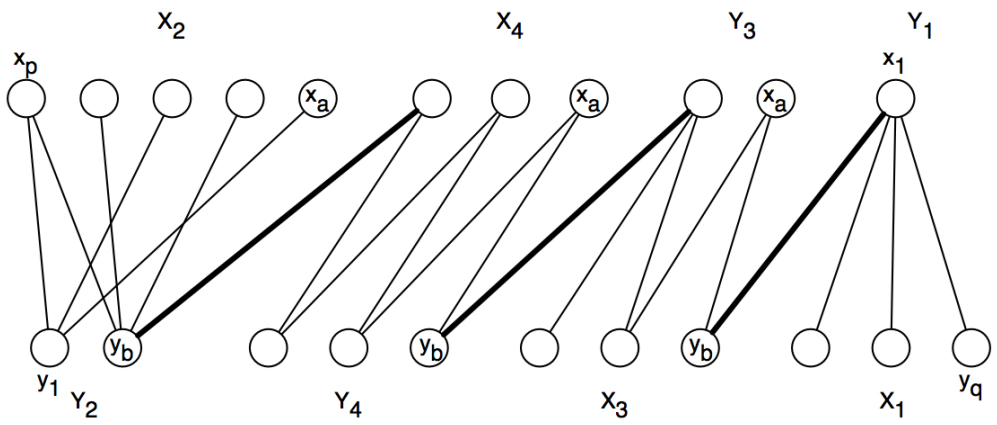

Figure 3.8: The final edges connecting each T_i from Figure 3.7 to form a spanning tree. The vertices x_a and y_b are labeled in each part, and the bold edge is (x_{a-1}, y_b).

$X(G) \cap C_i$ and y_b is the highest indexed vertex in $Y(G) \cap C_i$.

In order to add the edges (x_{a-1}, y_b) to T, we must show this edge is in $E(G)$. Since G is a chain graph, it has the nested neighbourhood ordering, and x_1 is a dominating vertex of Y. Each T_i is connected, and so $(x_{a'}, y_b) \in E(T_i)$ for some $x_{a'} \geq x_a$. Since $x_{a'}, x_1 \in N_G(y_b)$ and the neighbourhood of y_b is consecutive in the ordering, then $x_{a-1} \in N_G(y_b)$. So the edges used to connect exist in G.

Therefore, T is a spanning tree of G. It remains to be shown that $\Delta(T) = \lceil \frac{p_1}{q_1} \rceil$ or $\lceil \frac{p_1}{q_1} \rceil + 1$

By Lemma 3.3, $\Delta(T_i) = \lceil \frac{p_i}{q_i} \rceil$ or $\lceil \frac{p_i}{q_i} \rceil + 1$ for each C_i. Since T_i connects the vertices in X_i to Y_i, we just need to connect these T_i to form a spanning tree T of G with $\Delta(T)$ as one of the values stated in the theorem. Now we will show that the final connecting loop will only increase the maximum $\Delta(T_i)$ if it is necessary.

By how the parts are chosen, we know that $\frac{p_1}{q_1} \geq \frac{p_i}{q_i}$ for all i. By Lemma 3.3, we know that $\Delta(T_1) = \lceil \frac{p_1}{q_1} \rceil$ if $p_1 = p$, $q_1 = q$, $\lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1$, and $\lceil \frac{p_1}{q_1} \rceil + 1$ otherwise. It remains to be shown that $\Delta(T) = \Delta(T_1)$. If $p_1 = p$ and $q_1 = q$, then $C_1 = G$, and so T_1 is a spanning tree of G. We assume $p_1 < p$ and $q_1 < q$.

Case 1: First, assume $\Delta^*_G \geq 3$. Each edge added in the final loop is incident on the highest indexed vertex in $Y(G) \cap \mathbf{C}[j]$ and the vertex indexed one lower than the lowest indexed vertex in $X(G) \cap \mathbf{C}[j]$. This vertex is the highest indexed vertex in $X(G) \cap \mathbf{C}[j+1]$.

We now consider two cases for x_{a-1} in $\mathbf{C}[j+1]$. Let T_l be the spanning tree produced by PartMDST for $\mathbf{C}[j+1]$).

Subcase 1a: $x_{a-1} \in X(\mathbf{C}[j+1])$

As observed in the proof of Lemma 3.3, after PartMDST($\mathbf{C}[j+1]$)), $deg_{T_l}(x) =$ 1 or 2, for all $x \in X(T_l)$. So after this edge is added, $deg_T(x_{a-1}) = 2$ or 3. Since $\Delta^*_G \geq 3$, adding the edge (x_{a-1}, y_b) does not make $deg_T(x_{a-1}) > \Delta(T_1)$.

Subcase 1b: $x_{a-1} \in Y(\mathbf{C}[j+1])$

Let $p_j = |X(\mathbf{C}[j+1])|$ and let $q_j = |Y(\mathbf{C}[j+1])|$. By Lemma 3.2, the maximum degree of a vertex in T_l after the first loop in PartMDST is $\lceil \frac{p_j}{q_j} \rceil$. By Lemma 3.3, the maximum degree of a vertex in T_l after PartMDST is $\lceil \frac{p_j}{q_j} \rceil + 1$. However, since only $q_j - 1$ edges are added, at least one vertex in $Y(\mathbf{C}[j+1])$ has degree in T_l less than $\Delta(T_l)$, or every vertex in $Y(\mathbf{C}[j+1])$ has the same degree.

In the first case, by Lemma 3.2, we know this is the highest indexed vertex in $Y(\mathbf{C}[j+1])$. In this subcase, that vertex is x_{a-1}, and so adding the edge (x_{a-1}, y_b) does not make $deg_T(x_{a-1}) > \Delta(T_1)$. If all vertices in $Y(\mathbf{C}[j+1])$ have the same degree, then a degree increase is required since, by how a part of chosen, $X(\mathbf{C}[j+1])$ has no neighbours in $\mathbf{C}[j]$, and therefore an edge connecting $\mathbf{C}[j]$ to $\mathbf{C}[j+1]$ must be have an edge incident on a vertex in $Y(\mathbf{C}[j+1])$.

Equivalent arguments can be made for the subcases that $y_b \in X(\mathbf{C}[j])$ and $y_b \in Y(\mathbf{C}[j])$.

Case 2: Now assume $\Delta^*_G = 2$. This means that G satisfies one of the conditions from Theorem 3.1.

Subcase 2a: $p = q$ and $deg_G(x_p) = deg_G(y_p) = 1$.

Partitioning G will give two parts, C_1 and C_2, with $X_1 = \{x_p, ..., x_2\}$ and $Y_1 = \{y_1, ..., y_{p-1}\}$, and $X_2 = \{y_p\}$, $Y_2 = \{x_1\}$.

PartMDST(C_1) returns T_1, which will be a path $x_p, y_1, x_{p-1}, y_2, ..., x_2, y_{p-1}$. T_2 will contain only the edge (x_1, y_p). So the degree in T_1 is 1 for x_p, y_{p-1}, x_1, y_p, and 2 for the other vertices. The algorithm will then add the edge (y_{p-1}, x_1), which will not increase $\Delta(T)$. Now $\Delta(T) = \Delta(T_1) = 2$, and T is a spanning tree of G.

Subcase 2b: $p = q$ and at least one of $deg_G(x_q), deg_G(y_p) > 1$ OR $p = q + 1$.

G will be partitioned into just one part, C_1. By Lemma 3.3, T_1 is a spanning

tree of C_1 with the $\Delta(T)$ as defined in this theorem. $C_1 = G$, so $T_1 = T$ and no further edges are added.

Therefore, ChainGraphMDST returns a spanning tree T of G with $\Delta(T) = \lceil \frac{p_1}{q_1} \rceil$ if $p_1 = p$, $q_1 = q$, $\lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1$, and $\lceil \frac{p_1}{q_1} \rceil + 1$ otherwise. Thus, the theorem holds.

□

3.3 Solving the Minimum Degree Spanning Tree Problem on Chain Graphs

Theorem 3.5. *Let $G = (X, Y, E)$ be a chain graph with nested neighbourhood ordering $x_1, ..., x_p$ and $y_1, ..., y_q$. Recall the values $k, l, a_i, \alpha_i, b_i, \beta_i$, as defined in Section 3.2.1, as well as the ratios $f_X(i, G)$ and $f_Y(i, G)$.*

Fix c such that $c \in \{1, ..., k\}$, $f_X(c, G) \geq f_X(i, G)$ for $1 \leq i < c$, and $f_X(c, G) > f_X(i, G)$ for $c < i \leq k$. Fix d such that $d \in \{1, ..., l\}$, $f_Y(d, G) \geq f_Y(i, G)$ for $1 \leq i < d$, and $f_Y(d, G) > f_Y(i, G)$ for $d < i \leq l$.

We now define a part C of G. If $c \geq d$, let $X(C) = \{x \in X \,|deg_G(x) \leq a_c\}$, and $Y(C) = N_G(X(C))$. If $d > c$, let $X(C) = \{y \in Y \,|deg_G(y) \leq b_d\}$, and $Y(C) = N_G(X(C))$. We note that $X(C)$ and $Y(C)$ may correspond to vertices in $Y(G)$ and $X(G)$ respectively.

Let p_1 and q_1 be the sizes of $X(C)$ and $Y(C)$ respectively. Then,

$$\Delta^*_G = \begin{cases} \lceil \frac{p_1}{q_1} \rceil & \text{if } p_1 = p, q_1 = q, \lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1\,, \\ \lceil \frac{p_1}{q_1} \rceil + 1 & \text{otherwise}\,. \end{cases}$$

Proof. By Theorem 3.4, there exists a spanning tree T of a chain graph G such that

$$\Delta(T) = \begin{cases} \lceil \frac{p_1}{q_1} \rceil & \text{if } p_1 = p, q_1 = q, \lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1\,, \\ \lceil \frac{p_1}{q_1} \rceil + 1 & \text{otherwise}\,. \end{cases}$$

Therefore, $\Delta(G) \leq \Delta(T)$.

We now show that, unless $p = p_1$, $q = q_1$ and $\lceil \frac{p}{q} \rceil \times q = p+q-1$, $\Delta^*_G \geq \lceil \frac{p_1}{q_1} \rceil + 1$.

We note that if $p_1 < p$, then we must have $q_1 < q$, otherwise the graph $G - C_1$ is not connected.

If $p > p_1$, and $q > q_1$, then C_1 is a proper subset of G. Connecting each vertex in X_1 to exactly one vertex in Y_1 creates q_1 components. We must now add $q_1 - 1$ edges to connect these components, plus one edge to connect this part to the rest of the graph. Therefore, there are $p_1 + q_1$ edges incident on Y_1, and by the generalized pigeonhole principle, at least one vertex in Y_1 has degree $\geq \lceil \frac{p_1+q_1}{q_1} \rceil = \lceil \frac{p_1}{q_1} \rceil + 1$. Thus in this case, $\Delta^*_G \geq \lceil \frac{p_1}{q_1} \rceil + 1$.

If $p = p_1$ and $q = q_1$, then we must add $p + q - 1$ edges to the vertices in Y to create a spanning tree. Connecting the vertices in X to the vertices in Y gives at least one vertex in Y with degree $\geq \lceil \frac{p}{q} \rceil$. If there is exactly one vertex in Y with degree $\geq \lceil \frac{p}{q} \rceil$ after connecting the vertices in X, then adding the remaining $q - 1$ edges can be done without increasing the degree. This case only occurs if $\lceil \frac{p}{q} \rceil \times q = p + q - 1$. Otherwise, two or more vertices have degree $\geq \lceil \frac{p}{q} \rceil$ after connecting the vertices of X, and adding $q - 1$ more edges will give at least one vertex with degree $\geq \lceil \frac{p}{q} \rceil + 1$. Thus, unless $\lceil \frac{p}{q} \rceil \times q = p + q - 1$, $\Delta^*_G \geq \lceil \frac{p_1}{q_1} \rceil + 1$.

Therefore, this formula gives Δ^*_G for any chain graph G. □

Theorem 3.6. *The ChainGraphMDST algorithm produces an MDST of a given chain graph G in $O(n^2)$ time.*

Proof. By Theorem 3.4, ChainGraphMDST produces a spanning tree T of G such that

$$\Delta(T) = \begin{cases} \lceil \frac{p_1}{q_1} \rceil & \text{if } p_1 = p, q_1 = q, \lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1\,, \\ \lceil \frac{p_1}{q_1} \rceil + 1 & \text{otherwise}\,. \end{cases}$$

By Theorem 3.5,

$$\Delta^*_G = \begin{cases} \lceil \frac{p_1}{q_1} \rceil & \text{if } p_1 = p, q_1 = q, \lceil \frac{p_1}{q_1} \rceil \times q = p + q - 1\,, \\ \lceil \frac{p_1}{q_1} \rceil + 1 & \text{otherwise}\,. \end{cases}$$

Therefore, ChainGraphMDST produces a spanning tree T of G such that $\Delta(T) = \Delta^*_G$.

The algorithm Partition calls ComputeRatios, which checks the ratios for G', which in the worst case has $k = p$ and $l = q$, and so ComputeRatios runs in $O(|G'|) \in O(n)$ time. Partition then adds some number of vertices to the new part, which is $O(n)$, and thus Partition runs in $O(n)$ time. The PartMDST algorithm adds one edge for each vertex in C_i, and so runs in $O(n)$ time.

The ChainGraphMDST algorithm calls Partition and PartMDST once for each part in the partition of G. The upper bound on the number of parts r is q, since at least one vertex from each of X and Y must be in a part. Thus, in the worst case, the first loop of ChainGraphMDST runs in $q * (O(n) + O(n)) \in O(n^2)$ time. The second loop of ChainGraphMDST runs in $O(r)$ time.

Therefore, the ChainGraphMDST algorithm runs in $O(n^2)$ time.

□

Chapter 4

Bipartite Permutation Graphs

We present results related to the minimum degree spanning tree problem for bipartite permutation graphs. First, we give some structural results regarding longest paths in bipartite permutation graphs. We then show that there exists a crossing-free MDST for every connected bipartite permutation graph, and one that contains a longest path.

We combine these to show that for every connected bipartite permutation graph G, there exists a spanning tree T of G such that T is a caterpillar, T contains a longest path of G, T has no edge crossings, and $\Delta(T) = \Delta^*_G$.

Finally, we present a polynomial time algorithm that uses dynamic programming to solve the minimum degree spanning tree problem and the minimum degree spanning tree construction problem on connected bipartite permutation graphs.

Let $G = (X, Y, E)$ be a bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ...y_q$. We will consider G drawn with the vertices of X and Y in rows, sorted as in a strong ordering, as in Figure 4.1. We consider a vertex x_i to be to the *left* of a vertex x_j, $x_i, x_j \in X$, if $i < j$, and to the *right* if $i > j$. The same holds for $y_i, y_j \in Y$.

We define some order on a set of edge crossings in a strongly ordered bipartite permutation graph. Consider a set of edges F such that every edge in the set forms a crossing with at least one other edge. Let $x_{i_1}, x_{i_2}, ..., x_{i_k}$ and $y_{j_1}, y_{j_2}, ..., y_{j_h}$, ordered as in the strong ordering, be the vertices contained in the edges in F. We define the *leftmost* edge crossing as the pair of edges (x_{i_1}, y') and (x', y_{j_1}), where y' is the leftmost neighbour in F of x_{i_1} and x' is the leftmost neighbour in F of y_{j_1}. This

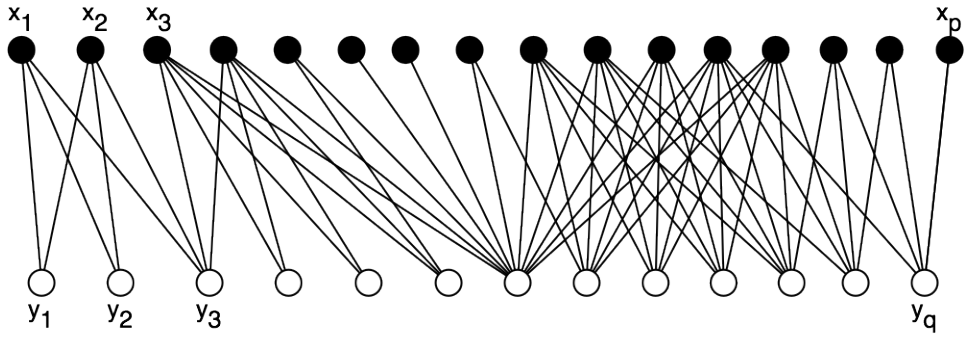

Figure 4.1: A bipartite graph with X and Y ordered as in a strong ordering of G.

pair of edges will form a crossing, since (x_{i_1}, y_{j_1}) will not be contained in F, and so $x_{i_1} < x'$ and $y_{j_1} < y'$.

Since a graph must be connected in order to have a spanning tree, we restrict our attention to connected bipartite permutation graphs in this chapter. Thus, all graphs in this chapter are assumed to be connected and we further assume that each graph has three or more vertices.

4.1 Properties of Longest Paths

In this section, we discuss some previous results about Hamiltonian path and longest path problems on bipartite permutation graphs, and present some new work regarding the structure of these paths.

Theorem 4.1. *[52] Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ...y_q$. Assume without loss of generality that there exists a longest path P in G such that the leftmost endpoint of P is in X. Then there is a longest path P of G with $P = (x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}, ...)$ such that $x_{i_k} < x_{i_{k+1}}$ and $y_{j_k} < y_{j_{k+1}}$ for each k.*

By our definition of zig-zag and Theorems 2.7 and 4.1, we see that if a bipartite permutation graph has a Hamiltonian path, then it has a Hamiltonian path that is a zig-zag, and that every bipartite permutation graph has a longest path that is a zig-zag.

We now present some results regarding the edges (x_1, y_1) and (x_p, y_q) in longest paths of G.

Theorem 4.2. *[1] Let $G = (X, Y, E)$ be a biconvex graph with convex orderings $x_1, ..., x_p$ and $y_1, ..., y_1$. Then G is a bipartite permutation graph if and only if $(x_1, y_1), (x_p, y_q) \in E$.*

Lemma 4.3. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ..., y_1$. Then at least one of x_1, y_1 is in every zig-zag longest path in G.*

Proof. Assume there exists a longest zig-zag path P in G such that neither x_1 or y_1 are in P. Assume without loss of generality $P = (x_i, y_j, ...)$, $i > 1$, $j > 1$. We have stated that we are only dealing with connected graphs, so there exists a path in G from x_i to x_1. If there exists a path in G from x_i to x_1 that contains (x_i, y_k), $y_k \notin P$, then P is not a longest path in G, as it could be extended by adding the edges in the (x_i, x_1)-path. So y_j is on the (x_i, x_1)-path in G. If this path is longer than length 1, replacing (x_i, y_j) with this path would give a path in G longer than P. So $(x_1, y_j) \in E$. But replacing (x_i, y_j) with $(x_1, y_j), (x_1, y_1)$ gives a longer zig-zag path.

Therefore, any zig-zag longest path in G must contain at least one of x_1, y_1.

□

Lemma 4.4. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ..., y_1$. Then there exists a zig-zag longest path P in G such that $(x_1, y_1) \in P$.*

Proof. By Lemma 4.3, at least one of x_1, y_1 is in every zig-zag longest path of G. Assume for all zig-zag longest paths of G, we have exactly one of x_1, y_1 in each path. By Theorem 4.2, we have $(x_1, y_1) \in E$.

Let P be an arbitrary zig-zag longest path of G. Assume without loss of generality that $x_1 \in V(P)$ and $y_1 \notin V(P)$. Since P is a longest path, x_1 must have degree two in P, otherwise we could add (x_1, y_1) to P to extend the path. So $P = (y_j, x_1, ...)$ for some y_j.

Let $P' = (V(P) - \{y_j\} \cup \{y_1\}, E(P) - \{(x_1, y_j)\} \cup \{(x_1, y_1)\})$. But P' is a zig-zag longest path in G. So there exists a longest path in G that contains the edge (x_1, y_1).

□

The proof of this lemma also shows that a path that does not contain (x_1, y_1) can be used to construct a path that contains this edge.

Corollary 4.5. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ..., y_1$. Let P be a zig-zag longest path in G with its leftmost endpoint in one side of the graph (either X or Y) that does not contain the edge (x_1, y_1). Then there exists a zig-zag longest path in G, P', that has its leftmost endpoint in the same side of the graph as P, contains the edge (x_1, y_1), and is equivalent to P in positions $2, ..., r$.*

Let $P = v_1, v_2, ..., v_k$ be a zig-zag longest path of a strongly ordered connected bipartite permutation graph G. P is a *leftmost longest path* if for each v_i, there does not exist a longest path P' such that the vertex at index i in P' is lower indexed than v_i. A bipartite permutation graph has either one or two leftmost longest paths. If it has two, one has its leftmost endpoint in X and the other has its leftmost endpoint in Y.

Lemma 4.6. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, x_2, ..., x_p$, $y_1, y_2, ..., y_q$, and let P be a leftmost longest path in G. Then P contains the edge (x_1, y_1).*

Proof. Let P be a leftmost longest path in G that does not contain the edge (x_1, y_1), and assume without loss of generality that the leftmost endpoint of P is in X. Therefore, P contains some edge (x_i, y_1) such that $i > 1$. So $P = (x_i, y_1, ...)$.

By Lemma 4.4, there exists some zig-zag longest path P' that contains (x_1, y_1). So $P' = (x_1, y_1, ...)$. By the definition of leftmost longest path, we get a contradiction.

Therefore, (x_1, y_1) is in every leftmost longest path of G. □

We have shown that there must exist zig-zag longest paths containing specific vertices. Now, we consider what other vertices must be included in longest paths in bipartite permutation graphs.

Lemma 4.7. *Given a connected bipartite permutation graph $G = (X, Y, E)$ with strong ordering $x_1, ..., x_p$ and $y_1, ...y_q$ and a longest zig-zag path P, if v is not in P, then the neighbourhood in G of v is contained in P, for all $v \in V(G)$.*

Proof. We consider three different cases for $|P|$:

Case 1: $|P| = 2$. Let $P = (x, y)$. If there exists some $v \in G$, $v \notin P$, then there exists a longer path containing two edges. So P is not a longest path.

Case 2: $|P| = 3$. Let $P = (x_{i_1}, y_{j_1}, x_{i_2})$. If there exists some $v \in Y$, $v \notin P$, then since G is connected there exists a (y_{j_1}, v)-path containing at most one of x_{i_1}, x_{i_2}. But then we can extend P, and so it is not a longest path. Assume without loss of generality that $v \in X$, for all $v \notin P$. Then $Y = \{y_{j_1}\}$, and since G is connected $N(v) = \{y_{j_1}\}$. So we have that for all $v \notin P$, $N(v) \subset P$.

Case 3: $|P| > 3$. Assume without loss of generality that $v \in X$. We first consider the case where v is between x_i and $x_j \in P$ in the strong ordering, with $i < j$, and x_i, y_k, x_j are consecutive in P. Assume there exists $u \in N_G(v), u \notin P$.

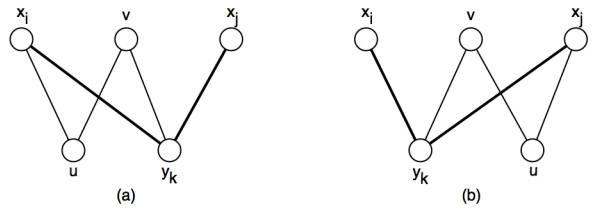

Figure 4.2: The edges in G if a vertex $v \notin P$ has some neighbour $u \notin P$.

In the strong ordering of G, we have either $u < y_k$ as in Figure 4.2(a), or $y_k < u$ as in Figure 4.2(b), and therefore either (x_i, u) or $(x_j, u) \in E(G)$, respectively. From the strong ordering, we also get that $(v, y_k) \in E(G)$. We can replace

$(x_i, y_k, x_j) \in P$ with either (x_i, u, v, y_k, x_j) or (x_i, y_k, v, u, x_j). But P is a longest path. Therefore $N_G(v) \subset P$.

Now consider the case that v is not between two path vertices in the strong ordering. We will say that v is on an *end* of G; that is for X_P being the set of vertices in X that are in P, we have $v < w$, for all $w \in X_P$, or $v > w$, for all $w \in X_P$. Consider the set of vertices $U = \{u | u \in N_G(v), u \notin P$ and u is not between two path vertices in the strong ordering$\}$. U is also on an end of G, and therefore the vertices of U are consecutive in the strong ordering of G. The subgraph of G induced by $\{v\} \cup U \cup \{p\}$, where $p \in P$ with $deg_P(p) = 1$ and the distance in G between v and p is minimized, is a bipartite permutation graph where both X and Y are non-empty and no vertices other than p are in P. Since v and U are on the ends of G, there exists a path from v to p that contains no vertices in P other than p. Adding this path to P gives us a longer path. But P is a longest path, so v cannot have such a subset U in its neighbourhood.

Therefore, for any vertex $v \notin P$, we have that $N_G(v) \subset P$.

□

Lemma 4.8. *Let $G = (X, Y, E)$ be a connected, strongly ordered bipartite permutation graph. For all $v \in G$, if $deg_G(v) \geq 2$, then v is on some longest path in G.*

Proof. Let P be an arbitrary longest zig-zag path of G. Let v be a vertex with $deg_G(v) \geq 2$ that is not in $V(P)$. Assume without loss of generality that $v \in X$. From Lemma 4.7, we have that $N_G(v) \subset V(P)$. Choose two neighbours of v, u_i, u_j such that $u_i, u_j \in P$, $u_i, u_j \in Y$, and no vertex of P is between them in the strong ordering of Y. By the adjacency property of bipartite permutation graphs, we have two vertices that satisfy this requirement. Therefore in P we have u_i, w, u_j consecutively for some $w \in X$. By replacing w in P by v gives another longest path.

Therefore, any vertex v with $deg_G(v) \geq 2$ is on some longest path of G.

□

We now show that the existence of certain zig-zag longest paths in a bipartite

permutation graph give more information about the graph. More specifically, if a bipartite permutation graph G has zig-zag longest paths beginning in both X and Y, this tells us the value of Δ_G^*.

Let P be a zig-zag longest path of a strongly ordered connected bipartite permutation graph G. Let r be the size of a longest path in G, and $P = v_1, v_2, ..., v_r$. We refer to the index i of $v_i \in P$ as the position of v_i in P. Let the *leftmost endpoint* of P be v_1 and the *rightmost endpoint* of P be v_r.

In the remainder of this section, we consider $G = (X, Y, E)$, a strongly ordered bipartite permutation graph that has zig-zag longest paths with leftmost endpoints in both X and Y. Let P_X and P_Y be zig-zag longest paths of G, P_X with leftmost endpoint in X and P_Y with leftmost endpoint in Y. By Lemma 4.4, we can fix these paths to both contain the edge (x_1, y_1). Assume both paths contain this edge, which means that the leftmost endpoint of P_X is x_1, and the leftmost endpoint of P_Y is y_1.

We now introduce some definitions related to P_X and P_Y. Let $pos(v, P)$ be the position in zig-zag path P of vertex v. The leftmost endpoint has position 1, etc. A vertex $v \in P_X \cup P_Y$ satisfies the ***Position Condition*** if one of the following holds:

- $v \in X(P_X \cap P_Y)$ and $pos(v, P_X) < pos(v, P_Y)$, or
- $v \in Y(P_X \cap P_Y)$ and $pos(v, P_Y) < pos(v, P_X)$, or
- $v \notin P_X \cap P_Y$

We define $front(P, v)$ to be the set of vertices in a zig-zag path P that have position less than $pos(v, P)$ and $back(P, v)$ to be the set of vertices in P that have position greater than $pos(v, P)$. A vertex u is to the *left* of $v \in P$ if $u \in front(P, v)$. A vertex u is to the *right* of $v \in P$ if $u \in back(P, v)$. For a vertex $w \notin P$, consider the subpath u_1, v, u_2 in P such that $u_1 < w < u_2$ in the strong ordering of the original bipartite permutation graph. We define vertices to the left and right of w to be those to the left and right of v.

Lemma 4.9. *Let $x_{i_1}, y_{j_1} x_{i_2}, y_{j_2}$ be a subpath of P_X such that the Position Condition holds for these vertices and all vertices to their left in P_X. Then $(x_{i_1}, y_{j_2}) \in E(G)$.*

Proof. Assume that for all four vertex subpaths of P_X beginning in X to the left of $x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}$, there exists an edge in G between the first and fourth vertices in the

subpath. Therefore, $x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}$ is the leftmost such subpath of P_X where we don't know if this edge exists.

By Lemma 4.7, at least one of x_{i_1}, y_{j_1} is in P_Y. We consider the following cases.

Case 1: Assume $x_{i_1}, y_{j_1} \in P_Y$. Let y' be the rightmost neighbour of x_{i_1} in P_Y. If $y' \geq y_{j_2}$, then by the adjacency property, $(x_{i_1}, y_{j_2}) \in G$. Therefore, assume $y' < y_{j_2}$.

If $y' \leq y_{j_1}$, let $a = pos(x_{i_1}, P_X)$ and $b = pos(x_{i_1}, P_Y)$. By our assumption, $a < b$. We have $pos(y', P_Y) = b + 1$ and $pos(y_{j_1}, P_X) = a + 1$. Because y' is to the left of y_{j_1} in P_Y and the Position Condition holds for y_{j_1}, we have that $b+1 \leq a+1$ implies $b \leq a$, a contradiction.

Now assume $y_{j_1} < y' < y_{j_2}$. Since P_X is a zig-zag, we know $y' \notin P_X$. We now show we can build a longer path containing y' and the vertices of P_X to get a contradiction.

If $x_{i_1}, y_{j_1} = x_1, y_1$, let $P' = y_1, x_1, y', x_{i_2}, y_{j_2}, \alpha$, where α is the rest of P_X following y_{j_2}. P' is longer than P_X, but P_X is a longest path and so we have a contradiction.

Otherwise, let $P' = P_X$, and then for each four-vertex subpath $x_{i_3}, y_{j_3}, x_{i_4}, y_{j_4}$ in P' to the left of x_{i_1}, remove the edge (x_{i_4}, y_{j_3}) and add the edge (x_{i_3}, y_{j_4}). Remove (x_{i_2}, y_{j_1}) and add $(x_{i_1}, y'), (x_{i_2}, y')$. Now P' is a longer path than P_X and we get a contradiction.

Case 2: Assume $x_{i_1} \in P_Y$ and $y_{j_1} \notin P_Y$. Let y' be the rightmost neighbour of x_{i_1} in P_Y. If $y' \geq y_{j_2}$, then by the adjacency property, $(x_{i_1}, y_{j_2}) \in G$.

If $y' < y_{j_1}$, we can use the fact that the Position Condition holds for x_{i_1} to get a longer path. We see that $|front(P_Y, x_{i_1})| > |front(P_X, x_{i_1})|$, and so $|back(P_X, x_{i_1})| > |back(P_Y, x_{i_1})|$. Since $y' < y_{j_1}$, and P_X and P_Y are zig-zags, we have that $front(P_Y, x_{i_1}) \cap back(P_X, x_{i_1}) = \emptyset$. Therefore, we can obtain a longer path $P' = front(P_Y, x_{i_1}) \cup \{x_{i_1}\} \cup back(P_X, x_{i_1})$, a contradiction.

Now, assume $y_{j_1} < y' < y_{j_2}$. We know $y' \notin P_X$ and so we can construct a longer path containing y' and the vertices of P_X as in Case 1.

Case 3: Assume $x_{i_1} \notin P_Y$ and $y_{j_1} \in P_Y$. Let x' be the rightmost neighbour of y_{j_1} in P_Y.

Subcase 3.a: Assume $x' < x_{i_1}$ and let y' be the rightmost neighbour of x' in P_Y. If $y' \geq y_{j_2}$, then by the adjacency property, $(x_{i_1}, y_{j_2}) \in G$. Otherwise, $y_{j_1} < y' < y_{j_2}$ and $y' \notin P_X$, and we can construct a longer path P' that contains y' and the vertices of P_X as in Case 1.

Subcase 3.b: If $x' > x_{i_2}$ and $x_{i_2} \notin P_Y$, by Lemma 4.7, we know that we must have $y_{j_2} \in P_Y$. We have that $front(P_X, y_{j_2}) \cap back(P_Y, y_{j_2}) = \emptyset$, and because the Position Condition holds for y_{j_2}, we can build a longer path $P' = front(P_X, y_{j_2}) \cup \{y_{j_2}\} \cup back(P_Y, y_{j_2})$, a contradiction.

Subcase 3.c: If $x' > x_{i_2}$ and $x_{i_2} \in P_Y$, let $a = pos(y_{j_1}, P_X)$ and $b = pos(y_{j_1}, P_Y)$, and so $b < a$ because the Position Condition holds for y_{j_1}. Now $pos(x_{i_2}, P_Y) \leq b - 1$ and $pos(x_{i_2}, P_X) = a + 1$. Because the Position Condition holds for x_{i_2}, we have $a+1 < b-1$, which gives us $a+2 < b$, and so $a+2 < b < a$, a contradiction.

Subcase 3.d: Assume $x' = x_{i_2}$. By the Position Condition, we have $pos(y_{j_1}, P_Y) < pos(y_{j_1}, P_X)$. The edge (x_{i_2}, y_{j_1}) is in both P_X and P_Y, so we get $pos(x_{i_2}, P_Y) = pos(y_{j_1}, P_Y) + 1$ and $pos(x_{i_2}, P_X) = pos(y_{j_1}, P_X) + 1$. This means $pos(x_{i_2}, P_Y) < pos(x_{i_2}, P_X)$, which contradicts our assumption that the Position Condition holds for x_{i_2}.

Subcase 3.e: Assume $x_{i_1} < x' < x_{i_2}$. If the rightmost neighbour y' of x' in P_Y is to the left of y_{j_2}, then $y' \notin P_X$ and we can construct a longer path P' that contains y' and the vertices of P_X as in Case 1. Otherwise, by our assumptions in this case, and because P_X and P_Y are zig-zags, we have that $front(P_X, y_{j_1}) \cap back(P_Y, y_{j_1}) = \emptyset$, and because the Position Condition holds for y_{j_1}, we can build a longer path $P' = front(P_X, y_{j_1}) \cup \{y_{j_1}\} \cup back(P_Y, y_{j_1})$, a contradiction.

Therefore, if the Position Condition holds for a subpath $x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}$ in P_X and all vertices to the left of this subpath, then $(x_{i_1}, y_{j_2}) \in G$.

□

Corollary 4.10. *Consider P_X as defined above in a bipartite permutation graph G. If there exists a vertex $y \in Y(G)$ with $y \notin P_X$, and if the Position Condition is satisfied for all vertices in P_X that are to the left of y, then there exists a path that is longer than P_X. Therefore, when the Position Condition holds in P_X, there are*

no vertices in $Y(G) - Y(P_X)$. Equivalently, when the Position Condition holds in P_Y, there are no vertices in $X(G) - X(P_Y)$.

Proof. The proof of Lemma 4.9, Case 1 shows how if such a vertex in $Y(G) - Y(P_X)$ exists, we can construct a path that is longer than P_X. □

Corollary 4.11. *Using an equivalent argument as in the proof of Lemma 4.9, we can show that for any subpath $y_{j_1}, x_{i_1}, y_{j_2}, x_{i_2} \in P_Y$ such that the Position Condition holds for these vertices and all vertices to their left in P_Y, then $(y_{j_1}, x_{i_2}) \in G$.*

Lemma 4.12. *The Position Condition holds for all vertices in $P_X \cup P_Y$.*

Proof. First, since P_X starts in X and alternates between X and Y along its length, and P_Y starts in Y, we can never have a vertex occupying the same position in both paths. In P_X, all odd vertices are in X and all even vertices are in Y. The reverse is true for P_Y. This means we don't have to worry about equality.

P_X begins with x_1 then y_1, and P_Y begins with y_1, then x_1. We can see the lemma holds here.

Assume that the Position Condition holds in P_X up to some vertex. We first assume this vertex is $x' \in X(P_X)$, and then that it is $y' \in Y(P_X)$.

Let $x' \in X$ be the leftmost vertex in P_X where the Position Condition does not hold. Then we must have $x' \in P_X \cap P_Y$ and $pos(x', P_X) > pos(x', P_Y)$. If $front(P_X, x') \cap back(P_Y, x') = \emptyset$, then we can get a longer path $P' = front(P_X, x') \cup \{x'\} \cup back(P_Y, x')$.

Assume $front(P_X, x') \cap back(P_Y, x') \neq \emptyset$. Let this non-empty set be Q. Let y'' be the rightmost neighbour of x' in P_Y. Since Q is non-empty, y'' must be to the left of the rightmost neighbour of x' in P_X.

Case 1: Assume $y'' \in N_{P_X}(x')$. Here we have two options: y'' can be the left neighbour of x' in P_X or it can be the right neighbour. First, assume y'' is the left neighbour of x' in P_X. By our assumptions, the Position Condition holds for y''. That is, $pos(y'', P_Y) < pos(y'', P_X)$. Because P_X and P_Y share the edge (x', y''), we have that $front(P_X, y'') \cap back(P_Y, y'') = \emptyset$. Therefore, we can build $P' = front(P_X, y'') \cup \{y''\} \cup back(P_Y, y'')$, which is a longer path, a contradiction.

Now assume y'' is the right neighbour of x' in P_X. Then the edge (x', y'') is in both P_X and P_Y and in both cases, y'' is the rightmost neighbour of x'. Since these paths are zig-zags, this shows that $front(P_X, x') \cap back(P_Y, x') = \emptyset$, a contradiction.

Case 2: Assume y'' is to the left of the leftmost neighbour of x' in P_X and $y'' \in P_X$. By our assumptions, the Position Condition holds for y'', so $pos(y'', P_Y) < pos(y'', P_X)$. The rightmost neighbour of y'' in P_Y is to the right of x', so $front(P_X, y'') \cap back(P_Y, y'') = \emptyset$, and so we can build a longer path $P' = front(P_X, y'') \cup \{y''\} \cup back(P_Y, y'')$, a contradiction.

Case 3: Assume $y'' \notin P_X$ and y'' is between the two neighbours of x' in P_X or y'' is to the left of the leftmost neighbour of x' in P_X. Let $x_{i_1}, y_{j_1}, x_{i_2}$ be the subpath of P_X such that y_{j_1} is the nearest vertex in P_X to the left of y''. By Lemma 4.9, for each four vertex subpath of P_X starting in X to the left of x', the edge between the first and fourth vertices is in G. Let $P' = P_X$. For each four vertex subpath of P' starting in X such that all vertices in the subpath are to the left of x_{i_2}, remove the edge connecting the second and third vertices and add an edge connecting the first and fourth. Remove (x_{i_2}, y_{j_1}) and add (x_{i_1}, y''), (x_{i_2}, y''). Now P' is a longer path in G containing y'' and the vertices of P_X, a contradiction.

These cases prove that all vertices in $X(P_X)$ must satisfy the Position Condition.

Now, let $y' \in Y$ be the leftmost vertex in P_X where the Position Condition does not hold. Let $a = pos(y', P_Y)$ and let $b = pos(y', P_X)$. We have $a > b$. Since P_X and P_Y are paths in a bipartite graph that start in X and Y respectively, then a is odd and b is even. The number of vertices in Y in the (y_1, y')-subpath of P_Y is $\lceil \frac{a}{2} \rceil$. The number of vertices in Y in the (x_1, y')-subpath of P_X is $\frac{b}{2}$.

Since a is odd, b is even, and $a > b$, then $\lceil \frac{a}{2} \rceil > \frac{b}{2}$. Therefore, there exists some $\bar{y} \in P_Y$ such that $\bar{y} \notin P_X$ and $y_1 < \bar{y} < y'$. Let $x_{i_1}, y_{j_1}, x_{i_2}$ be the subpath of P_X such that y_{j_1} is the nearest vertex in P_X to the left of $\bar{y}$. By Lemma 4.9, for each four vertex subpath of P_X starting in X to the left of y', the edge between the first and fourth vertices is in G. Let $P' = P_X$. For each four vertex subpath of P' starting in X such that all vertices in the subpath are to the left of x_{i_2}, remove the

edge connection the second and third vertices and add an edge connecting the first and fourth. Remove (x_{i_2}, y_{j_1}) and add $(x_{i_1}, \bar{y}), (x_{i_2}, \bar{y})$. Now P' is a longer path in G containing y'' and the vertices of P_X, a contradiction.

In each case we get a contradiction to our initial assumption that there exists a vertex in P_X where the Position Condition does not hold. Therefore, the Position Condition holds for every vertex in P_X. We can make a symmetrical argument for P_Y, and so therefore the Position Condition holds for all vertices in $P_X \cup P_Y$. □

Corollary 4.13. *P_X contains all vertices in $Y(G)$ and P_Y contains all vertices in $X(G)$.*

Proof. By Corollary 4.10, when the Position Condition holds P_X skips no vertices in Y and P_Y skips no vertices in X. Then by Lemma 4.12, we get that $V(P_X) \supset Y(G)$ and $V(P_Y) \supset X(G)$. □

Lemma 4.14. *If a strongly ordered, connected bipartite permutation graph $G = (X, Y, E)$ has at least one zig-zag longest path that starts in X and another zig-zag longest path that starts in Y, then G has exactly these two zig-zag longest paths, and they contain exactly the same vertices. More specifically, if $P_X = v_1, v_2, ..., v_{r-1}, v_r$, then $P_Y = v_2, v_1, ..., v_r, v_{r-1}$. That is, for each ordered pair of vertices v_i, v_{i+1} in P_X with i odd, then v_{i+1}, v_i is an ordered pair in P_Y.*

Proof. First, we show that r is even, and so P_X has its rightmost endpoint in Y, and P_Y in X.

Assume that r is odd. Since G is bipartite, we have $|X(P_X)| = |Y(P_X)| + 1 = |X(P_Y)| + 1 = |Y(P_Y)|$. Then there exists some x in $X(P_X)$ such that $x \notin P_Y$.

By Corollary 4.13, $X(G) \subset V(P_Y)$.

Therefore, r must be even.

Assume that P_X and P_Y contain the same alternating vertices up to x_{i_1}, y_{j_1}, and after these vertices is the first place they differ. Let $P_X = x_1, y_1, ..., x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}, ...$ and $P_Y = y_1, x_1, ..., y_{j_1}, x_{i_1}, y_{j_3}, x_{i_3}, ...$ Now we want to show that $x_{i_2} = x_{i_3}$ and $y_{j_2} = y_{j_3}$.

First, consider y_{j_3}. By Corollary 4.13, $y_{j_3} \in P_X$. Assume $y_{j_3} \neq y_{j_2}$. Since P_X is a zig-zag, then $y_{j_2} < y_{j_3}$. Since P_Y is a zig-zag, this means $y_{j_2} \notin P_Y$. Since r

is even, if there is some vertex in $Y(P_X - P_Y)$, then there is also some vertex in $Y(P_Y - P_X)$. But this contradicts Corollary 4.13 as P_X cannot skip any vertices in $Y(G)$, and so we must have that $y_{j_2} = y_{j_3}$.

Now consider x_{i_3}. Assume $x_{i_2} \neq x_{i_3}$. By Corollary 4.13, $x_{i_2} \in P_Y$. Since P_Y is a zig-zag, then $x_{i_3} < x_{i_2}$. Since P_X is a zig-zag, this means $x_{i_3} \notin P_X$. Since r is even, if there is some vertex in $X(P_Y - P_X)$, then there is also some vertex in $X(P_X - P_Y)$. But this contradicts Corollary 4.13 as P_Y cannot skip any vertices in $X(G)$. Therefore, we must have that $x_{i_2} = x_{i_3}$.

We assumed that P_X and P_Y differed after x_{i_1}, y_{j_1}. Therefore, the lemma holds by contradiction.

□

Corollary 4.15. *If G has P_X and P_Y, then G has no other zig-zag longest paths and $\Delta^*_G = 2$.*

Proof. By Corollary 4.13, the symmetry of the strong ordering of bipartite permutation graphs, and because we have shown that P_X has its rightmost endpoint in Y and P_Y has its rightmost endpoint in X, we have that P_X and P_Y can not skip any vertex in either $X(G)$ or $Y(G)$. Therefore, they are both Hamiltonian paths. □

4.2 Crossing-Free Minimum Degree Spanning Trees

In the previous section, we looked at longest paths in bipartite permutation graphs that contain no edge crossings. Now, we consider crossing-free spanning trees and MDSTs in bipartite permutation graphs.

Theorem 4.16. *[20] A bipartite graph with a vertex ordering $x_1, ..., x_p$, $y_1, ..., y_q$ contains no edge crossings only if each connected component is a caterpillar.*

Lemma 4.17. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ..., y_1$. Then any spanning tree of G with no edge crossings contains the edges (x_1, y_1) and (x_p, y_q).*

Proof. Assume the edge (x_1, y_1) is not in a crossing-free spanning tree of G. Then x_1 must be adjacent to some vertex y' such that $y' > y_1$, and y_1 must be adjacent to

some vertex $x' > x_1$. The edges (x_1, y'), (x', y_1) are in the spanning tree. But since $x_1 < x'$ and $y_1 < y'$, this is an edge crossing, a contradiction. Therefore, the edge (x_1, y_1) is in the crossing-free spanning tree.

An equivalent argument can be made for (x_p, y_q).

Therefore, (x_1, y_1) and (x_p, y_q) are in any crossing-free spanning tree of a bipartite permutation graph G.

□

Theorem 4.18. *Let $G = (X, Y, E)$ be a strongly ordered, connected bipartite permutation graph. Then G has an MDST with no edge crossings.*

Proof. Let $x_1, ..., x_p$ and $y_1, ...y_q$ be a strong ordering of G. If $\Delta^*_G = 2$, then G has a Hamiltonian path. By Theorem 2.7 G has a Hamiltonian path with no edge crossings.

Assume $\Delta^*_G \geq 3$ and that there does not exist an MDST of G that contains no edge crossings. Let T be an MDST of G such that the leftmost edge crossing in T is furthest to the right over all MDSTs of G.

Let the leftmost edge crossing in T be $(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1})$. This edge crossing will be referred to as the original crossing. T will match one of the following cases, and we will show that there exists an MDST T' of G such that all edge crossings of T' are further right than the original crossing in T.

The figures below show an example of a crossing for each case. Black vertices in the figures have degree Δ^*_G, or have unknown degree. White vertices are known to have degree less than Δ^*_G. Note that each figure only shows the vertices and edges in T and T' that are relevant in each case. Other vertices and edges in the graph are not included. Curved edges represent paths containing at least one vertex not shown.

Case 1: $(x_{i_1}, y_{j_1}) \in T$ and $deg_T(x_{i_2}) < \Delta^*_G$ or $deg_T(y_{j_2}) < \Delta^*_G$, as in Figure 4.3 (a). Assume without loss of generality we have the case that $deg_T(x_{i_2}) < \Delta^*_G$. Let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_2})\} \cup \{(x_{i_2}, y_{j_2})\})$, as in Figure 4.3 (b). Since $(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1})$ is a crossing, we know that $(x_{i_2}, y_{j_2}) \in G$. Removing the edge

from T disconnects the tree into two components, one containing x_{i_1}, y_{j_2} and one containing x_{i_2}. Adding (x_{i_2}, y_{j_2}) reconnects the tree. The degree of each vertex in T' is the same or less than in T, except for x_{i_2}. The degree of x_{i_2} increases by one, but since it was originally less than Δ^*_G in T, $\Delta(T') = \Delta^*_G$. So T' is an MDST of G.

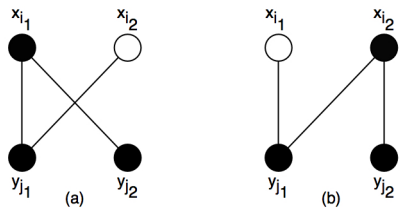

Figure 4.3: An MDST with leftmost crossing further right is constructed for Case 1.

The original edge crossing in T has been removed from T', and any new edge crossings are introduced further to the right, and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

Case 2: $(x_{i_1}, y_{j_1}) \in T$ and $deg_T(x_{i_2}) = deg_T(y_{j_2}) = \Delta^*_G$. Let T_x be the subtree of T containing all vertices v such that $x_{i_2} \in (y_{j_1}, v)$-path in T, and let T_y be the subtree of T containing all vertices v such that $y_{j_2} \in (x_{i_1}, v)$-path in T. Note that $x_{i_2} \in T_x$ and $y_{j_2} \in T_y$. T_x and T_y are vertex disjoint, as $(x_{i_1}, y_{j_1}) \in T$ and there are no cycles in a tree. Since the original crossing is leftmost, we get that $x_{i'} \geq x_{i_2}$ and $y_{j'} \geq y_{j_2}$, for all $x_{i'}, y_{j'} \in T_x \cup T_y$.

We show that there exists an MDST T' of G that has its leftmost edge crossing to the right of the original crossing by finding an edge in G that could connect the disjoint subtrees T_x and T_y so T' remains connected if (x_{i_1}, y_{j_2}) or (x_{i_2}, y_{j_1}) is removed, or by finding a way to decrease the degree of either x_{i_2} or y_{j_2} to give us Case 1.

For the following subcases, we will assume without loss of generality $u_1, u_2 \in X$ and $v_1, v_2 \in Y$, $u_1 < u_2$, $v_2 < v_1$, and that $(u_1, v_1) \in T_x$ and $(u_2, v_2) \in T_y$. In the other cases, a symmetric argument can be made.

Subcase A: There exists a crossing $(u_1, v_1), (u_2, v_2)$ such that (i) $deg_T(u_1) < \Delta^*_G$ and $deg_T(v_2) < \Delta^*_G$, or (ii) $deg_T(u_2) < \Delta^*_G$ and $deg_T(v_1) < \Delta^*_G$. Assume without loss of generality that we have (i), as in Figure 4.4 (a). Let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_2})\} \cup \{(u_1, v_2)\})$, as in Figure 4.4 (b). Since $(u_1, v_1), (u_2, v_2)$ is a crossing, $(u_1, v_2) \in G$. Adding (u_1, v_2) to T creates a cycle containing the edges (x_{i_1}, y_{j_1}), (x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1}) and the paths from x_{i_2} to u_1 and from y_{j_2} to v_2. Removing the edge (x_{i_1}, y_{j_2}) from the cycle makes T' a spanning tree. The only degrees in T' that have increased from T are u_1, v_2, which we defined as being less than Δ^*_G, and so $\Delta(T') = \Delta^*_G$ and T' is an MDST of G.

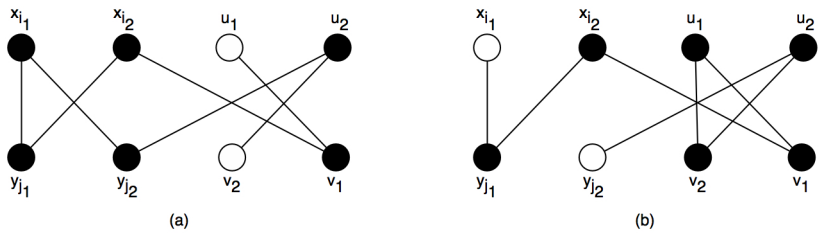

Figure 4.4: An MDST with leftmost crossing further right is constructed for Case 2, Subcase A.

The original edge crossing in T has been removed from T', and any new crossings created are to the right of the original crossing, and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

Subcase B: There exists a crossing $(u_1, v_1), (u_2, v_2)$, such that (i) $deg_T(u_1) = deg_T(v_1) = \Delta^*_G$, $deg_T(u_2) < \Delta^*_G$, and $deg_T(v_2) < \Delta^*_G$, or (ii) $deg_T(u_2) = deg_T(v_2) = \Delta^*_G$, $deg_T(u_1) < \Delta^*_G$, and $deg_T(v_1) < \Delta^*_G$. Assume without loss of generality that we have (i), as Figure 4.5 (a). Let $T' = (X \cup Y, E(T) - \{(u_1, v_1), (x_{i_1}, y_{j_2})\} \cup \{(u_1, v_2), (u_2, v_1)\})$, as in Figure 4.5 (b). Since (u_1, v_1), (u_2, v_2) is a crossing, $(u_1, v_2), (u_2, v_1) \in G$.

Removing (u_1, v_1) disconnects T_x into 2 components, one containing u_1, and the other containing v_1. Adding $(u_1, v_2), (u_2, v_1)$ connects both components of T_x to T_y and creates a cycle containing the paths in T_x and T_y from the original crossing to the $(u_1, v_1), (u_2, v_2)$ crossing. Removing the edge (x_{i_1}, y_{j_2}) from the cycle makes

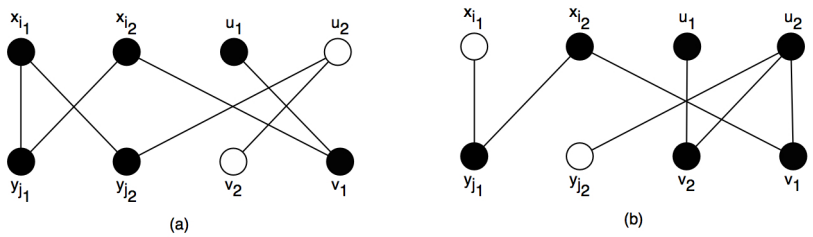

Figure 4.5: An MDST with leftmost crossing further right is constructed for Case 2, Subcase B.

T' a spanning tree. Only u_2, v_2 have higher degree in T' than T, but since they were defined to have degree less than Δ^*_G, $\Delta(T') = \Delta^*_G$ and so T' is an MDST of G.

The original edge crossing in T has been removed from T' and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

Subcase C: For all crossings $(u_1, v_1), (u_2, v_2)$, we have that $deg_T(u_1) = deg_T(u_2) = deg_T(v_1) = deg_T(v_2) = \Delta^*_G$. Consider T_x and T_y as previously defined. Since T is a tree, both T_x, and T_y must have at least one leaf. Let v' be a leaf of T_x and, by the definition of this case, we know its incident edge (u', v') is contained in no edge crossing. Therefore, all vertices in T_y must be to the left of u', v' in the ordering of G. Let v'' be a leaf of T_y. We know $v'' < v'$, and so $(u'', v'') \in T_y$ must be crossed by some edge in T_x. But this gives a crossing containing a vertex with degree less than Δ^*_G. Therefore, this case does not occur.

Subcase D: There does not exist a crossing as in Subcase A or B. Therefore, there exists some edge crossing $(u_1, v_1), (u_2, v_2)$ with either (i) $deg_T(u_1) = deg_T(u_2) = \Delta^*_G$ and at least one of $deg_T(v_1), deg_T(v_2) < \Delta^*_G$, or (ii) $deg_T(v_1) = deg_T(v_2) = \Delta^*_G$ and at least one of $deg_T(u_1), deg_T(u_2) < \Delta^*_G$. Assume without loss of generality that we have (i).

Fix $(u_1, v_1), (u_2, v_2)$ to be the crossing *nearest* to x_{i_2} and y_{j_2} that meets the criteria of either Subcase D.1 or D.2 as stated below. That is, there does not exist a crossing that meets either set of criteria such that $dist_T(u_1, x_{i_2})$ or $dist_T(v_2, y_{j_2})$ is less than the chosen crossing.

We show that such a crossing exists: Since we do not have Subcase A or B, the degree constraints for either Subcase D.1 or D.2 must be satisfied. It remains to show that the restriction on vertices in paths will hold. Assume we have a crossing as in Subcase D.1, but $v_1 \in (x_{i_2}, u_1)$-path. But then since $x_{i_2} < u_1$, there must be some edge on the (x_{i_2}, v_1)-path that crosses (u_2, v_2). We would choose that edge instead of (u_1, v_1). Next, assume we have a crossing as in Subcase D.2, but $v_2 \notin (y_{j_2}, u_2)$-path. But then since $y_{j_2} < v_2$, there must be some edge on the (y_{j_2}, u_2)-path that crosses (u_1, v_1). We would choose that edge instead of (u_2, v_2).

Subcase D.1: The nearest crossing in Subcase D has $deg_T(u_1) = deg_T(u_2) = \Delta^*_G$, $deg_T(v_2) < \Delta^*_G$, and $v_1 \notin (x_{i_2}, u_1)$-path, as in Figure 4.6 (a).

If $u_1 \neq x_{i_2}$, take $(u_1, v_3) \in (x_{i_2}, u_1)$-path. Let $T' = (X \cup Y, E(T) - \{(u_1, v_3)\} \cup \{(u_1, v_2)\})$. We know $(u_1, v_2) \in G$ because $(u_1, v_1), (u_2, v_2)$ is a crossing. Removing (u_1, v_3) disconnects the spanning tree by splitting T_x into 2 components, one containing x_{i_2} and one containing u_1. By adding (u_1, v_2), all vertices in T_x that had u_1 on their path to x_{i_2} are now part of T_y. The degree of u_1 remains the same, the degree of v_3 decreases, and the degree of v_2 increases by 1. But in this case we chose $deg_T(v_2) < \Delta^*_G$. So $\Delta(T') = \Delta^*_G$. So T' is an MDST of G, and T' has a nearer valid crossing for this subcase than T, or it may now match Subcase A or B. A nearer crossing for this subcase is shown in Figure 4.6 (b).

If T' matches Subcase A or Subcase B, we get a contradiction. If T' has a further left crossing as in Subcase D, there exists T'' that can be constructed from T' that either matches Subcase A or B, or matches Subcase D with a further left crossing. Since G is finite, if these transformations of T' never produce another MDST that matches Subcase A or B, eventually there will be an MDST with degree of x_{i_2} or y_{j_2} less than Δ^*_G, as in Case 1. In any case, we can get a new MDST with leftmost edge crossing further to the left of T and a contradiction.

Otherwise, if $u_1 = x_{i_2}$, let $T' = (X \cup Y, E(T) - \{(x_{i_2}, y_{j_1})\} \cup \{(x_{i_2}, v_2)\}$. We know $(x_{i_2}, v_2) \in G$ because $(u_1, v_1), (u_2, v_2)$ is a crossing. Removing (x_{i_2}, y_{j_1}) disconnects T_x from the original crossing, and adding (x_{i_2}, v_2) reconnects the T_x component to T_y. So T' is connected. Since $deg_T(v_2) < \Delta^*_G$, we have that $\Delta(T') =$

Δ^*_G and so T' is an MDST of G.

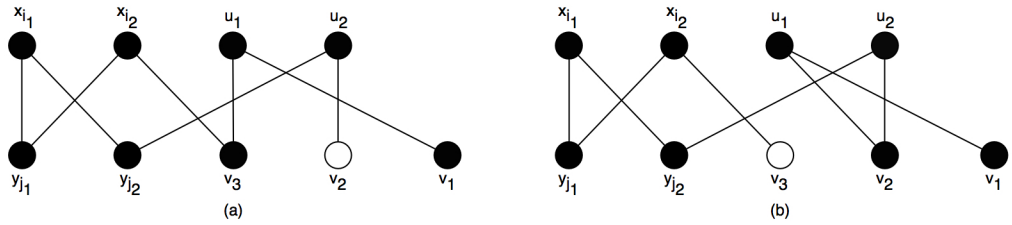

Figure 4.6: A nearer crossing is created in Case 2, Subcase D.

The original edge crossing in T has been removed from T' and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

Subcase D.2: The nearest crossing in D has $deg_T(u_1) = deg_T(u_2) = deg_T(v_2) = \Delta^*_G$ and $deg_T(v_1) < \Delta^*_G$, and $v_2 \in (y_{j_2}, u_2)$-path, as in Figure 4.7 (a).

Let $T' = (X \cup Y, E(T) - \{(u_2, v_2)\} \cup \{(u_2, v_1)\})$, as in Figure 4.7 (b). We know $(u_2, v_1) \in G$ because $(u_1, v_1), (u_2, v_2)$ is a crossing. Removing (u_2, v_2) disconnects T_y into two components, one containing v_2, y_{j_2} and the original crossing, and the other containing u_2, and decreases the degree of u_2 and v_2 by 1. Adding (u_2, v_1) connects the component not connected to the original crossing to T_x, and since $deg_T(v_1) < \Delta^*_G$, $\Delta(T') = \Delta^*_G$. So T' is an MDST of G.

In this case, if $v_2 = y_{j_2}$, then T' matches Case 1, and we get a contradiction.

Otherwise, if $v_2 \neq y_{j_2}$, we have created a new crossing between T_x and T_y that is nearer to the original crossing and contains a vertex with degree less than Δ^*_G. This crossing is the edge on the (y_{j_2}, v_2)-path that contains v_2, and an edge on the (x_{i_2}, v_1)-path, and meets the criteria for one of Subcases A, B, or D.1.

If T' matches Subcase A or Subcase B, we get a contradiction. If T' has a further left crossing as in Subcase D, there exists T'' that can be constructed from T' that either matches Subcase A or B, or matches Subcase D with a further left crossing. Since G is finite, if these transformations of T' never produce another MDST that matches Subcase A or B, eventually there will be an MDST with degree of x_{i_2} or y_{j_2} less than Δ^*_G, as in Case 1. In any case, we can get a new MDST with leftmost edge crossing further to the left of T and a contradiction.

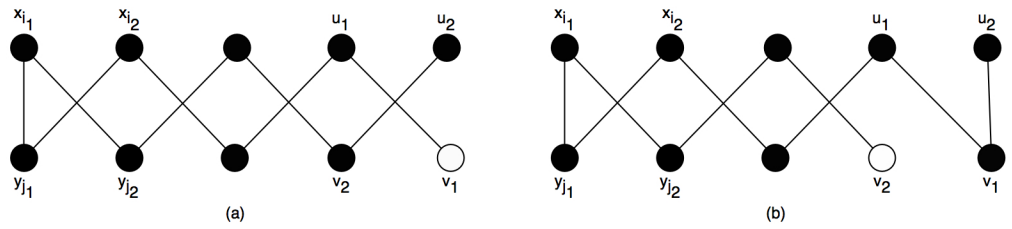

Figure 4.7: A nearer crossing is created for Case 2, Subcase D.

Case 3: $(x_{i_1}, y_{j_1}) \notin T$ and $deg_T(x_{i_1}) < \Delta^*_G$ or $deg_T(y_{j_1}) < \Delta^*_G$ as in Figure 4.8 (a). We have selected the leftmost crossing in T, and so we know that the path between x_{i_1} and y_{j_1} does not contain a vertex v such that $v < x_{i_1}$ or $v < y_{j_1}$.

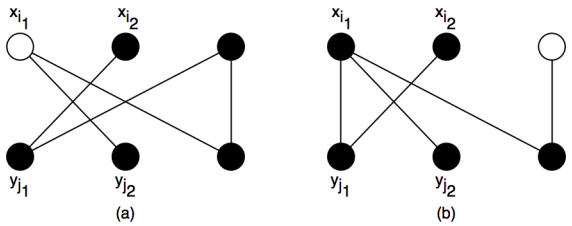

Figure 4.8: Case 3 can be reduced to Case 1 or Case 2, depending on the degrees of x_{i_2} and y_{j_2}.

Let (u, v) be the edge on the (x_{i_1}, y_{j_1}) path in T containing whichever of x_{i_1}, y_{j_1} has the highest degree in T. Let $T' = (X \cup Y, E(T) - \{(u, v)\} \cup \{(x_{i_1}, y_{j_1})\})$ as in Figure 4.8. The removal of (u, v) disconnects the tree into 2 components, one containing x_{i_1} and one containing y_{j_1}. Adding edge (x_{i_1}, y_{j_1}) connects the components and creates a new spanning tree. After removing (u, v) we know that both $deg_T(x_{i_1}) < \Delta^*_G$ and $deg_T(y_{j_1}) < \Delta^*_G$. Therefore $\Delta(T'') = \Delta^*_G$ and T' is an MDST of G. T' contains (x_{i_1}, y_{j_1}) and so matches either Case 1 or Case 2. In both these cases, we get a contradiction.

Case 4: $(x_{i_1}, y_{j_1}) \notin T$ and $deg_T(x_{i_1}) = deg_T(y_{j_1}) = \Delta^*_G$. Since we have selected the leftmost edge crossing, only one of x_{i_1}, y_{j_1} can have neighbours that are to the left of the crossing. If both x_{i_1} and y_{j_1} have neighbours y' and x' respectively, with $y' < y_{j_1}$ and $x' < x_{i_1}$, then this is a crossing that is further left than the

original crossing. Assume without loss of generality that x_{i_1} is that vertex, and so $\forall\, v \in N_T(y_{j_1})$, $v \geq x_{i_2}$.

Subcase A: $(x_{i_2}, y_{j_2}) \notin T$ and $x_{i_2} \notin (y_{j_1}, y_{j_2})$-path in T, as in Figure 4.9 (a). Let $T' = (X \cup Y, E(T) - \{(x_{i_2}, y_{j_1}), (x_{i_1}, y_{j_2})\} \cup \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})\})$ as in Figure 4.9 (b).

The removal of (x_{i_2}, y_{j_1}) divides the tree into 2 components, one containing y_{j_1} and y_{j_2} and one containing x_{i_2}, and reduces the degree of y_{j_1} and x_{i_2} by 1. Removing (x_{i_1}, y_{j_2}) further disconnects the tree into 3 components, the first containing x_{i_1}, the second y_{j_1} and y_{j_2}, and the third x_{i_2}, and decreases the degree of x_{i_1} and y_{j_2}. Adding (x_{i_2}, y_{j_2}) connects the first and second components, and adding (x_{i_1}, y_{j_1}) connects the second and third, and since all these vertices had an edge removed, $\Delta(T') = \Delta^*_G$. So T' is a minimum degree spanning tree.

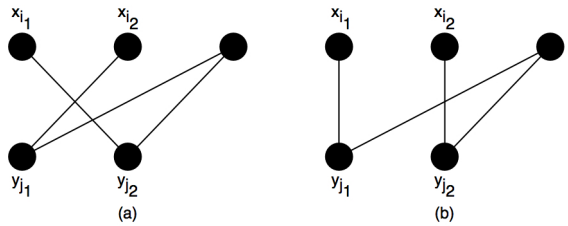

Figure 4.9: A new MDST with leftmost edge crossing further right can be constructed in Case 4, Subcase A.

The original edge crossing in T has been removed from T' and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

Subcase B: $(x_{i_2}, y_{j_2}) \in T$. Let T_x be the subtree of T containing all vertices v such that $x_{i_2} \in (y_{j_1}, v)$-path, and let T_y be the subtree of T containing all vertices v such that for some $u \in N_T(y_{j_1}) - \{x_{i_2}\}$, $u \in (y_{j_1}, v)$-path. So $x_{i_2}, y_{j_2} \in T_x$ but not T_y, and $y_{j_1} \in T_y$.

We proceed with T_x and T_y as in the subcases of Case 2, and construct T''. We have two possibilities for the construction of T': first, in Case 2 we are able to find an edge to add between this T_x and T_y that creates a cycle in T and does not

increase $\Delta(T)$. Let $T' = (X \cup Y, E(T) - \{(x_{i_2}, y_{j_1})\} \cup \{e\})$, where e is the edge that can be added between T_x and T_y. An example of this is shown in Figure 4.10. Adding e creates a cycle that contains (x_{i_2}, y_{j_1}) and so removing that edge makes T' a spanning tree of G. Since e was chosen as in Case 2, $\Delta(T') = \Delta^*_G$. So T' is an MDST of G.

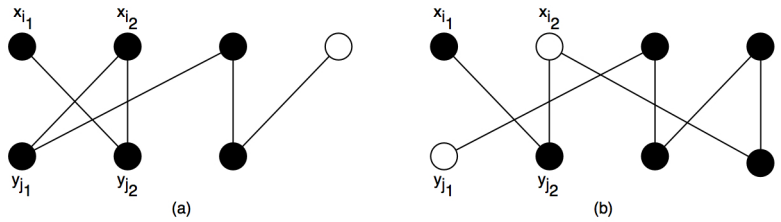

Figure 4.10: A new MDST with leftmost edge crossing further right can be constructed in Case 4, Subcase B if an edge can be added between T_x and T_y.

The original edge crossing in T has been removed from T' and so all edge crossings in T' are further right than the original crossing in T. So we get a contradiction.

The second possibility is that between T_x and T_y, every pair of edge crossings contains only one vertex with degree less than Δ^*_G, as in Subcase 2.D. In Subcase 2.D, we showed there either exists an MDST with a crossing as in Subcase 2.A or Subcase 2.B, and we can obtain a contradiction, or there exists an MDST T' with the degree of the leftmost vertex in T'_x or T'_y less than Δ^*_G. In the latter case, we have either $deg_{T'}(x_{i_2}) = deg_T(x_{i_2}) - 1 < \Delta^*_G$ or $deg_{T'}(y_{j_1}) = deg_T(y_{j_1}) - 1 < \Delta^*_G$. Note that instead of checking for $v_2 = y_{j_2}$ in Subcase D.2, in this case we want to check for $v_2 = y_{j_1}$.

First assume $deg_{T'}(y_{j_1}) = deg_T(y_{j_1}) - 1 < \Delta^*_G$. We have reduced T' to Case 3 as in Figure 4.11, and so we know we have a contradiction.

Now assume $deg_{T'}(x_{i_2}) = deg_T(x_{i_2}) - 1 < \Delta^*_G$, as in Figure 4.12. The degree of x_{i_2} is decreased in T' by moving some vertex from $N_T(x_{i_2}) \cap T_x$ to T_y. Let y_{j_3} be that vertex, with x_{i_3} as its new neighbour in T'_y. We have that $x_{i_2} < x_{i_3}$ since the original crossing is leftmost.

Let $T'' = (X \cup Y, E(T') - \{(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1}), (x_{i_3}, y_{j_3})\} \cup \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_3}), (x_{i_3}, y_{j_2})\})$. We know that $(x_{i_3}, y_{j_2}) \in G$ because $(y_{j_1}, x_{i_3}), (x_{i_2}, y_{j_2})$

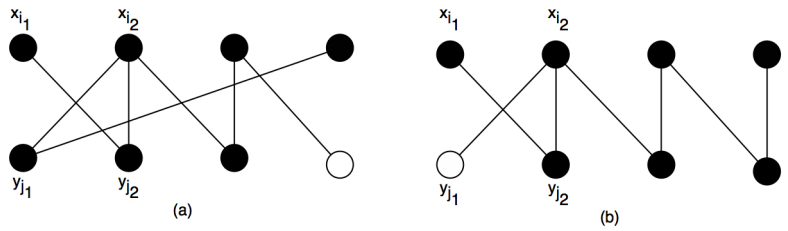

Figure 4.11: In Case 4, Subcase B, if the degree of y_{j_1} is decreased, we have reduced this case to Case 3.

is a crossing. Removing (x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1}), (x_{i_3}, y_{j_3}) creates 4 components of T', the first containing x_{i_1}, the second x_{i_2}, y_{j_2}, the third x_{i_3}, y_{j_1}, and the fourth y_{j_3}. (x_{i_1}, y_{j_1}) connects the first and third components. (x_{i_2}, y_{j_3}) connects the second and fourth, and (x_{i_3}, y_{j_2}) connects these 2 remaining components. Therefore, T'' is a spanning tree of G. Each vertex that has one edge added to it in T'' also has one edge removed from it that was in T'. Therefore $\Delta(T'') = \Delta^*_G$.

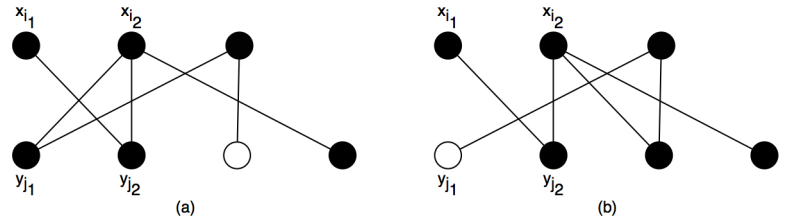

Figure 4.12: A new MDST with leftmost edge crossing further right can be constructed in Case 4, Subcase B if the degree of x_{i_2} can be decreased.

Removing (x_{i_1}, y_{j_2}) and (x_{i_2}, y_{j_1}) eliminates the original crossing and only (x_{i_1}, y_{j_1}) is added back between those four vertices. The original edge crossing in T has been removed from T'' and so all edge crossings in T'' are further right that the original crossing in T. So we get a contradiction.

Subcase C: $(x_{i_2}, y_{j_2}) \notin T$ and $x_{i_2} \in (y_{j_1}, y_{j_2})$-path in T. Let T_x be the subtree of T containing all vertices v such that $x_{i_2} \in (y_{j_1}, v)$-path and $v \notin (y_{j_1}, y_{j_2})$-path, and also containing x_{i_2}. Let T_y be the subtree of T containing all vertices v such that for some $u \in N_T(y_{j_1}) - \{x_{i_2}\}$, $u \in (y_{j_1}, v)$-path, plus y_{j_1}. So x_{i_2} is part of T_x but not T_y, and $y_{j_1} \in T_y$.

We proceed with T_x and T_y as in the subcases of Case 2, and construct T''. We will have one of two possibilities: first, between this T_x and T_y is a crossing as in Subcase 2.A or Subcase 2.B, and an edge can be added as shown in Figure 4.13 between T_x and T_y as in these subcases. Let e be the edge. Let $T' = (X \cup Y, E(T) - \{(x_{i_2}, y_{j_1})\} \cup \{e\})$. Adding e connects T_x to T_y and removing (x_{i_2}, y_{j_1}) disconnects T_x from the original crossing. So T' is a spanning tree. Since e was selected as in Subcase 2.A or 2.B, $\Delta(T') = \Delta^*_G$. The original edge crossing is also removed, so T' is an MDST of G. The original edge crossing in T has been removed from T' and so all edge crossings in T' are further right that the original crossing in T. So we get a contradiction.

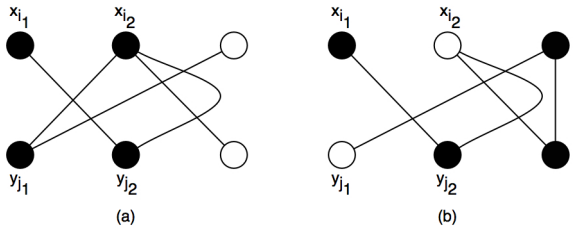

Figure 4.13: A new MDST with leftmost edge crossing further right can be constructed in Case 4, Subcase C if an edge can be added between T_x and T_y.

Alternatively, all crossings between T_x and T_y are as in Subcase 2.D. As we have seen in Case 2, in this case either we have an MDST that satisfies Subcase 2.A or 2.B, and can obtain a contradiction as above, or we have an MDST T'' with $deg_{T''}(x_{i_2}) = deg_T(x_{i_2}) - 1 < \Delta^*_G$ or $deg_{T''}(y_{j_1}) = deg_T(y_{j_1}) - 1 < \Delta^*_G$.

Assume first that $deg_{T''}(x_{i_2}) < \Delta^*_G$. Fix x_{i_3} such that $(x_{i_3}, y_{j_2}) \in (x_{i_2}, y_{j_2})$-path. By the definition of this subcase, $x_{i_2} \neq x_{i_3}$. Let $T' = (X \cup Y, E(T'') - \{(x_{i_3}, y_{j_2})\} \cup \{(x_{i_2}, y_{j_2})\})$. Removing (x_{i_3}, y_{j_2}) disconnects T'' into a component containing x_{i_1} and y_{j_2} and a component containing y_{j_1}, x_{i_2}, and the rest of the vertices on the former (x_{i_2}, y_{j_2})-path. Adding (x_{i_2}, y_{j_2}) connects these two components. Since we had that $deg_{T''}(x_{i_2}) < \Delta^*_G$, and we removed an edge from y_{j_2}, then $\Delta(T') = \Delta^*_G$. So T' is an MDST of G. The original edge crossing still exists, but $(x_{i_2}, y_{j_2}) \in T'$, so we have reduced this to Subcase 4.B, as is shown in Figure 4.14. In Subcase 4.B, we get a contradiction.

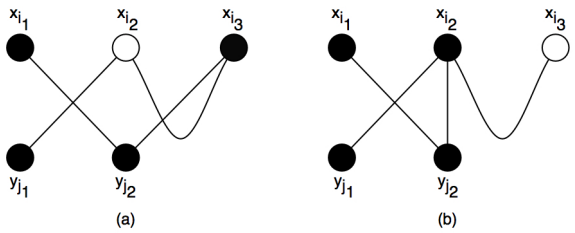

Figure 4.14: If the degree of x_{i_2} can be decreased in Case 4, Subcase C, then this case can be reduced to Case 4, Subcase B.

Now assume $deg_{T''}(y_{j_1}) = deg_T(y_{j_1}) - 1 < \Delta^*_G$. We have reduced T'' to Case 3, in the same way shown for Subcase B in Figure 4.11, and as in Case 3 we get a contradiction.

We have shown that each case gives a contradiction. Therefore, there exists an MDST for any bipartite permutation graph G that has no edge crossings.

□

Corollary 4.19. *Let G be a strongly ordered connected bipartite permutation graph. Then there exists an MDST of G that is a caterpillar with no edge crossings.*

Proof. By Theorem 4.18, there exists an MDST T of G that has no edge crossings. By Theorem 4.16, T is a caterpillar.

□

4.3 Longest Paths in Minimum Degree Spanning Trees

In previous sections, we have seen that, given a bipartite permutation graph G, there exists a longest path in G that contains no edge crossings, and an MDST of G that contains no edge crossings. Now, we explore MDSTs of bipartite permutation graphs that contains longest paths.

Theorem 4.20. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p$ and $y_1, ...y_q$ and let T be an MDST of G that is a caterpillar. Then G has an MDST that is a caterpillar in which no edges cross, and with spine of size greater than or equal to the size of the spine of T.*

Proof. Let T be an MDST of G that is a caterpillar with maximum degree $\Delta(T)$ and let T_S be the spine of T. That is, T_S is the set of all vertices in T that are not leaves.

The following steps allow us to transform an MDST of G that is a caterpillar into an MDST of G that is a caterpillar with no edge crossings without decreasing the size of the spine. Figures accompany each step, with spine edges in bold and leaf edges unbolded. Vertices that are leaves prior to each transformation are black, and spine vertices are white. Each figure is an example for each case, and there are many symmetric cases not shown.

Step 1: Assume T_S contains edge crossings, as in Figure 4.15 (a). We can remove all such crossings of two edges (x_{i_1}, y_{j_1}) and $(x_{i_2}, y_{j_2}) \in V(T_S)$, or *spine edges*, and create a new MDST T that contains no crossings in T_S. This new T_S will be a zig-zag.

Consider the subgraph G_{T_S} of G, which is the subgraph induced by the vertices in T_S plus two leaf vertices, one adjacent to each endpoint of T_S. G_{T_S} is a bipartite permutation graph and T_S is a Hamiltonian path in it. From [51] we know that if we have a Hamiltonian path, we have a zig-zag Hamiltonian path. By definition, a zig-zag Hamiltonian path has no edge crossings. Replace the edges of T_S with the edges a zig-zag Hamiltonian path of G_{T_S}. An example of this uncrossing is given in Figure 4.15 (b).

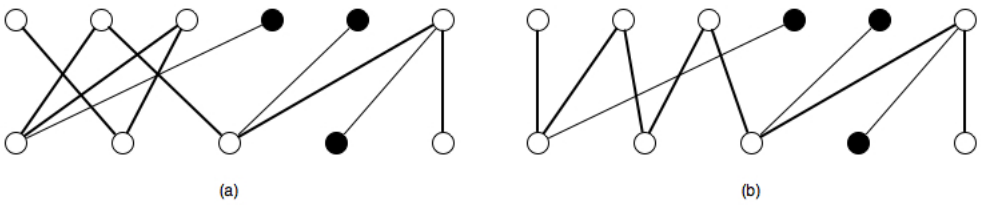

Figure 4.15: Edge crossings in T_S can be removed by constructing a zig-zag Hamiltonian path on the subgraph of G induced by T_S.

The degrees of vertices in T_S either remain the same, decrease (if a degree 2 vertex becomes a leaf), or increase by 1. A vertex whose degree increases will go from 1 to 2 in T_S, but this means it was a leaf initially and so had no other neigh-

bours outside of T_S. Therefore, $\Delta(T) = \Delta^*_G$.

Step 2: If T contains an edge crossing containing two leaf vertices, we construct a new MDST T such that there are fewer crossings of this type. That is, we remove crossings involving two *leaf edges*, which are edges that contain a vertex not in $V(T_S)$. We consider crossings with one leaf in X and one leaf in Y separate from crossings where both leaves are in either X or Y.

Step 2a: Remove all crossings involving edges (x_{i_1}, y_{j_1}) and (x_{i_2}, y_{j_2}) where $x_{i_1}, y_{j_2} \in V(T_S)$ and $x_{i_2}, y_{j_1} \notin V(T_S)$. We show how to construct a new caterpillar T' from T such that this crossing is removed and the size of T'_S is greater than the size of T_S:

Assume without loss of generality that $x_{i_1} < x_{i_2}$ and $y_{j_2} < y_{j_1}$. Consider $T \cup (x_{i_2}, y_{j_1})$, as in Figures 4.16 and 4.17.

Case 1: There exists some edge in T_S that forms a crossing with (x_{i_2}, y_{j_1}).

If only one edge in T_S crosses (x_{i_2}, y_{j_1}), let that edge be (x', y'). Assume without loss of generality that $x' < x_{i_2}$, $y' > y_{j_1}$. Let $T' = (X \cup Y,\, E(T) - \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), (x', y')\} \cup \{(x', y_{j_1}), (x_{i_2}, y_{j_1}), (x_{i_2}, y')\}$. The edges removed disconnect T into four components, one containing x', one containing y', and the two isolated vertices y_{j_1} and x_{i_2}. Adding (x', y_{j_1}) connects the first component and one isolated vertex, (x_{i_2}, y') connects the second component to the other isolated vertex, and(x_{i_2}, y_{j_1}) connects these two components. We have also now added y_{j_1} and x_{i_2} to T'_S, and so $|T'_S| = |T_S| + 2$. T' has two fewer leaf vertices than T, and since only spine edges were added, no new leaf edge crossings were introduced. No new leaf vertices were added to previous leaf vertices, and $T_S \subset T'_S$, and so T' is a caterpillar. All vertex degrees decreased or stayed the same except for x_{i_2} and y_{j_1}, which both increased by one. Since these were leaves of T, we have that $\Delta(T') = \Delta^*_G$. Therefore, T' is a caterpillar MDST of G with fewer leaf edge crossings and a longer spine than T.

Otherwise, if multiple spine edges cross (x_{i_2}, y_{j_1}), let (x', y'') be the rightmost

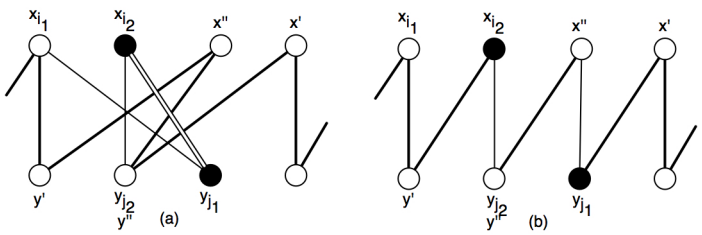

Figure 4.16: An example of a crossing of two leaf edges, removed as in Step 2a, Case 1.

edge that crosses (x_{i_2}, y_{j_1}) in $T \cup (x_{i_2}, y_{j_1})$ and let (x'', y') be the leftmost.

First, assume that $x', x'' > x_{i_2}$. An example is given in Figure 4.16 (a). Let $T' = (X \cup Y, E(T)$ - $\{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), (x'', y'), (x', y'')\} \cup \{(x_{i_2}, y'), (x_{i_2}, y''), (x'', y_{j_1}), (x', y_{j_1})\})$, as in Figure 4.16 (b). The edges removed disconnects T into five components, one containing x', one containing y', one containing x'' and y'', and the two isolated vertices x_{i_2} and y_{j_1}. Adding $(x_{i_2}, y'), (x_{i_2}, y'')$ connects the second and third components through x_{i_2}, and adding $(x'', y_{j_1}), (x', y_{j_1})$ connects the first and third components through y_{j_1}. So T' is a spanning tree. No new leaves are created and x_{i_2} and y_{j_1} are in T'_S. So T' is a caterpillar with $|T'_S| > |T_S|$, and T'_S is still a zig-zag. The degrees of all vertices decreased or stayed the same in T', except for x_{i_2} and y_{j_1}. These were leaves in T, and $deg_{T'}(x_{i_2}) = deg_{T'}(y_{j_1}) = 2$, so $\Delta(T') = \Delta^*_G$. Therefore, T' is a caterpillar MDST of G with a longer spine than T and with fewer edge crossings of two leaf edges.

Now assume that $x', x'' < x_{i_2}$. Let $T' = (X \cup Y, E(T)$ - $\{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), (x'', y'), (x', y'')\} \cup \{(x_{i_2}, y'), (x_{i_2}, y''), (x'', y_{j_1}), (x', y_{j_1})\})$, the same as defined above. The edges removed disconnects T into five components, one containing x'', one containing y'', one containing x' and y', and the two isolated vertices x_{i_2} and y_{j_1}. Adding $(x_{i_2}, y''), (x_{i_2}, y')$ connects the second and third components through x_{i_2}, and adding $(x'', y_{j_1}), (x', y_{j_1})$ connects the first and third components through y_{j_1}. So T' is a spanning tree. No new leaves are created and x_{i_2} and y_{j_1} are in T'_S. So T' is a caterpillar with $|T'_S| > |T_S|$, and T'_S is still a zig-zag. The degrees of all vertices decreased or stayed the same in T', except for x_{i_2} and y_{j_1}. These were leaves in T, and $deg_{T'}(x_{i_2}) = deg_{T'}(y_{j_1}) = 2$, so $\Delta(T') = \Delta^*_G$. Therefore, T' is a

caterpillar MDST of G with a longer spine than T and with fewer edge crossings of two leaf edges.

In this case, it is possible that T' contains edge crossings between edges added to the spine. However, from Step 1 we know there exists another MDST that is a caterpillar with the same spine as T' that contains no spine edge crossings.

Case 2: No edge in T_S forms a crossing with (x_{i_2}, y_{j_1}). Recall our assumption that $x_{i_1} < x_{i_2}$ and $y_{j_2} < y_{j_1}$. Since T_S is a zig-zag after Step 1, then x_{i_2} or y_{j_1} must be adjacent in G to the rightmost endpoint of T_S (or leftmost if we had assumed $x_{i_1} > x_{i_2}$). Note that an endpoint of T_S is not a leaf in T. Assume without loss of generality that the rightmost endpoint of T_S is in Y. Let y_{j_3} be this endpoint.

If $y_{j_2} = y_{j_3}$, let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_1})\} \cup \{(x_{i_2}, y_{j_1})\}$. y_{j_1} is now an endpoint of T'_S and x_{i_2} is now in T'_S as a middle vertex. Since x_{i_2} was an endpoint, T' is also a caterpillar. After removing (x_{i_1}, y_{j_1}), y_{j_1} becomes an isolated vertex. Adding (x_{i_2}, y_{j_1}) reconnects it to the graph. All vertex degrees decrease or remain the same, except for y_{j_1}. Since $deg_T(y_{j_1}) = 1$, we get $deg_{T'}(y_{j_1}) = 2$ and so $\Delta(T') = \Delta^*_G$. Therefore T' is a caterpillar MDST of G with one fewer edge crossing involving two leaf vertices, and $|T'_S| > |T_S|$.

Otherwise $y_{j_2} < y_{j_3}$, as in Figure 4.17 (a). Since y_{j_3} is the endpoint of T_S and has degree at least 2, and we have completed Step 1, we know that y_{j_3} must have at least one leaf in T that does not cross any edge in T_S. Let x_{i_3} be this leaf. We have two cases for the ordering of x_{i_2}, x_{i_3}.

If $x_{i_3} < x_{i_2}$, then let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})\} \cup \{(x_{i_3}, y_{j_1}), (x_{i_2}, y_{j_1})\}$, as in Figure 4.17 (b). Since (x_{i_3}, y_{j_3}) does not cross any other spine edges, we must have that $(x_{i_3}, y_{j_3}), (x_{i_1}, y_{j_1})$ is an edge crossing in T. Therefore we know that $(x_{i_3}, y_{j_1}) \in E(G)$. x_{i_3} was a leaf of an endpoint of T_S, and so adding y_{j_1} as its neighbour makes it a new endpoint of the spine with y_{j_1} as a leaf. Similarly, y_{j_1} becomes an endpoint with x_{i_2} as a leaf. So T' is a caterpillar. Additionally, a crossing of two leaf edges has been removed. Removing $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})$ disconnects T into three components, two of which are the isolated vertices x_{i_2}, y_{j_1}. Adding (x_{i_3}, y_{j_1}) connects y_{j_1} to the rest of the tree, and adding (x_{i_2}, y_{j_1}) connects

x_{i_2}. So T' is a spanning tree of G. Each vertex in T' has less than or equal to its degree in T, except for x_{i_3} and y_{j_1}. But these vertices were leaves in T and have degree 2 in T', so $\Delta(T') = \Delta^*_G$. Therefore T' is a caterpillar MDST of G with one fewer edge crossing involving two leaf vertices, and $|T'_S| > |T_S|$.

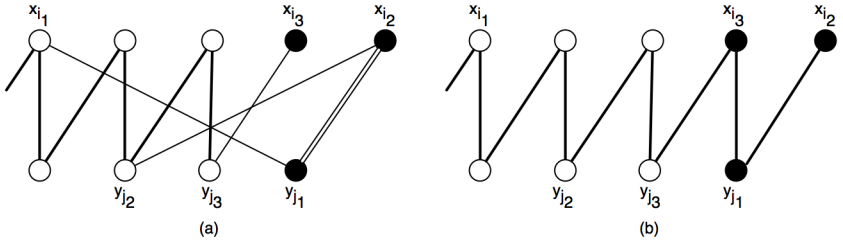

Figure 4.17: In Step 2a, Case 2, if $y_{j_2} < y_{j_3}$, we can extend the spine.

Otherwise, if $x_{i_2} < x_{i_3}$, then let $T' = (X \cup Y,\, E(T) - \{(x_{i_1}, y_{j_1}),\, (x_{i_2}, y_{j_2}),\, (x_{i_3}, y_{j_3})\} \cup \{(x_{i_2}, y_{j_3}),\, (x_{i_2}, y_{j_1}),\, (x_{i_3}, y_{j_1})\}$. Since (x_{i_3}, y_{j_3}) does not cross any other spine edges, we must have that $(x_{i_3}, y_{j_3}), (x_{i_1}, y_{j_1})$ is an edge crossing in T. Therefore we know that $(x_{i_3}, y_{j_1}) \in E(G)$. We also know since no edge of T_S crosses (x_{i_2}, y_{j_1}) in G, that $(x_{i_2}, y_{j_3}) \in E(G)$. T'_S is extended from T_S to have x_{i_2}, y_{j_1} added from the endpoint y_{j_3}, with x_{i_3} now a leaf of y_{j_1}. So T' is a caterpillar. The edge crossing $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})$ has been eliminated. Additionally, no new edge crossings in T'_S have been introduced. The edges of T removed to form T' disconnect T into four components, with the isolated vertices $x_{i_2}, x_{i_3}, y_{j_1}$. Adding (x_{i_2}, y_{j_3}) connects x_{i_2} to the rest of the tree, adding (x_{i_2}, y_{j_1}) connects y_{j_1} to the tree, and adding (x_{i_3}, y_{j_1}) makes T' a spanning tree of G. Each vertex in T' has less than or equal to its degree in T, except for x_{i_2} and y_{j_1}. But these vertices were leaves in T and have degree 2 in T', so $\Delta(T') = \Delta^*_G$. Therefore T' is a caterpillar MDST of G with one fewer edge crossing involving two leaf vertices, and $|T'_S| > |T_S|$.

If an MDST exists with an edge crossing as defined in this step, there exists an MDST with one fewer such edge crossing, with no edge crossings between two spine edges, and with a spine that is no shorter. Let T be an MDST with no such edge crossings, constructed as detailed above.

Step 2b: Next, if T contains an edge crossing that involves two leaf vertices that are either both in X or both in Y, we construct a new MDST T that contains fewer such crossings. That is, we will remove crossings involving two edges (x_{i_1}, y_{j_1}) and (x_{i_2}, y_{j_2}) where $x_{i_1}, x_{i_2} \in V(T_S)$ and $y_{j_1}, y_{j_2} \notin V(T_S)$, or $x_{i_1}, x_{i_2} \notin V(T_S)$ and $y_{j_1}, y_{j_2} \in V(T_S)$.

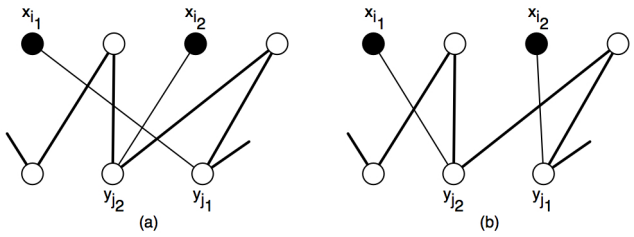

Figure 4.18: A crossing of two leaves both in X is removed in Step 2b.

Assume $i_1 < i_2$ and $j_1 > j_2$, as in Figure 4.18 (a). Then, $(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1}) \in E(G)$. Replace (x_{i_1}, y_{j_1}) and (x_{i_2}, y_{j_2}) with (x_{i_1}, y_{j_2}) and (x_{i_2}, y_{j_1}) in T, as in Figure 4.18 (b).

The degrees of all vertices remain the same, the crossing is removed, T is still a spanning tree, and the size of T_S does not change.

Using this transformation of T, we get a new MDST with one fewer of these edge crossings. So there exists an MDST constructed from T that contains no edge crossings containing two leaf edges or two spine edges, and has a spine no shorter. Let T be this MDST.

Step 3: Next, consider crossings in T containing a leaf edge that is involved in more than one distinct crossing. Assume without loss of generality that this leaf vertex is in Y. We remove these crossings, where an edge (x_{i_1}, y_{j_1}) with $x_{i_1} \in V(T_S)$ and $y_{j_1} \notin V(T_S)$, is contained in two distinct crossings.

Assume without loss of generality that $y_{j_1} \notin T_S$. Fix $y_{j_2} \in T_S$ such that $|j_2 - j_1|$ is minimized and y_{j_2} is in a crossing with (x_{i_1}, y_{j_1}). Assume without loss of generality that $y_{j_2} < y_{j_1}$.

We first show that two spine edges (x_{i_2}, y_{j_2}), (x_{i_3}, y_{j_3}) with distinct endpoints that cross (x_{i_1}, y_{j_1}) must exist: We have removed all crossings involving two leaf

edges, and so if (x_{i_1}, y_{j_1}) crosses multiple edges, these edges must be spine edges. Let y' be the rightmost neighbour of x_{i_1} in T_S and x' the rightmost neighbour of $y' \in T_S$. Since we have assumed $y_{j_2} < y_{j_1}$, and because T_S has no edge crossings, we have $y' < y_{j_1}$ as well. x_{i_1} is the leftmost neighbour in T_S of y', and so $x_{i_1} < x'$. Therefore, (x', y') forms a crossing with (x_{i_1}, y_{j_1}). Since (x_{i_1}, y_{j_1}) crosses multiple spine edges, and T_S is a zig-zag, then (x_{i_1}, y_{j_1}) is also crossed by (x', y''), where y'' is the rightmost neighbour of $x' \in T_S$. Since we have removed all crossings between leaf edges, y'' must have another neighbour $x'' \in T_S$, and because T_S is a zig-zag, $(x'', y''), (x_{i_1}, y_{j_1})$ is a crossing. So we have at least two edges with distinct vertices that form a crossing with (x_{i_1}, y_{j_1}).

Now, we keep y_{j_2} as previously fixed, and redefine $x_{i_2}, x_{i_3}, y_{j_3}$.

Case 1: y_{j_2} has no leaf in T, as in Figure 4.19 (a). Let x_{i_2}, x_{i_3} be the neighbours of y_{j_2} in T_S. Let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), (x_{i_3}, y_{j_2})\} \cup \{(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1}), (x_{i_3}, y_{j_1})\})$, as in Figure 4.19 (b). T'_S contains y_{j_1} instead of y_{j_2}, and y_{j_2} is a leaf in T'. Since y_{j_2} had no leaves in T, then T' is still a caterpillar. Replacing y_{j_2} with y_{j_1} in T'_S implies $|T_S| = |T'_S|$. We have that (x_{i_1}, y_{j_1}) crosses both (x_{i_2}, y_{j_2}) and (x_{i_3}, y_{j_2}) in T. Since G is a bipartite permutation graph, we know that the edges we added to T' must be in G.

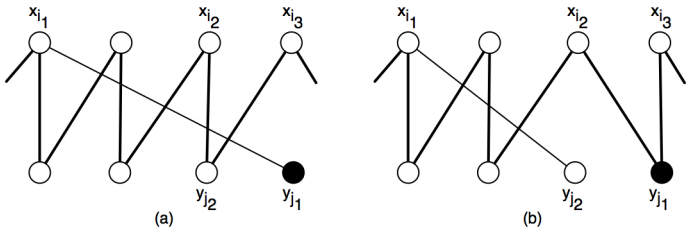

Figure 4.19: If y_{j_2} has no leaves, T' has fewer crossings of leaf and spine edges.

Removing $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), (x_{i_3}, y_{j_2})$ disconnects T into four components, one containing x_{i_1} and x_{i_2}, one containing x_{i_3}, and the two isolated vertices y_{j_1} and y_{j_2}. Adding $(x_{i_1}, y_{j_2}), (x_{i_2}, y_{j_1})$ to construct T' connects the isolated vertices to the first component and adding (x_{i_3}, y_{j_1}) connects the first and second components. Therefore T' is a spanning tree of G. $deg_{T'}(v) \leq deg_T(v)$ for all vertices $v \neq y_{j_1}$.

Since y_{j_1} was a leaf in T, we get $\Delta(T') = \Delta^*_G$. Therefore T' is an MDST of G.

Case 2: y_{j_2} has at least one leaf in T. Fix x_{i_2} as the rightmost such leaf. Since we have no edge crossing involving two leaf vertices, we have that $x_{i_2} < x_{i_1}$. Let x_{i_3} be the rightmost neighbour of y_{j_2} in T_S and let y_{j_3} be the leftmost neighbour of x_{i_1} in T_S, as labeled in Figure 4.20 (a). Consider the subpath P_S of T_S between y_{j_3} and x_{i_3}. Label the vertices in this subpath $(a_1, b_1, a_2, b_2, ..., a_k, b_k)$, with $a_1 = y_{j_3}$ and $b_k = x_{i_3}$. The edges in this subpath are (a_i, b_i), $1 \leq i \leq k$, and (b_i, a_{i+1}), $1 \leq i < k$. Label y_{j_1} as a_{k+1} and x_{i_2} as b_0. These labels are shown in Figure 4.20 (b), which shows the same subgraph of T as Figure 4.20 (a).

Let $T' = (X \cup Y, E(T) - \{\{(a_i, b_i)|1 \leq i \leq k\}, (x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})\} \cup \{\{(a_{i+2}, b_i)|0 \leq i \leq k-1\}, (x_{i_3}, y_{j_1}), (x_{i_2}, y_{j_3})\}$. This part of T' is shown in Figure 4.20 (c), and also in Figure 4.20 (d), with both labelling schemes.

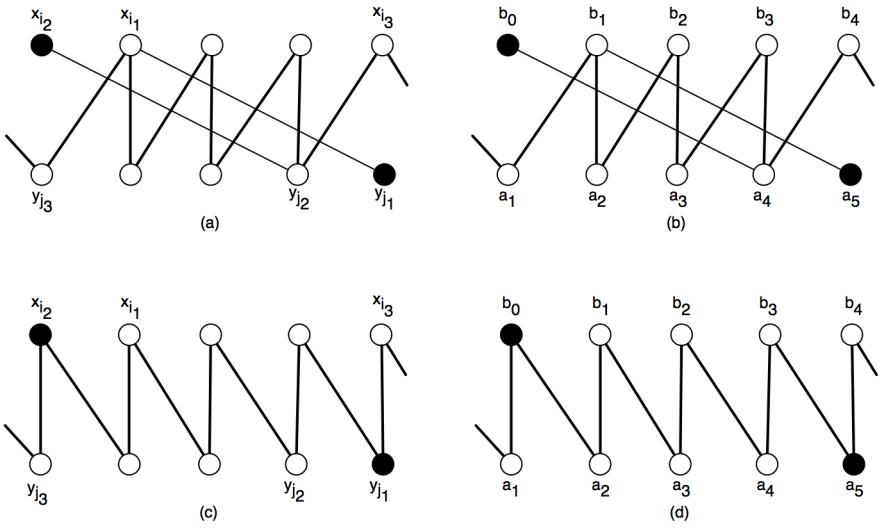

Figure 4.20: If y_{j_2} has a leaf, T' has a longer spine than T.

Because we have (x_{i_1}, y_{j_1}) crossing multiple spine edges, we know that b_i is adjacent in G to at least the vertices $a_i, ..., a_{k+1}$. So the edges added to T' are in G. No new leaves are created in T', and two leaf vertices in T are now in T'_S. So T' is a caterpillar with a longer spine than T.

The number of edges removed from T equals the number of edges added to

construct T'. After removing the edges from T and adding $\{(a_{i+2}, b_i)|0 \leq i \leq k-1\}$.which forms a path $(b_0, a_2, b_1, a_3, ..., a_k, b_{k-1}, a_{k+1})$, we have three components. One containing y_{j_3}, one containing x_{i_3}, and one containing this path. Adding (x_{i_3}, y_{j_1}) and (x_{i_2}, y_{j_3}) connects these components. $deg_{T'}(v) \leq deg_T(v)$ for all vertices $v \neq y_{j_1}$ or x_{i_2}. Since y_{j_1} and x_{i_2} were leaves in T and are now in T'_S with no leaves, $deg_{T'}(y_{j_1}) = deg_{T'}(x_{i_2}) = 2$. So $\Delta(T') = \Delta^*_G$. Therefore, T' is an MDST of G.

We have shown that if there exists an MDST of G with a leaf edge that crosses multiple spine edges, there exists an MDST of G with one fewer such crossing. So there exists an MDST of G with no leaf edges crossing multiple spine edges. Let T be this MDST, constructed as detailed above.

Step 4: Remove crossings that allow the spine of T to be extended. That is, given a subpath $(x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}) \in T_S$ with $x_{i_1} < x_{i_2}$, $y_{j_1} < y_{j_2}$, if there exist crossings $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_3})$ and $(x_{i_2}, y_{j_2}), (x_{i_3}, y_{j_1})$ such that $x_{i_3}, y_{j_3} \notin T_S$, we show how to extend the spine. If either x_{i_2} or y_{j_1} has multiple leaves, then choose x_{i_3} such that $|i_2 - i_3|$ is minimized, and choose y_{j_3} such that $|j_1 - j_3|$ is minimized. We assume without loss of generality that we have this particular case where our subpath starts in X on the left, however there is a symmetric case with $(y_{j_1}, x_{i_1}, y_{j_2}, x_{i_2})$ as the subpath.

Since we have completed Steps 2 and 3, we know that we have no leaf edges crossing and that (x_{i_2}, y_{j_3}) and (x_{i_3}, y_{j_1}) are involved in only one crossing, as shown in Figure 4.21 (a) . Therefore, we know that $x_{i_2} < x_{i_3}$ and $y_{j_3} < y_{j_1}$. G is a bipartite permutation graph, so we have $(x_{i_1}, y_{j_3}), (x_{i_3}, y_{j_2}) \in E(G)$.

Let $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})\} \cup \{(x_{i_1}, y_{j_3}), (x_{i_3}, y_{j_2})\}$, as in Figure 4.21 (b). The subpath $(x_{i_1}, y_{j_1}, x_{i_2}, y_{j_2}) \in T_S$ is extended to $(x_{i_1}, y_{j_3}, x_{i_2}, y_{j_1}, x_{i_3}, y_{j_2}) \in T'_S$. No new leaf vertices are created and $T_S \subset T'_S$, so T' is a caterpillar with a longer spine than T. Removing (x_{i_1}, y_{j_1}) and (x_{i_2}, y_{j_2}) disconnects T into three components, one containing x_{i_1}, one containing y_{j_2}, and one containing $x_{i_2}, x_{i_3}, y_{j_1}, y_{j_3}$. Adding (x_{i_1}, y_{j_3}) connects the first and third compo-

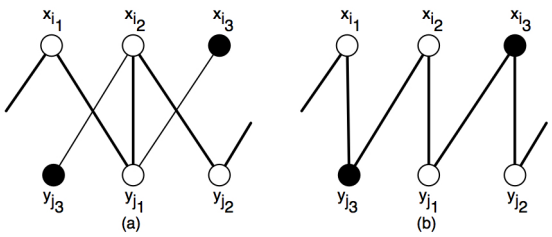

Figure 4.21: Two adjacent spine vertices with leaves imply the spine can be extended.

nents in T', and (x_{i_3}, y_{j_2}) connects the second and third. So T' is a spanning tree. $deg_{T'}(v) \leq deg_T(v)$ for all vertices $v \neq x_{i_3}$ or y_{j_3}. Since x_{i_3}, y_{j_3} were leaves in T and are now in T'_S with no leaves, $deg_{T'}(x_{i_3}) = deg_{T'}(y_{j_3}) = 2$. Therefore $\Delta(T') = \Delta^*_G$ and T' is an MDST of G.

We have shown that if there exists an MDST T with the edges crossings described in this case, we can construct an MDST T' that has one fewer such edge crossing, and a longer spine than T. So there exists an MDST with none of this type of edge crossing. Let T be that MDST, constructed as described above.

Step 5: We have eliminated all edge crossings involving two spine edges, two leaf edges, or a leaf edge that crosses multiple spine edges. All remaining crossings in T involve one spine edge and one leaf edge, $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2})$, with x_{i_2} or $y_{j_2} \notin V(T_S)$, and the rest in $V(T_S)$. Assume without loss of generality that $x_{i_1} < x_{i_2}$, $y_{j_2} < y_{j_1}$, and that $y_{j_2} \notin V(T_S)$ as shown in Figure 4.22 (a). We now construct a new MDST that has fewer edge crossings than T.

Consider the leftmost such crossing. Since (x_{i_2}, y_{j_2}) only crosses one spine edge, and T_S is a zig-zag, we have that $N_{T_S}(y_{j_1}) = \{x_{i_1}, x_{i_2}\}$. Assume y_{j_1} has a leaf $x_{i_3} \in T$. If $x_{i_3} < x_{i_2}$, then $(x_{i_2}, y_{j_2}), (x_{i_3}, y_{j_1})$ is a crossing in T. But T has no crossings of two leaf edges. So $x_{i_3} > x_{i_2}$. If x_{i_3} is in a crossing with an edge of T_S, then because (x_{i_2}, y_{j_2}) is a leaf edge, it would have been dealt with in Step 4. But if it does not cross an edge in T_S then T_S is not a zig-zag, or x_{i_2} is an endpoint of T_S. So y_{j_1} has no leaf in T.

Construct a new tree $T' = (X \cup Y, E(T) - \{(x_{i_1}, y_{j_1})\} \cup \{(x_{i_1}, y_{j_2})\})$, as in

Figure 4.22 (b). The crossing is removed, so all crossings in T' are further right. A new leaf is created in T', but we know y_{j_1} had no leaf in T. We created a new leaf y_{j_1}, but it was replaced in T'_S by y_{j_2}. So $|T_S| = |T'_S|$. So T' is a caterpillar with its spine the same size as T.

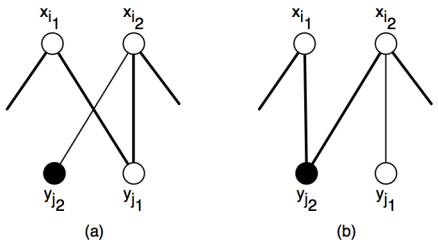

Figure 4.22: By Step 5, crossings of one spine edge and one leaf edge can be eliminated.

Removing (x_{i_1}, y_{j_1}) disconnects T into two components, one containing x_{i_1} and one containing y_{j_1}, x_{i_2}, and y_{j_2}. Adding (x_{i_1}, y_{j_2}) in T' connects these components.

$deg_{T'}(v) \leq deg_T(v)$ for all vertices $v \neq y_{j_2}$. Since y_{j_2} was a leaf in T, we have $deg_{T'}(y_{j_2}) = 2$. So $\Delta(T') = \Delta^*_G$. So T' is an MDST of G.

We have shown that if there exists an MDST that contains only edge crossings involving one leaf edge and one spine edge, as described in this case, there exists an MDST that contains one fewer such crossing, has no other edge crossings, and has a spine no shorter.

Therefore, we have shown that if there exists an MDST of G that is a caterpillar, there exists an MDST of G that is a caterpillar with a spine that has size greater than or equal to the size of the original spine, and that has no edge crossings.

□

We have shown that, given an MDST of G that contains edge crossings, there exists an MDST of G that contains no edge crossings and has a spine at least as long as the original MDST. Equivalently, the second MDST contains fewer leaves than the first. With this result, if we can show there exists some MDST of G that contains a longest path, we then know there exists some MDST of G that contains a longest path and has no edge crossings.

Let $G = (X, Y, E)$ be a strongly ordered, connected bipartite permutation graph. Let $\mathbb{P}$ be the set of all zig-zag longest paths in G. Let T be a crossing-free caterpillar MDST of G, and let P^* be the longest path in T that contains (x_1, y_1) and (x_p, y_q). By Lemma 4.4, we know T contains these edges, and that they are leaf edges on the endpoints of the spine of T. Therefore, such a P^* exists.

Lemma 4.21. *Let G be a strongly ordered, connected bipartite permutation graph, and let P^* be a zig-zag path in G that contains the edges (x_1, y_1) and (x_p, y_q). Then the vertices in P^*, $V(P^*)$, are contained in some zig-zag longest path in G.*

Proof. If P^* itself is a longest path in G, then the lemma holds. Assume P^* is not a longest path in G. Assume there does not exist a longest path in G that contains $V(P^*)$. By Lemma 4.4 we know there exists some longest path in G that contains the edges (x_1, y_1) and (x_p, y_q). Let $\mathbb{P}$ be the set of all such longest paths. Fix $P \in \mathbb{P}$ such that the number of consecutive vertices of P^* that are in P starting from the left is as large as possible. That is, choose P such that the first vertex in $P^* - P$ is positioned as far right in P^* as possible.

Assume without loss of generality that $x \in X(P^*)$ is the leftmost (in P^*) vertex in $P^* - P$. We know x is not an endpoint, or adjacent to an endpoint, in either P or P^*, since the edges (x_1, y_1) and (x_p, y_q) are in both paths. Fix y_{j_1}, y_{j_2} to be the neighbours of x in P^* with $y_{j_1} < y_{j_2}$. By Lemma 4.7 we have $y_{j_1}, y_{j_2} \in P$.

Now consider the (y_{j_1}, y_{j_2})-subpath of P. Let Q be this subpath.

Case 1: There exists some x' in Q such that $x' \in P - P^*$ or $x < x'$. In this case, we will show how to build another zig-zag longest path P' that matches P^* up to some vertex further right than x.

If $Q = y_{j_1}, x', y_{j_2}$, let $P' = (V(P), E(P) - \{(x', y_{j_1}), (x', y_{j_2})\} \cup \{(x, y_{j_1}), (x, y_{j_2})\})$. Now P' is a zig-zag longest path that contains the vertices of P^* from the left up to y_{j_2} instead of just to y_{j_1}, a contradiction.

Now consider when $|Q| > 3$. Fix x_{i_1}, x_{i_2} such that $Q = y_{j_1}, x_{i_1}, ..., x_{i_2}, y_{j_2}$. Let y', y'', $y' < y''$, be the neighbours of x' in P. By the adjacency property, $Y(Q) \subseteq N_G(x)$. Let $P' = (V(P), E(P) - \{(x', y'), (x', y'')\} \cup \{(x, y'), (x, y'')\})$. We now have P' as a longest path that contains $V(P^*)$ further right than P. We

may have introduced edge crossings in P', but we know there exists a crossing-free path containing the same vertices as P', since G is a bipartite permutation graph, any subgraph of a bipartite permutation graph is also a bipartite permutation graph, and P' is a Hamiltonian path in the bipartite permutation subgraph of G induced by its vertices. Now we have a zig-zag longest path that contains the vertices of P^* from the left up to y_{j_2} instead of just to y_{j_1}. If $x' \in P^*$, we have $x < x'$, and so now x' could be the leftmost vertex in $P^* - P$, a contradiction.

Case 2: $X(Q) \in P^*$ and $X(Q) \leq x$. In this case, we want to find some x' such that $x' < x$ and $x' \in P - P^*$ and build a new path P' that contains x instead of x'.

First, we show that in this case, there must exist $x' \in P - P^*$. Since we have assumed that P^* is not contained in any longest path of G, then $\Delta^*_G \geq 3$. By Lemma 4.14, we can assume that zig-zag longest paths in G all start in X or all start in Y. Assume that there does not exist such an x' (that is, all X vertices in P that are to the left of x are also in P^*), and consider the left endpoint of P.

Subcase 2a: Assume P starts in X. By our choice of $\mathbb{P}$, then P starts at x_1. Since G is bipartite, the (x_1, y_{j_2})-subpath of P has even size. Let k be the number of vertices in X in this subpath. Then there are also k vertices in Y in this subpath. Then $|P| = 2k + |back(P, y_{j_2})|$.

The (v_1, y_{j_2})-subpath of P^*, where $v_1 = x_1$ or y_1, contains x and so has one more vertex in X than the subpath in P. Since $y_{j_2} \in Y$ is an endpoint of this subpath, it must contain at least one more vertex in Y than P.

We also have that $front(P^*, y_{j_2}) \cap back(P, y_{j_2}) = \emptyset$, otherwise we would have Case 1. Let $P' = front(P^*, y_{j_2}) \cup \{y_{j_2}\} \cup back(P, y_{j_2})$. Now $|P'| = 2(k+1) + |back(P, y_{j_2})| > |P|$. We said P was a longest path, so this gives a contradiction.

Subcase 2b: Assume P starts in Y. By our choice of $\mathbb{P}$, then P starts at y_1. Since G is bipartite, the (y_1, y_{j_2})-subpath of P has an odd number of vertices. Let k be the number of vertices in X in this subpath. Then there are also $k+1$ vertices in Y in this subpath. Then $|P| = 2k + 1 + |back(P, y_{j_2})|$.

The (v_1, y_{j_2})-subpath of P^*, where $v_1 = x_1$ or y_1, contains x and so has one more vertex in X than the subpath in P. Since $y_{j_2} \in Y$ is an endpoint of this subpath, it must contain at least one more vertex in Y than P.

We also have that $front(P^*, y_{j_2}) \cap back(P, y_{j_2}) = \emptyset$, otherwise we would have Case 1. Let $P' = front(P^*, y_{j_2}) \cup \{y_{j_2}\} \cup back(P, y_{j_2})$. Now $|P'| = 2(k+1) + |back(P, y_{j_2})| > |P|$. We said P was a longest path, so this gives a contradiction.

Therefore, there exists some $x' \in P - P^*$ with $x' < x$. Fix x' to be the rightmost such vertex and let y' be the leftmost neighbour of x' in P. We will now show how to construct another zig-zag longest path P' that contains the vertices of P^* further right than P.

Let $X_r = x_{r_0}, x_{r_1}, ..., x_{r_i}$ with $r_j < r_{j+1}$ be the vertices in $X(P)$ between x' and x, where $x_{r_0} = x'$, and $x \notin X_r$. Let $Y_r = y_{r_0}, y_{r_1}, ..., y_{r_{i+1}}$ be the neighbours of X_P in P such that the leftmost neighbour of x_{r_j} in P is y_{r_j} and the rightmost neighbour of x_{r_j} in P is $y_{r_{j+1}}$. Let $y_{r_0} = y'$ and $y_{r_{i+1}} = y_{j_2}$. By our choices of x and x', we know that all vertices in P^* between x' and x, and between y' and y_{j_2} are in $X_P \cup Y_P$.

Constructing P' requires that we show $(x_{r_j}, y_{r_{j-1}}) \in E(G)$ for $1 \leq j \leq i$. Define $left(v, P)$ to be the leftmost neighbour of v in zig-zag path P and define $right(v, P)$ to be the rightmost neighbour in zig-zag path P. Since G is a bipartite permutation graph and has the adjacency property, we can show these edge are in $E(G)$ by showing that $left(v, P^*) < left(v, P)$ for all $v \in X_P$.

First consider x_{r_i}. Since x_{r_i} is on the (y_{j_1}, y_{j_2})-subpath in P, we have $left(x_{r_i}, P) \geq y_{j_1}$. We have that $y_{j_1} = left(x, P^*)$, and since $x_{r_i} < x$, and our paths are zig-zags, then $left(x_{r_i}, P^*) < y_{j_1} \leq left(x_{r_i}, P)$.

Next, for the rest of X_r, we know that for x_{r_j} and x_{r_k} with $x_{r_j} < x_{r_k}$, $left(x_{r_j}, P) = y_{r_j} < left(x_{r_k}, P) = y_{r_k}$ and there are exactly as many vertices in X_r between x_{r_j} and x_{r_k} as there are between y_{r_j} and y_{r_k} in Y_r. Now consider $left(x_{r_j}, P^*)$ and $left(x_{r_k}, P^*)$. Moving from right to left, we know that the distance between $left(x_{r_j}, P)$ and $left(x_{r_k}, P)$ is the same as the distance between x_{r_j} and x_{r_k}. However, since $V(P^*) \subset V(P)$ in this subpath, but Y_r is not necessarily contained in P^*, then the distance between $left(x_{r_j}, P^*)$ and $left(x_{r_k}, P^*)$ is at least the distance between x_{r_j} and x_{r_k}. Therefore, since we already had that $left(x_{r_i}, P^*) < left(x_{r_i}, P)$, for each $x_{r_j} \in X_r$, we have that $left(x_{r_j}, P^*) < y_{j_1} \leq left(x_{r_j}, P)$.

So $(x_{r_j}, y_{r_{j-1}}) \in E(G)$ for $1 \leq j \leq i$.

Let $E_L = \{(x_{r_j}, y_{r_{j-1}})\}$, for $1 \leq j \leq i$, and let $E_R = \{(x_{r_j}, y_{r_{j+1}})\}$ for $1 \leq j \leq i$.

Now, let $P' = (V(P), E(P) - \{E_R, (x', y'), (x', right(x', P))\} \cup \{E_L, (x, y_{r_i}), (x, y_{j_2})\})$. The edges between each x_{r_j} and its rightmost neighbour in P are removed, as well as the edges containing x'. The edges added in E_L connect each x_{r_j} to the vertex in $Y(P)$ immediately left of its leftmost neighbour. We showed above that these edges are in G. This results in two subpaths, one from the left up to y_{r_i}, which is now $right(x_{r_i}, P')$, and the other beginning at y_{j_2} and going to the right.

We know $(x, y_{j_2}) \in E(G)$ because it is in P^*. We also know that $y_{r_i} \geq y_{j_1}$. By the adjacency property, $(x, y_{r_i}) \in E(G)$. Adding these edges connects the two path components. Therefore P' is a path.

Each edge added between x_{r_j} and $y_{r_{j-1}}$ will not introduce any edge crossings because the only edge in P it would cross is $(x_{r_{j-1}}, y_{r_j})$, which is in E_R. Therefore, P' contains no edge crossings and is a zig-zag.

P' contains all vertices of P except for x', and it instead contains x. Therefore, P' is a zig-zag longest path that contains the vertices of P^* further to the right than P, a contradiction.

Therefore, there must exist a zig-zag longest path in G that contains the vertices of P^*.

□

Theorem 4.22. *Let $G = (X, Y, E)$ be a strongly ordered, connected bipartite permutation graph. Then there exists a crossing-free caterpillar MDST of G that contains a longest path of G.*

Proof. Assume that no MDST of G contains a longest path. That is, for every MDST T of G, the longest path P^* in T has size less than the size of a longest path. Choose T to be a crossing-free MDST of G and let P^* be the spine of T.

By Lemma 4.21, there is some longest path that contains the vertices of P^*. Fix P to be such a zig-zag longest path. Let E' be the set of edges in T that contain a vertex in $V(P)$.

Let $T' = (V(T), E(T) - E' \cup E(P)$. First, we show that T' is a spanning tree. The subgraph of T induced by E' contains the spine of T and some leaves. Since

$V(P^*) \subset V(P)$, this subgraph contains only spine vertices and leaves in T that are also on P. If a vertex is not in E', then it is not in P. Therefore, $|E'| = |P| - 1$. Since P is a path, $|E(P)| = |P| - 1$ as well, and so T' contains the same number of edges as T. Removing the edges of E' from T produces components that are either isolated vertices, which were either leaves in T that were on P, or spine vertices in T that were only adjacent to other vertices that were in P, or are K_2, containing one vertex from P^* and another vertex from $T - P$. Since each component contains exactly one vertex from P, adding the edges from P connects the tree.

Now consider the degrees of vertices in T'. For all vertices in $T - P$, the degree remained the same. Vertices in $P - P^*$ were leaves in T and are on the spine of T'. However, since we removed the one edge containing these vertices from T and only added path edges back, vertices in $P - P^*$ will have degree at most 2 in T.

Vertices in P^* have degree at least 2 in T. In T', some of these vertices will have leaves removed, if these leaves were in $P - P^*$. Then, each vertex in P^* has its two incident spine edges removed, and replaced with two path edges from P. Therefore, $deg_{T'}(v) \leq deg_T(v)$, for all $v \in P^*$. Therefore, T' is an MDST of G.

It is possible that edge crossings were introduced in the construction of T'. If T' contains edge crossings, by Theorem 4.20, there exists another MDST of G that contains no edge crossings and has a spine at least as long as T'. Since T' contains a longest path, this other MDST will also contain a longest path.

Therefore, by contradiction, there exists a crossing-free MDST of G that contains a longest path.

□

4.4 An Algorithm for $\triangle_G^*$ of a Bipartite Permutation Graph

We now present a dynamic programming algorithm to solve the minimum degree spanning tree problem on and construct an MDST of a bipartite permutation graph.

Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p, y_1, ..., y_q$. We define $[i, j]$ to be the induced subgraph of G con-

taining the vertices $x_i, ..., x_p, y_j, ..., y_q$.

Lemma 4.23. *Let $G = (X, Y, E)$ be a connected bipartite permutation graph with strong ordering $x_1, ..., x_p, y_1, ..., y_q$, and for all $1 \leq i \leq p$, $1 \leq j \leq q$, let*

$$\Delta_x^*[i,j] = \begin{cases} \infty & \text{if } [i,j] \text{ is not connected} \\ \Delta([i,j]) & \text{if } [i,j] \text{ is a tree with } x_i \text{ as a leaf} \\ \infty & \text{if } N_{[i,j]}(y_j) = \{x_i\} \\ \min_k\{\max\{k+1, \Delta_y^*[i+k,j]\}\} & \text{for } 1 \leq k \leq |N_{[i,j]}(y_j)| - 1 \text{ otherwise} \end{cases}$$

and

$$\Delta_y^*[i,j] = \begin{cases} \infty & \text{if } [i,j] \text{ is not connected} \\ \Delta([i,j]) & \text{if } [i,j] \text{ is a tree with } y_j \text{ as a leaf} \\ \infty & \text{if } N_{[i,j]}(x_i) = \{y_j\} \\ \min_k\{\max\{k+1, \Delta_x^*[i,j+k]\}\} & \text{for } 1 \leq k \leq |N_{[i,j]}(x_i)| - 1 \text{ otherwise} \end{cases}$$

Then for all i, j, (1) $\Delta_x^*[i,j]$ is the minimum maximum degree of a crossing-free spanning tree of $[i,j]$ with x_i as a leaf, and (2) $\Delta_y^*[i,j]$ is the minimum maximum degree of a crossing-free spanning tree of $[i,j]$ with y_j as a leaf.

Proof. Let $i = p$ and $j = q$. By Theorem 4.2, the edge (x_p, y_q) is in every bipartite permutation graph. Therefore, the subgraph $[p,q]$ consists of just that edge, which is a tree with both x_p and y_q as leaves. So $\Delta_x^*[p,q] = \Delta([p,q]) = 1$ and $\Delta_y^*[p,q] = \Delta([p,q]) = 1$.

Assume the claim holds for all $[i,j]$ such that $i > a$ and $j \geq b$, or $i \geq a$ and $j > b$.

Now consider $\Delta_x^*[a,b]$. If $[a,b]$ is not connected, then it does not have a spanning tree, and $\Delta_{[a,b]}^* = \Delta_x^*[a,b] = \infty$. If $[a,b]$ is a tree with x_a as a leaf, then there is exactly one spanning tree of $[a,b]$, and by Theorem 4.16, we know it is crossing-free. Thus, in this case $\Delta_x^*[a,b] = \Delta([a,b])$. If $[a,b]$ is a tree without x_a as a leaf, then $N_{[a,b]}(y_b) = x_a$. If $N_{[a,b]}(y_b) = x_a$, there is no crossing-free spanning tree of $[a,b]$ with x_a as a leaf, since in this case any crossing-free spanning tree must have y_b as a leaf of x_a. Thus, in this case $\Delta_x^*[a,b] = \infty$.

In all other cases, we claim that $\Delta_x^*[a,b] = \min_k\{\max\{k+1, \Delta_y^*[a+k,b]\}\}$ for $1 \leq k \leq |N_{[a,b]}(y_b)| - 1$. In all crossing-free spanning trees of $[a,b]$ with x_a as a leaf, x_a must be a leaf of y_b, and y_b can have between one and $|N_{[a,b]}(y_b)| - 1$ leaves in such a crossing-free spanning tree, plus one non-leaf (spine) neighbour. Thus, the degree of y_b in a crossing-free spanning tree of $[a,b]$ is $k+1$ where k is the number of leaves of y_b.

Let Δ'_x be the minimum maximum degree of a crossing-free spanning tree of $[a,b]$ with x_a as a leaf. First, assume $\Delta'_x > \Delta_x^*[a,b]$. Then all crossing-free spanning trees of $[a,b]$ with x_a as a leaf must have maximum degree $\geq \Delta'_x$. Let l be the value of k that minimizes $\max\{k+1, \Delta_y^*[a+k,b]\}$. So $\Delta'_x > l+1$ and $\Delta'_x > \Delta_y^*[a+l,b]$. Let T_y be a crossing-free MDST of $[a+l,b]$. Let $T = (V(T_y) \cup \{x_a, ..., x_{a+l-1}\}, E(T_y) \cup \{(x', y_b) \,|\, x_a \leq x' \leq x_{a+l-1}\})$. Now T is a crossing-free spanning tree of $[a,b]$ with x_a as a leaf and $\Delta(T) = \max\{l+1, \Delta_y^*[a+l,b]\} < \Delta'_x$, a contradiction.

Now assume $\Delta'_x < \Delta_x^*[a,b]$. Let T be a crossing-free spanning tree of G with x_a as a leaf, such that $\Delta(T) = \Delta'_x$. Since T is crossing-free, y_b has between one and $|N_{[a,b]}(y_b)| - 1$ leaves in T. Let l be the number of leaves y_b has in T. Then $\Delta(T) \geq l+1$. Since $\Delta_x^*[a,b]$ is too large, then when $k = l$, we must have $\Delta_y^*[a+l,b] > \Delta'_x \geq l+1$. Let $T_y = (V(T) - \{x_a, ..., x_{a+l-1}\}, E(T) - \{(x', y_b) \,|\, x_a \leq x' \leq x_{a+l-1}\})$. Now T_y is a crossing-free spanning tree of $[a+l,b]$ with y_b as leaf, and $\Delta(T_y) \leq \Delta'_x$. Therefore, $\Delta(T_y) < \Delta_y^*[a+l,b]$. But by our inductive hypothesis, $\Delta_y^*[a+l,b]$ is the minimum maximum degree of a crossing-free spanning tree of $[a+l,b]$ with y_b as a leaf, a contradiction. Therefore, $\Delta'_x = \Delta_x^*[a,b]$.

Therefore, (1) holds. An equivalent argument can be made for (2), and so the lemma holds.

□

Theorem 4.24. *For a strongly ordered, connected bipartite permutation graph G, Δ_G^* can be found in $O(n^3)$ time.*

Proof. By Corollary 4.19, every crossing-free spanning tree of a bipartite permutation graph is a caterpillar. By Lemma 4.17, if $[i,j]$ is connected then the edge

(x_i, y_j) is in any crossing-free spanning tree of $[i, j]$. If G has two or fewer vertices, then either G has no edges or G has at most one edge and that edge is a crossing-free spanning tree of G. We now assume that G contains at least three vertices.

If both x_i and y_j are in the spine, then they each have a leaf, y' and x' respectively such that $x_i < x'$ and $y_j < y'$. Then $(x_i, y'), (x', y_j)$ is an edge crossing. Therefore, at least one of x_i, y_j must be a leaf. By the definition of a caterpillar, leaf vertices are only adjacent to spine vertices so at most one of x_i, y_j is a leaf vertex.

Therefore, in any crossing-free spanning tree, either x_i is a leaf or y_j is a leaf. Then the minimum maximum degree of crossing-free spanning trees of $[i, j]$ can be expressed as the minimum of the minimum maximum degree of crossing-free spanning trees of $[i, j]$ with x_i as a leaf and the minimum maximum degree of crossing-free spanning trees of $[i, j]$ with y_j as a leaf.

By Lemma 4.23, we know that $\Delta^*_x[i, j]$ is the minimum maximum degree of a crossing-free spanning tree of $[i, j]$ with x_i as a leaf, and $\Delta^*_y[i, j]$ is the minimum maximum degree of a crossing-free spanning tree of $[i, j]$ with y_j as a leaf. Therefore, $\min\{\Delta^*_x[i, j], \Delta^*_y[i, j]\}$ gives the minimum maximum degree of a crossing-free spanning tree of G. By Theorem 4.18 there exists an MDST of all bipartite permutation graphs that is crossing free, and so $\Delta^*_{[i,j]} = \min\{\Delta^*_x[i, j], \Delta^*_y[i, j]\}$.

When $i = 1, j = 1$, then $[i, j] = G$. Therefore, $\Delta^*_G = \min\{\Delta^*_x[1, 1], \Delta^*_y[1, 1]\}$ for a bipartite permutation graph G.

An algorithm based on the formulas of the lemma constructs two p by q tables for $\Delta^*_x[i, j]$ and $\Delta^*_y[i, j]$, for a total of $2 \times p \times q \in O(n^2)$ entries. For each entry, $k \leq \max\{p, q\} \leq n$. Thus, the algorithm finds Δ^*_G in $O(n^3)$ time.

□

Theorem 4.25. *For a strongly ordered, connected bipartite permutation graph G, an MDST of G can be constructed in $O(n^3)$ time.*

Proof. Using the tables produced by the algorithm, we can construct a crossing-free spanning tree T of each $[i, j]$ where $\Delta^*_x[i, j]$ is finite such that x_i is a leaf in T and $\Delta(T) = \Delta^*_x[i, j]$. If $[i, j]$ is a tree, then $[i, j]$ is a crossing-free spanning tree

of itself. Otherwise, we can construct T by choosing a value of k that minimizes $\max\{k+1, \Delta_y^*[i+k, j]\}$ and adding $x_i, ..., x_{i+k-1}$ as leaves of y_j to a crossing-free spanning tree T' of $[i+k, j]$ with $\Delta(T') = \Delta_y^*[i+k, j]$ that has y_j as a leaf.

Similarly, we can construct a crossing-free spanning tree T of each $[i, j]$ where $\Delta_y^*[i, j]$ is finite such that y_j is a leaf in T and $\Delta(T) = \Delta_y^*[i, j]$. If $[i, j]$ is a tree, then $[i, j]$ is a crossing-free spanning tree of itself. Otherwise, we can construct T by choosing a value of k that minimizes $\max\{k+1, \Delta_x^*[i, j+k]\}$ and adding $y_j, ..., y_{j+k-1}$ as leaves of x_i to a crossing-free spanning tree T' of $[i, j+k]$ with $\Delta(T') = \Delta_x^*[i, j+k]$ that has x_i as a leaf.

By Theorem 4.24, we know $\Delta_G^* = \min\{\Delta_x^*[1, 1], \Delta_y^*[1, 1]\}$, and so we can use this method to construct a crossing-free MDST of G.

Therefore, the theorem holds. □

Chapter 5

Conclusion

We now summarize the work presented in this thesis, and discuss some directions for future work.

5.1 Summary

In this thesis, we explored the minimum degree spanning tree problem on bipartite permutation graphs and their subclasses.

Chain graphs are a subclass of bipartite permutation graphs, and we give an algorithm to construct an MDST on a given chain graph, as well as a formula to find Δ_G^* for a chain graph G in $O(n^2)$ time.

For bipartite permutation graphs, we build on work done on the Hamiltonian path and longest path problems to get a number of structural results for longest paths and MDSTs on this class. We show which vertices must be in a zig-zag longest path in a bipartite permutation graph, and that certain longest paths can only exist in a bipartite permutation graph with a Hamiltonian path.

We generalize the idea of zig-zag paths to show that every bipartite permutation graph has an MDST that has no edge crossings, and so is a caterpillar. We also show that every bipartite permutation graph has an MDST with no edge crossings that contains a longest path.

We present a dynamic programming algorithm that solves the minimum degree spanning tree problem for bipartite permutation graphs by finding Δ_G^* and constructing an MDST of G in $O(n^3)$ time.

Therefore, we have shown that the minimum degree spanning tree problem can be solved in polynomial time for bipartite permutation graphs and their subclasses.

5.2 Future Work

There are a number of improvements that could be made based on the results presented in this thesis.

We initially conjectured that every longest path in a bipartite permutation graph G was contained in some caterpillar MDST of G. However, the graph and longest path in Figure 5.1 is a counterexample to this conjecture.

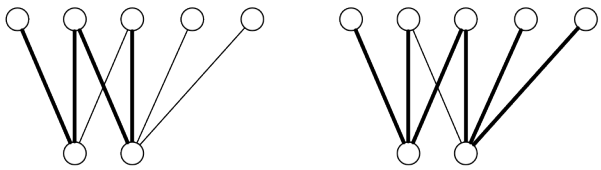

Figure 5.1: The longest path on the left is not contained in any MDST of the graph. On the right is a spanning tree with maximum degree three and so $\Delta^* \leq 3$.

By Theorem 4.22, we know that some longest paths are contained in crossing-free MDSTs and, by the counterexample above, we know some are not. Finding a characterization of the zig-zag longest paths that are contained in crossing-free MDSTs could help lead to a more efficient algorithm to solve the problem. Since these paths can be found in linear time, we may be able to find a linear time algorithm for the minimum degree spanning tree problem.

There is some room for improvement on the running time of the algorithm in Theorem 4.24. One possible improvement is to add checks for a Hamiltonian path in $[i, j]$ when computing Δ^*_x and Δ^*_y. Another is that once k exceeds the minimum $\max\{k+1, \Delta^*_y[i+k, j]\}$ or $\max\{k+1, \Delta^*_x[i, j+k]\}$ value found so far, considering the case where there are k leaves will always give a higher maximum degree than optimal, and so these higher k values do not need to be calculated.

The results for bipartite permutation graphs introduce a number of structural features of some MDSTs in chain graphs. Combining these results with the nested

neighbourhood properties of chain graphs, it may be possible to improve on our result for finding an MDST in a chain graph. In particular, it is worth considering the possibility that an MDST can be constructed for an arbitrary chain graph in linear time.

The results in this thesis could also be generalized to superclasses of bipartite permutation graphs. Biconvex graphs have an ordering that generalizes the strong ordering of bipartite permutation graphs [1]. The properties of this ordering may allow our results on bipartite permutation graphs to be generalized to this class of graphs.

We use the nested neighbourhood ordering property of chain graphs in our algorithm to solve the minimum degree spanning tree problem on these graphs. The nested neighbourhood ordering also appears in the independent set in the partition of a threshold graph, and so these results may be transferable to threshold graphs as well.

In this thesis, we focus on the Hamiltonian path and longest path problems as starting points to solve the minimum degree spanning tree problem. However, there are other related problems that may also be helpful in solving this problem on other graph classes. One potential area for exploration is the dominating path problem. This problem can be solved in linear time on AT-free graphs [13]. A dominating path can be used to easily construct a spanning caterpillar of a graph, by adding all vertices not on the path as leaves. In a crossing-free MDST of a bipartite permutation graph, the spine of the caterpillar is a dominating path in the graph. It is worth exploring if a relationship exists between the dominating path problem and the minimum degree spanning tree problem.

Bibliography

[1] N Abbas and L Stewart. Biconvex graphs: ordering and algorithms. *Discrete Applied Mathematics*, 103(1-3):1–19, 2000.

[2] L Babel and G Woeginger. Pseudo-Hamiltonian graphs. In *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'97)*, volume 1335 of *LNCS*, pages 38–51. Springer-Verlag, 1997.

[3] H Bodlaender, T Kloks, and D Kratsch. Treewidth and pathwidth of permutation graphs. In *Automata, Languages and Programming*, volume 700 of *LNCS*, pages 114–125. Springer Berlin / Heidelberg, 1993.

[4] J A Bondy and V Chvátal. A method in graph theory. *Discrete Mathematics*, 15(2):111 – 135, 1976.

[5] O Borůvka. On a minimal problem. *Práce Moravské Pridovedecké Spolecnosti*, 3, 1926.

[6] A Brandstädt and D Kratsch. On the restriction of some NP-complete graph problems to permutation graphs. In L Budach, editor, *Fundamentals of Computation Theory*, volume 199 of *LNCS*, pages 53–62. Springer Berlin / Heidelberg, 1985.

[7] A Brandstädt, V.B. Le, and J. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[8] Y Caro, I Krasikov, and Y Roditty. On the largest tree of given maximum degree in a connected graph. *Journal of Graph Theory*, 15(1):7–13, 1991.

[9] K Chaudhuri, S Rao, S Riesenfeld, and K Talwar. What would Edmonds do? Augmenting paths and witnesses for degree-bounded MSTs. *Algorithmica*, 55(1):157–189, 2009.

[10] B Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[11] V Chvátal and P Erdös. A note on Hamiltonian circuits. *Discrete Mathematics*, 2(11):1–1, 1972.

[12] T Cormen, C Leiserson, R Rivest, and C Stein. *Introduction to Algorithms*. The MIT Press, New York, second edition, 2001.

[13] D Corneil, S Olariu, and L Stewart. A linear time algorithm to compute a dominating path in an AT-free graph. *Information Processing Letters*, 54(5):253–257, 1995.

[14] D Corneil, S Olariu, and L Stewart. Asteroidal triple-free graphs. *SIAM Journal of Discrete Mathematics*, 10(3):399–430, 1997.

[15] A Czumaj and WB Strothmann. Bounded degree spanning trees. In R Burkard and G Woeginger, editors, *Algorithms ESA '97*, volume 1284 of *LNCS*, pages 104–117. Springer Berlin / Heidelberg, 1997.

[16] A Czygrinow, G Fan, G Hurlbert, HA Kierstead, and WT Trotter. Spanning trees of bounded degree. *The Electronic Journal of Combinatorics*, 8(R33):1, 2001.

[17] P Damaschke, JS Deogun, D Kratsch, and G Steiner. Finding Hamiltonian paths in cocomparability graphs using the bump number algorithm. *Order*, 8:383–391, 1992.

[18] EW Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[19] GA Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, 3(1):69, 1952.

[20] P Eades, B McKay, and N Wormald. On an edge crossing problem. In *Proceedings of the Ninth Australian Computer Science Conference*. 1986.

[21] S Fekete, S Khuller, M Klemmstein, and B Raghavachari. A network-flow technique for finding low-weight bounded-degree spanning trees. *Journal of Algorithms*, 24:310–324, 1997.

[22] H Fernau, J Kneis, D Kratsch, A Langer, M Liedloff, D Raible, and P Rossmanith. An exact algorithm for the maximum leaf spanning tree problem. In J Chen and F Fomin, editors, *Parameterized and Exact Computation*, volume 5917 of *LNCS*, pages 161–172. Springer Berlin / Heidelberg, 2009.

[23] T Fischer. Optimizing the degree of minimum weight spanning trees. *Technical Report 93-1338, Department of Computer Science, Cornell University*, 1993.

[24] M Fürer and B Raghavachari. An NC approximation algorithm for the minimum degree spanning tree problem. *Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing*, pages 274–281, 1990.

[25] M Fürer and B Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994.

[26] H.N Gabow and R.E Tarjan. Efficient algorithms for a family of matroid intersection problems. *Journal of Algorithms*, 5(1):80–131, 1984.

[27] MR Garey and DS Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* W. H. Freeman & Co., New York, 1979.

[28] MX Goemans. Minimum bounded degree spanning trees. *47th Annual IEEE Symposium on Foundations of Computer Science, 2006. FOCS'06*, pages 273–282, 2006.

[29] MC Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland, Amsterdam, The Netherlands, second edition, 2004.

[30] RL Graham and P Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[31] F Harary and U Peled. Hamiltonian threshold graphs. *Discrete Applied Mathematics*, 16(1):15, 1987.

[32] K Ioannidou, G Mertzios, and S Nikolopoulos. The longest path problem is polynomial on interval graphs. In R Krlovic and D Niwinski, editors, *Mathematical Foundations of Computer Science 2009*, volume 5734 of *LNCS*, pages 403–414. Springer Berlin / Heidelberg, 2009.

[33] V Jarník. About a certain minimal problem. *Práce Moravské Pridovedecké Spolecnosti*, 6:57–63, 1930.

[34] R Jothi and B Raghavachari. Degree-bounded minimum spanning trees. *Discrete Applied Mathematics*, 157(5):960–970, 2009.

[35] JB Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[36] J M Keil. Finding Hamiltonian circuits in interval graphs. *Information Processing Letters*, 20(4):201–206, 1985.

[37] S Khuller, B Raghavachari, and N Young. Low degree spanning trees of small weight. *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, page 421, 1994.

[38] H Kishimoto and M Kano. Spanning k-trees of n-connected graphs. *Graphs and Combinatorics*, pages 1–6, 2011.

[39] E.L Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, New York, 1976.

[40] GB Mertzios and DG Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *Technical report available at http://arxiv.org/abs/1004.4560*, 2010.

[41] J Moon and L Moser. On Hamiltonian bipartite graphs. *Israel Journal of Mathematics*, 1(3):163–165, 1963.

[42] H Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1):291–298, 1996.

[43] V Neumann-Lara and E Rivera-Campo. Spanning trees with bounded degrees. *Combinatorica*, 11(1):55–61, 1991.

[44] O Ore. Note on Hamilton circuits. *American Mathematical Monthly*, 67(1):55–55, 1960.

[45] K Ozeki and T Yamashita. Spanning trees: A survey. *Graphs and Combinatorics*, pages 1–26, 2010.

[46] RC Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[47] R Ravi and M Singh. Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In M Bugliesi, B Preneel, V Sassone, and I Wegener, editors, *Automata, Languages and Programming*, volume 4051 of *LNCS*, pages 169–180. Springer Berlin / Heidelberg, 2006.

[48] SJ Riesenfeld. Optimization and reconstruction over graphs. *PhD. thesis, University of California at Berkeley*, Jan 2008.

[49] G Robins and JS Salowe. Low-degree minimum spanning trees. *Discrete and Computational Geometry*, 14(1):151–165, 1995.

[50] M Singh and LC Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, page 670, 2007.

[51] J Spinrad, A Brandstädt, and L Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

[52] R Uehara and Y Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5):911–930, 2007.

[53] R Uehara and G Valiente. Linear structure of bipartite permutation graphs and the longest path problem. *Information Processing Letters*, 103(2):71–77, 2007.

[54] D West. *Introduction to Graph Theory*. Prentice Hall, second edition, 2000.

[55] S Win. On a conjecture of Las Vergnas concerning certain spanning trees of graphs. *Resultate der Mathematik*, 2:215–224.

[56] S Win. Über Gerüste mit vorgeschriebenen Maximalgraden und eine Vermutung von Las Vergnas. *Ph.D. Thesis, Universität Hamburg*, 1979.

[57] S Win. On a connection between the existence of k-trees and the toughness of a graph. *Graphs and Combinatorics*, 5(1):201–205, 1989.

[58] M Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods*, 3(3):351–358, 1982.

[59] L Zhenhong and X Baoguang. On low bound of degree sequences of spanning trees in k-edge-connected graphs. *Journal of Graph Theory*, 28(2):87–95, 1998.