Bandwidth Problem in Graphs

A thesis submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

Thesis Supervisor: Prof. C. Pandu Rangan

by
R. Vinay Yadav
April, 2012

Indian Institute of Science Education and Research Pune
Sai Trinity Building, Pashan, Pune India 411021

This is to certify that this thesis entitled "Bandwidth Problem in Graphs" submitted towards the partial fulfillment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research Pune, represents work carried out by R. Vinay Yadav under the supervision of Prof. C. Pandu Rangan.

R. Vinay Yadav

Thesis committee:
Prof. C. Pandu Rangan
Dr. Soumen Maity

Prof. A. Raghuram
Coordinator of Mathematics

to all lovers of Graph Theory

Acknowledgments

It is a pleasure to thank those who made this thesis possible. First and foremost, I would like to thank my advisor **Prof. C. Pandu Rangan** for his immense support and guidance. I feel privileged to be among his students. He celebrates research in the truest sense of the term. I would also like to thank him for providing us with such an excellent opportunity, and also for his encouragement throughout the duration of the project. Despite his busy schedule, he always found time to discuss with us and guide us throughout the project. His guidance has been an extraordinary learning experience.

I would like to express my sincere gratitude to my IISER guide, **Dr. Soumen Maity**, Assistant Professor, Department of Mathematics, IISER Pune, for his valuable suggestions, encouragement, and guidance throughout the entire duration of the project work. I am also grateful to **Dr. Ayan Mahalanobis** for his constant advice and assistance during my stay in IISER Pune.

I would like to thank **Ms. Esha Ghosh** who was my mentor during the course of the project. She provided me with the insight and constant help throughout the project. I am also thankful to Vivek Pradhan and Uma Sridhar, my classmate and colleague, without whom, this project would have been impossible.

I thank all my lab mates Kartik, Guhan, Prateek, Sachin, Chaya, Dinakar, Salini, Akash, Sharmi madam, Vivek sir, Preetha madam, and Sangeetha who made my stay at TCS lab fond and special. This place has left a lasting impression in me. This has been undoubtedly the best environment I have studied in and the above people were responsible for it. I also thank Brinda, Parshu, Navya, Abhilash, Bala and Satish for making my stay in IIT Madras a memorable one. I would like to thank "The Cobras" of HR1 who have given me experiences which will stay forever with me.

Last but not the least, I owe my deep sense of gratitude to my family as they have been very supportive throughout my life during the good and hard times.

Abstract

Bandwidth Problem in Graphs

by R. Vinay Yadav

KEYWORDS: *Computational Complexity, NP completeness, bandwidth problem, biconvex bipartite graph, multiple processor scheduling problem.*

Bandwidth is a graph layout problem that is known for its difficulty even on small graph classes. The bandwidth problem of an arbitrary graph is known to be NP-complete. It is NP-complete even for a tree of maximum degree 3. Bandwidth problem is amongst the most studied graph layout problems and intuitively, it seems to be harder to solve than various other graph layout problems like pathwidth and vertex separation. Bandwidth is a benchmark problem known for its difficulty among the often studied NP-hard graph problems.

Surprisingly, to our knowledge, few polynomial time algorithms for exact computation of bandwidth are known only for caterpillars of hair length at most 2, chain graphs, cographs, interval graphs and bipartite permutation graphs. The following relationship between these classes of graphs is known : Bipartite Permutation Graphs $\subset$ Biconvex Bipartite Graphs $\subset$ Convex Bipartite Graphs $\subset$ 2-directional Orthogonal Ray Graphs $\subset$ Chordal Bipartite Graphs. In the above mentioned family, bandwidth problem is NP-complete for convex bipartite graphs and all its super classes. Polynomial time algorithm have been given for bipartite permutation graphs. But for Biconvex Bipartite graph, no standard result has been proposed.

The NP-completeness of bandwidth problem in graphs has been traditionally proved using multiple processor scheduling problem. The multiple processor scheduling problem is known to be strongly NP-complete and is reduced to the bandwidth problem in certain graph classes and hence the latter is also proved to be NP-complete. In this thesis, we propose that bandwidth problem is not NP-complete for biconvex bipartite graphs. This result, though, cannot be accepted as a formal proof as we only prove that NP-Completeness cannot be proved by using the traditional method of multiple processor scheduling problem reduction to bandwidth problem in case of biconvex bipartite graphs. We see that polynomial time algorithms have been given

for Bipartite Permutation graph which is the nearest established subset of biconvex bipartite graphs. We try to extend a similar approach for biconvex bipartite graphs. We expect to obtain a polynomial time algorithm though it is still under development.

Contents

Chapter 1

Introduction

A linear layout of a graph G with vertex set $V(G)$ is a bijection $\pi : V(G) \rightarrow \{1, 2, \ldots, |V(G)|\}$. When we are given a graph G and integer k, bandwidth problem checks whether there exists a linear layout of the vertices of G such that no edge of G has its endpoints mapped to positions at distance more than k. The bandwidth of a graph is the maximum bandwidth of a connected component. So, we consider only connected graphs. Papadimitriou [1] showed that the bandwidth problem is NP-Complete for general graphs. Monien [2] showed that it is NP-complete even for caterpillars of hair length at most 3, which is a very special tree. This proves that bandwidth problem is NP-Complete for chordal bipartite graphs.

The following relationship of the graph classes is known: Bipartite Permutation Graphs $\subset$ Biconvex Bipartite Graphs $\subset$ Convex Bipartite Graphs $\subset$ 2-directional Orthogonal Ray Graphs $\subset$ Chordal Bipartite Graphs. But for the above family, Heggernes et al [5] recently proposed a polynomial time algorithm that computes bandwidth of bipartite permutation graphs in $O(n^4\ logn)$ time. They have used a new approach to solve the bandwidth problem. They do not use the previously known algorithms or techniques. They characterize bipartite permutation graphs based on vertex ordering called strong ordering. But in our survey, this proved to be a complicated approach. But, Uehara [8] proposed a faster algorithm for the same problem shortly afterwards by extending known results for interval graphs and chain graphs making use of a new characterization of bipartite permutation graph based on graph decomposition. Uehara's algorithm solves the bandwidth problem in $O(n^2\ logn)$ time.

1.1 Some Applications of Bandwidth Problem

Bandwidth problem is extensively used in the sparse matrix computations. Sparse matrix computation has significant importance mainly in engineering applications as practically used matrices arising in those fields are usually sparse. The problem is to decide if there is a permutation matrix P such that PAP^T is a matrix with all non-zero entries on the central diagonal or on the k diagonals on either side of the central diagonal. Standard matrix operations like inversion and multiplication and also Gaussian elimination can be sped up considerably if the input matrix A is transformed into a matrix PAP^T of small bandwidth. The graph version and the matrix version of the bandwidth problem are equivalent, and both of them are being studied extensively. In this thesis, we study the graph version only of certain graph classes.

Also, some hard problems become efficiently solvable when restricted to particular graph classes. This is a very common approach to solve hard problems; for example, the class of interval graphs was first studied by molecular biologists, and it was found that many hard problems could be solved efficiently on these graph classes. On the other hand, these algorithms reveal and make use of graph theoretical properties of the graph classes.

Chapter 2

Preliminaries

In this chapter we discuss the basic definitions and theorems in graph theory and computational complexity. In section 2.1, we discuss the basic definitions in graph theory. In section 2.2 we discuss about the computational complexity classes P and NP. In Section 2.3, we introduce and define the bandwidth problem.

2.1 Graph Theoretic Preliminaries

Definition 1. *A graph G = (V, E) consists of a finite set of vertices V(G) and a collection E(G) of edges. An undirected edge is a pair of distinct vertices* $(u, v) \in V(G)$*, and is denoted by uv. We say that the vertex u is adjacent to the vertex v if there is an edge* $uv \in E(G)$*.*

Let S be a set of vertices of a graph G. Then, the cardinality of the set S is denoted by $|S|$ and the subgraph of G induced by S is denoted by $G[S]$. The set $\mathrm{N}(v) = \{u \in V(G)\colon\ uv \in E(G)\}$ is called the neighbourhood of the vertex $v \in V$ in G, the set $\mathrm{N}[v] = \mathrm{N}(v) \cup \{v\}$ is called the closed neighbourhood of the vertex $v \in \mathrm{V(G)}$. Given a graph $G(V, E)$, its compliment $\overline{G} = (V, \overline{E})$ is defined by $\overline{E} = \{\{uv\} | \{u, v\} \notin E\}$. A vertex set I is called independent if $G[I]$ does not contain edges.

Clique is a set of vertices such that any pair of vertices in that set is connected by an edge. For a graph $G = (V, E)$, a sequence of distinct vertices $\{v_0, v_1, \ldots, v_l\}$ is a path, denoted by $(v_0, v_1, \ldots, v_l)$, if $\{v_j, v_{j+1}\} \in E$ for each $0 \leq j < l$. The length of

a path is the number of edges on the path. For two vertices u and v, the distance of the vertices which is denoted by dist(u, v), is the minimum length among the paths joining u and v.

Definition 2. *A caterpillar is a tree in which all the vertices of degree greater than one are contained in a single path called a body. An edge incident to a vertex of degree one is called a hair. A generalized caterpillar is a tree obtained from a caterpillar by replacing each hair by a path. A path replacing a hair is also called a hair.*

Definition 3. *A graph G = (V, E) is bipartite iff V can be partitioned into two sets X and Y such that every edge joins a vertex in X and the other vertex in Y. Now let G(V, E) be a bipartite graph with bipartition X and Y. An ordering $\prec$ of X is said to fulfil the adjacency property if for each $y \in Y$, the set of neighbours of y consists of vertices that are consecutive in the ordering $\prec$ of X.*

A bipartite graph G is considered to be *chordal* if G does not contain induced cycles of length greater than 4 i.e, a graph is *chordal* if each of its cycles with four or more nodes has a chord (an edge joining two nodes that are not adjacent in the cycle). We see that trees are chordal bipartite graphs by definition.

Definition 4. *G is said to be convex if there exists an ordering of X that fulfils the adjacency property. G is said to be biconvex if there exists an ordering of X and an ordering of Y as well that fulfil the adjacency property.*

Definition 5. *A biconvex graph is called a chain graph if and only if it has a vertex ordering of X such that $N(x_n) \subseteq N(x_{n-1}) \subseteq \ldots \subseteq N(x_1)$. We can also sort vertices of Y as $N(y_1) \subseteq N(y_2) \subseteq \ldots \subseteq N(y'_n)$ by this property.*

Definition 6. *A graph G with vertex set V(G)= $\{v_0, v_1, ..., v_n\}$ is said to be a permutation graph if there exists a pair of permutations π_1 and π_2 on $N = \{1, 2, ..., n\}$ such that for all $i, j \in N$, $(v_i, v_j) \in E(G)$ if and only if $(\pi_1^{-1}(i) - (\pi_2^{-1}(j)) \cdot (\pi_2^{-1}(i) - \pi_1^{-1}(j)) < 0$.*

Bipartite permutation graphs are those which are both bipartite and permutation graph.

Intuitively, we see that each vertex v in a permutation graph corresponds to a line l_v joining two points on a pair of parallel lines L_1 and L_2, which is called *line representation*. Then, two vertices v and u are adjacent if and only if the corresponding lines l_v and l_u are crossing. Vertex indices give the ordering of the points on L_1, and the permutation of the indices gives the ordering of the points on L_2.

Definition 7. *A graph G = (V, E) with V(G)= $\{v_0, v_1, ..., v_n\}$ is called an interval graph if there is a finite set of intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, ..., I_{v_n}\}$ on the real line such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \phi$ for each i and j with $0 < i, j \leq n$.*

We call the set $\mathcal{I}$ of intervals an interval representation of the graph. For each interval I, we denote the left and right endpoints by *L(I)* and *R(I)* respectively i.e, I = [L(I), R(I)]. For any interval representation $\mathcal{I}$ and a point p, N[p] denotes the set of intervals which contain the point p. An interval representation is called proper *iff* $L(I) \leq L(J)$ and $R(I) \leq R(J)$ for every pair of intervals I and J or vice versa. An interval graph is proper if and only if it has a proper interval representation. That is any proper interval graph will have a proper interval representation which consists of intervals of one unit length.

Definition 8. *A graph G = (V, E) is called a threshold graph if there exists non negative weights w(v) for $v \in V$ and t such that $\{u, v\} \in E$ iff $w(u) + w(v) \geq t$.*

Definition 9. *A Helly family of order k is a family of sets such that any minimal subfamily with an empty intersection has k or fewer sets in it. The k- Helly property is the property of being a Helly family of order k.*

2.2 Computational Complexity

Computational complexity theory classifies computational problems according to their difficulty, and relates those classes to each other. The complexity class P is seen as

mathematical abstraction modelling of those computational problems that admit an efficient algorithm. This hypothesis is called the CobhamEdmonds thesis. The complexity class NP, on the other hand, contains many problems that needs to be solved efficiently, but for which no efficient algorithm has been found.

The theory of NP-Completeness is designed to be applied to decision problems only. Before formally introducing the notion of NP-Completeness, let us briefly discuss about decision and optimization version of a problem. Decision problems have only two possible solutions, either the answer is *yes* or the answer is *no*. If the optimization problem asks for a structure of certain type that has minimum cost among all such structures, we can associate with that problem,the decision problem that includes a numerical bound k as an additional parameter and asks whether there exists a structure of the required type having cost no more than k. Let us illustrate with an example. Vertex Cover is a well known graph theoretic problem. The problem is the following:

Definition: A vertex-cover of an undirected graph $G = (V, E)$ is a subset of V' of V such that $\forall(u, v) \in E$ either u or v (or both) belongs to V' .

Problem: Finding minimum size vertex cover in a given undirected graph.

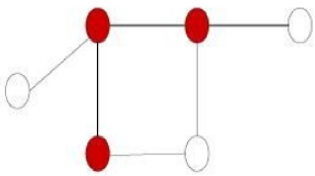


Figure 2.1: The vertices shown in dark shade constitute the minimum vertex cover

The *optimization version* of this problem is: What is the size of minimum vertex cover for a given graph G?

The *decision version* of this problem is: Does there exist a vertex cover of size k for a given graph G, where k is a fixed integer?

Decision problems can be derived from maximization problems in an analogous

way, simply by replacing “no more than” by “at least”.

2.2.1 NP-Hardness and NP-Completeness

The question of whether P equals NP is one of the most important open questions in theoretical computer science because of the wide implications of a solution. If the answer is yes, many important problems can be shown to have more efficient solutions. These include various types of integer programming problems in operations research, many problems in logistics, protein structure prediction in biology, and the ability to find formal proofs of pure mathematics theorems. The P versus NP problem is one of the Millennium Prize Problems proposed by the Clay Mathematics Institute. There is a US$1,000,000 prize for resolving the problem.

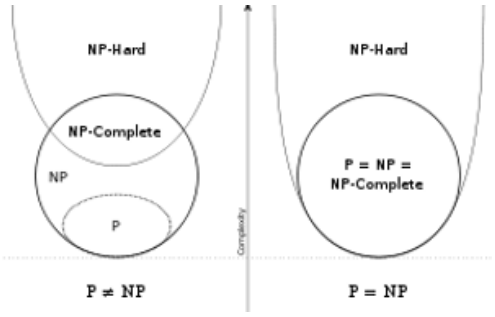


Figure 2.2: P, NP-complete, NP-hard for different set of problems

2.2.2 Approximation Algorithms

In this section, we look at some basic definitions related to approximation algorithms. Approximation algorithms are algorithms used to find approximate solutions to optimization problems. It is usually associated with NP-hard problems as it is unlikely that there can be efficient polynomial time exact algorithms for NP-hard problems. If we know that a problem is NP-hard then we can work on the problem in the following ways:

1. Restricted class of Inputs: Here the input is restricted to a subset of the original class. Sometimes a problem can be solved in polynomial time if we restrict

the input to subset of the original class. For example minimum vertex cover problem is NP-complete on general graphs. If we restrict our input to bipartite graphs this can be solved in polynomial time.

2. Approximation Algorithms: Here we design an algorithm which finds an approximate solution with a small error bound instead of finding the exact solution. In this method, we compromise on the exactness of the solution to reduce the complexity.

2.3 Bandwidth of a graph

For a graph G, a layout (or vertex ordering) β is a bijective function $\beta : V(G)\{1, ..., n\}$ to V . We also write β as $\langle\beta(1), ..., \beta(n)\rangle$. For a vertex pair u, v of G, the distance between u and v in β is $d_\beta(u, v) = |\beta^{-1}(u)-\beta^{-1}(v)|$. We write $u \preceq_\beta v$ if $\beta^{-1}(u) \leq \beta^{-1}(v)$ and $u \prec_\beta v$ if $\beta_u < \beta_v$. The leftmost and rightmost vertex in β are respectively $\beta(1)$ and $\beta(n)$. For an integer k ≥ 1, we call β a k-layout for G if for every edge uv of G, $d_\beta(u, v) \leq k$.

Definition 10. *A layout of a graph G = (V, E) is a bijection π between the vertices in V and the set $\{1, 2, ..., |V|\}$.*
The bandwidth of π is defined as $b_\pi(G) = max|\pi(u) - \pi(v)||(u, v) \in E(G)$.
The bandwidth of G is defined as b(G) = min $b_\pi(G)$ where π ranges over all linear layouts of G.

A linear layout π of G is said to be optimal if $b_\pi(G) = b(G)$. Given a graph G and an integer k, the bandwidth problem asks whether the bandwidth of G is at most k. Also, the layout which achieves $b(G)$ is called as *optimal layout.* Since the bandwidth of a graph is the maximum bandwidth over all its connected components, we shall consider only connected graphs.

2.3.1 Multiple Processor Scheduling Problem

The problem statement is: *"Given a set J of jobs where job j_i has length l_i and a number of processors m_i, what is the minimum possible time required to schedule all*

jobs in J on m processors such that none overlap?"

The multiprocessor scheduling problem is a well known strong NP-Complete problem. We have discussed that the bandwidth problem is NP-complete for general graphs and even for very specialized graphs like caterpillar of hairlength at most 3. The NP-completeness of bandwidth problem is proved by reducing a multiple processor scheduling problem to the bandwidth problem. Monien [2] reduced the bandwidth problem for caterpillars with hairs of length at most 3 and then Anish et al., [3] developed this for convex trees and then for convex bipartite graphs.

Chapter 3

The NP-Complete Family

In this chapter, we study about the bandwidth problem in some of the graph classes for which it is NP-Complete. In Section 3.1, we discuss about caterpillars with hairs of length at most 3. In section 3.2 and 3.3, we study about convex trees and convex graphs respectively.

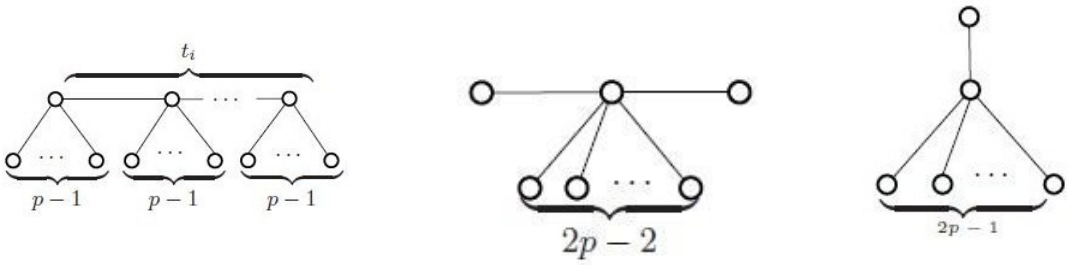
3.1 Bandwidth problem of caterpillars with hairs of length at most 3

Theorem 1. *The bandwidth minimization problem for caterpillars with hairs of length at most 3 is NP-complete.*

We will discuss the bandwidth problem for caterpillars of hairlength at most 3 in detail though we are not going to give the entire proof. Monien [2] reduced the bandwidth problem for caterpillars with hairs of length at most 3. Given a set $T = \{t_1, t_2, ..., t_n\}$ of tasks (the i^{th} task in T has execution time t_i), a deadline D, and a number m of processors, we construct the required graph and an integer k such that C has bandwidth k if and only if the tasks in T can be scheduled on the m processors to satisfy the deadline D. Here we can assume that all the t_i are polynomially bounded in n as the multiple processor scheduling problem is strong NP-complete.

We construct two portions of the caterpillar called "barrier" and "turning point" as shown in Figure 3.1. Here, the barrier of height p and the turning point of height

p both have bandwidth p. Note that, in every optimal layout of the turning point both nodes a and g either belong to the first half of the layout or the second half of the layout, i.e., in every optimal layout of the turning point the backbone has to be folded. Let T_p denote the turning point of height p. T_p has exactly $6p+1$ nodes.



(a) T_i corresponding to task t_i (b) p-barrier (c) Turning point of height p

Figure 3.1: [3] Components of the caterpillar

Lemma 1. *Let T_p = (V, E), let $\sigma : V \to \{1, ..., 6p+1\}$ be a layout with $|\sigma(i)-\sigma(j)| \leq p$ for all $i, j \in E$ and let $p \geq 4$. Then either $\sigma(a), \sigma(g) < 3p+1$ or $\sigma(a), \sigma(g) > 3p+1$*

.

The instance $Y = (t_1, t_2, ..., t_n, D, m)$ of the multiple processor scheduling problem is associated with caterpillar C as shown in the Figure 3.2. We only consider instances Y with $\Sigma_{i=1}^{n} t_i = D.m$. The multiple processor scheduling problem is NP-complete when restricted to instances of this class also.(Monien [2])

Each task t_i is represented by a caterpillar T_i. Each processor i is represented by a chain P_i of length $D-1$. C is constructed from these components as shown in Figure 3.2. Task caterpillars T_i and T_{i+1} are separated by a chain L_i of length Δ. Processor chains P_i and P_{i+1} are separated by a (p+1)-barrier B_i. A turning point of height p+2n+1 separates the upper task portion and the lower processor portion. A (p + 2n + 1)-barrier B_0 is attached to the left of P_1. The behaviour of the turning point

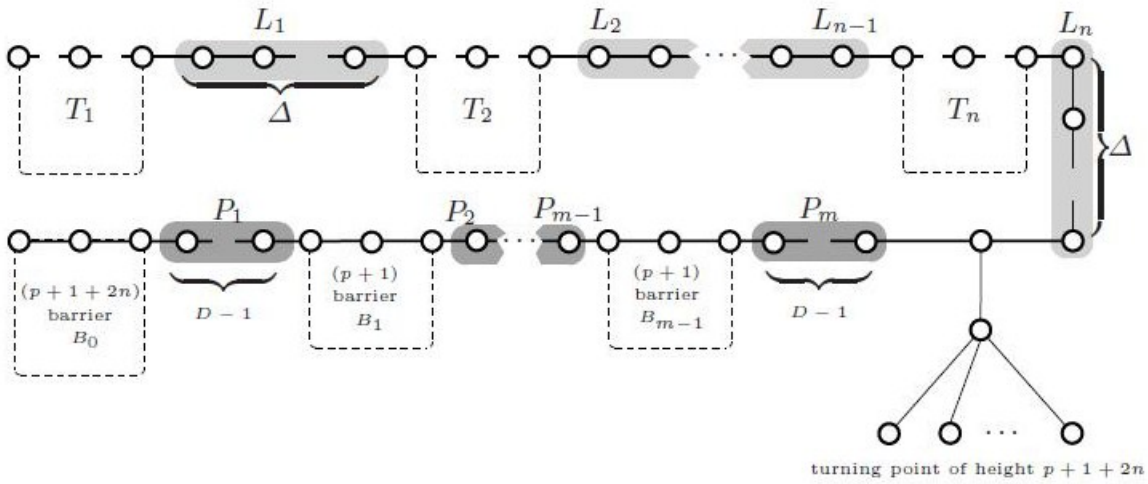


Figure 3.2: [3] The reduction from Multiple Processor Scheduling Problem

will force every layout with bandwidth p+2n+1 to place the two parts one upon the other. A layout with bandwidth p+2n+1 exists if and only if each of the holes can be filled by using all the nodes of the blocks encoding certain execution times which is equivalent to the instance of Y having a solution. Each task consists of the chain of length t_i and the 'ground line' consists of $\lambda = m$(D+2)+1 nodes

If there exists a scheduling of the tasks in T such that tasks $t_{i_1}, t_{i_2}, ..., t_{i_j}$ are assigned to processor i, then C has bandwidth k and an optimal layout can be achieved by:

(a) laying out the vertices of the body of $T_{i_1}, T_{i_2}, ..., T_{i_j}$ between barriers B_{i-1} and B_i (between B_{m-1} and turning point, for i = m) and

(b) laying out the vertices of B_0 at the extreme left and those of the turning point at the extreme right.

Conversely, if C has bandwidth k, then in any optimal layout of C,

(a) the turning point must be laid out at one of the extreme ends, and barrier B_0 must be laid out at the other,

(b) all the vertices of the body of each T_j must be laid out between two barriers B_i and B_{i+1} for some i (or B_{m-1} and the turning point for i = *m- 1*), and

(c) for each i, if between B_i and B_{i+1} (or between B_{m-1} and turning point for i = *m-1*), bodies of $T_{i_1}, T_{i_2}, ..., T_{i_j}$ are laid out, then $t_{i_1} + t_{i_2} + \ + t_{i_j} < D$.

This gives us a scheduling of the tasks in T .

Lemma 2. *If Y has a solution then C has bandwidth p+2n+1.*

Lemma 3. *If p > 2n·(d+4) and if C has bandwidth p+2n+1, then Y has a solution.*

For the proof of the above lemmas, please refer to Monien's paper. The proof of Theorem 1 follows from the above lemmas.

3.2 Bandwidth of Convex trees

Theorem 2. *The bandwidth problem is NP-complete for convex trees.*

The proof is almost same as that of Theorem 1 with a slight modification.
We will use the same construction of C as discussed in the previous section of caterpillar with hairs of length at most 3. If we remove from C the degree-1 vertices of the turning point, the remaining tree is a caterpillar. It is easy to see that a caterpillar is biconvex, and therefore both partitions of C have an ordering satisfying the adjacency property. If we restore the degree-1 vertices, irrespective of their positions in the ordering of the partition to which they belong, they do not disturb the adjacency property of the ordering of the other partition. Thus C is a convex tree. If we set the values of Δ and p such that $\Delta = 2{\cdot}(m(D+2) - 2)$ and $p > 2n(D+4)$, C can be constructed in time polynomial in n, m, and D; and it can be shown that the tasks in T can be scheduled on the m processors if and only if C has a bandwidth of $k = p + 2n + 1$. In fact, this proof is exactly the same as the proof of Theorem 1, except only for a slight difference in the structure of the turning point.

3.3 Bandwidth of Convex Bipartite Graphs

Theorem 3. *The bandwidth problem is NP-complete for Convex bipartite graphs.*

We have seen that the bandwidth problem is NP-complete for caterpillars with hairs of length at most 3, which are very special trees. This implies that it is NP-complete for chordal bipartite graphs. In this section, we see that it is NP-complete for convex bipartite graphs as well which is a subclass of chordal bipartite graphs. We also discuss an $O(n)$-time, 4-approximation algorithm. In the previous section, the bandwidth problem has been proved to be NP-complete for convex trees which is a subclass of Convex bipartite graphs. We will discuss the algorithm in detail in this section.

Let G be a convex bipartite graph with bipartition *(X, Y)* and an ordering $\prec$ of X satisfying the adjacency property with $X = \{x_1, x_2, ..., x_{|X|}\}$ and $x_1 \prec ... \prec x_{|X|}$. Assume $Y = \{1, 2, \ldots, |Y|\}$. Define mappings $s : Y \to \{1, 2, ..., n\}$ and $l : Y \to \{1, 2, ..., n\}$ such that for $y \in Y$, $x_{s(y)}$ and $x_{l(y)}$ are, respectively, the smallest and largest vertices in $\prec$ adjacent to y. For each vertex $y \in Y$, let m(y) = $\lceil (s(y)+l(y))/2 \rceil$.

Algorithm

The algorithm takes G as input along with the mappings s and l and outputs a linear layout π of G. The general idea of the working of the algorithm is that it takes the vertices of X in the same order as they appear in $\prec$ and insert vertices of Y in between them. While inserting the vertices of Y between the vertices of X, the algorithm follows the property that for each $y \in Y$, $\lfloor |N(y)|/2 \rfloor$ vertices of the set *N(y)* of its neighbours are onto its left and the remaining to its right.

Algorithm starts by computing *m(y)* for each vertex of Y and sorting the vertices according to their *m(i)* values (Lines 1 and 2). It incrementally assigns labels to the vertices of X in the order in which they appear in $\prec$; stopping at each x_j to check whether there is a vertex in y with *m(y)* value equal to j, in which case it assigns the current label to y. The process is repeated until all vertices have been labelled (Lines 3 through 8).
We shall now analyse the performance of the algorithm. Consider a layout π output

by the algorithm.

Algorithm 1 Algorithm to compute bandwidth of convex graph [3]

1. Compute $m(i)$ for each vertex $i \in Y$. Add a dummy vertex $|Y| + 1$ to Y with $m(|Y| + 1) = |X| + 1$.
2. Let $\sigma(1), \ldots, \sigma(|Y + 1|)$ be the vertices of Y sorted in the non-decreasing order of $m(i)$ value, where σ is a permutation on $\{1, \ldots, |Y| + 1\}$.
3. Initialize $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$.
4. **while** $(j \leq |X|)$
5. **if** $j < m(\sigma(i))$
6. $\pi(x_j) = k; j \leftarrow j + 1; k \leftarrow k + 1$.
7. **else if** $j = m(\sigma(i))$
8. $\pi(\sigma(i)) = k; i \leftarrow i + 1; k \leftarrow k + 1$.
9. return π

Lemma 4. *Algorithm 1 preserves the ordering $\prec$ of X, i.e,*

$$\pi(x_1) < \pi(x_2) < \cdots < \pi(x_{|X|})$$

Proof: This is easy to see as the vertices of Y are just inserted in between vertices of X and the order of x is not changed. □

For $y \in Y$, we define G_y as the subgraph of G induced by the vertices in

$$V_y = \{v|\pi(x_{s(y)}) \leq \pi(v) \leq \pi(y)\} \cup \{v|\pi(y) \leq \pi(v) \leq \pi(x_{l(y)})\}.$$

The *diameter* of a graph is the least integer k such that the shortest path between any pair of vertices is at most k.

Lemma 5. *For any $y \in Y$, the diameter of G_y is at most 4.*

Proof. Consider G_y corresponding to some $y \in Y$. Let u, v be a pair of vertices of V_y. Now we will divide the proof for 3 cases.
Case 1: Both $u, v \in V_y \cap X$.
By the construction of V_y and Lemma 1, for some $s(y) \leq i, j \leq l(y)$, u must be x_i and v must be x_j. This implies that both u and v are adjacent to y. Hence the distance between u and v is 2. The 2-distance path is $u \rightarrow y \rightarrow v$.
Case 2: $u \in V_y \cap X$ and $v \in V_y \cap Y$.
Here, vertex v should be adjacent to at least one vertex u' in $V_y \cap X$. If not , then it contradicts the assumption that the algorithm placed v between $x_s(y)$ and y or between y and $x_l(y)$ (by the construction of V_y). So v is adjacent to u'. Now, if u' is u, then the distance between them is 1. Else, both u and u' will be at a distance of 2 as they both lie in X (proved in the previous case). This makes the distance between u and v to be 3.
Case 3: Both $u, v \in V_y \cap Y$.
From case 2 we can see that u must be adjacent to u' and v must be adjacent to v', where $u', v' \in V_y \cap X$. And according to the construction of V_y, both u and v are adjacent to y. And from case 2, the shortest path is $u \rightarrow u' \rightarrow y \rightarrow v' \rightarrow v$. So the distance between u and v is at most 4.
Any pair of vertices u and v will fall in one of the above cases and hence the lemma is proved. □

Assmann et. al [7] have proved the following.

Lemma 6. *For a graph $G, b(G) \geq max\lceil (N' - 1)/D' \rceil$, where the maximum is taken over all connected subgraphs G' of G, N' is the number of vertices of G', and D' is the diameter of G'.* □

We will now show the approximation ratio of the algorithm.

Lemma 7. *For layout π returned by the Algorithm, $b_\pi(G) \leq 4 \times b(G)$.*

Proof. Here π is the layout given as output by the algorithm. Let *(x, y)*, $x \in X, y \in Y$ be an edge of G so that $|\pi(x) - \pi(y)| = b_\pi(G)$. Now consider V' be the set of vertices v such that v lies between x and y in π. It is easy to see that $b_\pi(G) = |V'| - 1$. But from Lemma 5 and Lemma 6, we get $b(G) \geq \lceil (|V_y| - 1)/4 \rceil$. So we have:

$$\frac{b_\pi(G)}{b(G)} \leq \frac{|V'| - 1}{(|V_y| - 1)/4}$$

. Also we can see that, x must be $x_{s(y)}$ or $x_{l(y)}$ from the way we have constructed V', and therefore $V' \subseteq V_y$. This completes the proof and we get:

$$\frac{b_\pi(G)}{b(G)} \leq 4.$$

□

Now it is left to discuss the computational complexity of the approximation algorithm which we have discussed. We can see that step 1 in the algorithm runs in $O(|Y|)$ time and the while loop in the step 4 runs in $O(|X|)$ time while the other steps are constant. So we can conclude that the above algorithm runs in $O(|X|+|Y|)$ time. So, the above statement and Lemma 7 implies that:

Theorem 4. *Algorithm 1 computes a linear layout of a convex bipartite graph G whose partitions are X and Y in $O(|X| + |Y|)$ time such that $b_\pi(G) \leq 4 \times b(G)$.* □

Chapter 4

The polynomial time family

The bandwidth problem can be solved in polynomial time for a very few graph classes. In this chapter we study 4 such graph classes. In section 4.2 and 4.3, linear time algorithms are discussed for threshold graphs and chain graphs respectively. In section 4.4, $O(n^2)$ time algorithm is given for solving bandwidth problem in bipartite permutation graph. In section 4.5, we discuss the recent result given by Anish et al., [3] regarding biconvex trees. The algorithms used in threshold and chain graphs help us in the development of the algorithm for bipartite permutation graph. This has been done by Uehara[8] which is an improvement of the result given by Heggernes, Kratsch, and Meister[5]. The latter used a complicated yet new approach to solve the bandwidth problem of graphs. Uehara, on the other hand, extended known results for interval graphs and chain graphs and applied it to bipartite permutation graphs based on a particular graph decomposition which we will discuss later in the chapter.

4.1 Some results on Interval Graphs

The following is a known proper inclusion:

Lemma 8. *[4] (1) Threshold graphs $\subset$ interval graphs $\subset$, (2) Chain Graphs $\subset$ bipartite permutation graphs.*

For a graph $G = (V, E)$, a *proper interval completion* is a subset E' of E such that $G' = (V, E')$ is a proper interval graph. We will deal with only proper interval completions. So, we say a completion E' is *minimum* if and only if $|C'| \leq |C''|$ where

C' and C" are maximal cliques in $G' = (V, E')$ and $G'' = (V, E'')$ respectively where E'' is another completion.
We have seen that every proper interval graph has a proper interval representation that consists of unit interval length. We can also say that the ordering is unique and hence the following proposition:

Proposition 1. *For a proper interval graph G = (V, E) there is a unique ordering (up to reversal)* $v_1, v_2, \ldots, v_n$ *of n vertices such that G has a unique proper interval representation* $\mathcal{I}$ *such that* $L(I_{v_1}) < L(I_{v_2}) < \ldots < L(I_{v_n})$.

Kaplan and Shamir [10] have shown that for a graph G and its minimum completion E', $b(G) = |C'| - 1$. Here C' is the maximal clique in $G' = (V, E')$. For an interval graph $G = (V, E)$ with interval representation $\mathcal{I} = \{I_{v_1}, I_{v_2}, \ldots, I_{v_n}\}$. For each maximal clique C, there exists a point p such that N[p] induces clique C by Helly property. Thus, we can compute $b(G)$ by the following algorithm for a given graph G;

Input : Graph $G = (V, E)$
Output: $b(G)$

1. generate a proper interval graph $G' = (V, E')$ that gives a minimum completion of G.
2. make a unique interval representation $\mathcal{I}$ of G';
3. find a point p such that $|N[p]| \geq |N[p']|$ for any point p' on $\mathcal{I}(G')$;
4. **return** $(|N[p]| - 1)$.

The following has been observed by Kaplan and Shamir [10]:
Observation 1. *If* $G' = (V, E')$ *is a minimum completion of G = (V, E), let* $\mathcal{I}(G') = (I_{v_1}, I_{v_2}, \ldots, I_{v_n})$ *be the unique proper interval representation of* G' *as seen in Proposition 1, then the ordering* $v_1, v_2, \ldots, v_n$ *an optimal layout of G, and vice versa.*
The following lemma on proper interval subgraphs of an interval graph will play an important role in our results:

Lemma 9. *Let G = (V, E) be an interval graph with* $V = \{v_1, v_2, \ldots, v_n\}$, *and* $\mathcal{I} = \{I_{v_1}, I_{v_2}, \ldots, I_{v_n}\}$ *as an interval representation. Let* $\mathcal{J} = \{J_{u_1}, J_{u_2}, \ldots, J_{u_k}\}$ *be a subset of* $\mathcal{I}$ *such that* $\mathcal{J}$ *forms a proper interval representation. Let* ρ *be the*

injection from $\mathcal{J}$ to $\mathcal{I}$ with $J_{v_i} = I_{v_{\rho(i)}}$ for each $1 \leq i \leq k \leq n$. Then G has an optimal layout π such that each interval J_{u_i} appears according to the ordering in $\mathcal{J}$ i.e, $\pi(I_{v\rho(i)}) < \pi(I_{v_{\rho(i+1)}})$.

Proof. The idea of the proof is based on algorithm proposed by Kleitman and Vohra [10]. We will assume that the algorithm given by Kleitman and Vohra receives as inputs the interval representation $\mathcal{I}$ and b(G) and outputs the optimal layout π that achieves $b(G)$.

4.2 Bandwidth of Threshold Graphs

In this section, we study, in detail, the linear time algorithm which computes $b(G)$ of a threshold graph G. We have seen that, in a threshold graph there exists non-negative weights *w(v)* for each $v \in V$ and t so that $\{u, v\} \in E$ *iff* $w(u) + w(v) \geq t$. Here, we assume that G is a connected graph and V is ordered as $\{v_1, v_2, \ldots, vn\}$ with $w(v_i) \leq w(v_{i+1})$ for $1 \leq i < n$. Note that this sorting can be done in *O(n)* time by bucket sort using degrees of the vertices. We can find l such that $w(v_{l-1}) + w(v_l) < t$ and $w(v_l) + w(v_{l+1}) \geq t$ in *O(n)* time. Based on this construction, we can see that G has the following interval representation $\mathcal{I}(G)$:

- v_i corresponds to the point i for $1 \leq i \leq l$. Here, $I_{v_i} = [i, i]$.
- v_i corresponds to the interval $[j, l]$ for $l < i \leq n$. Here j is the minimum index with $w(v_i) + w(v_j) \geq t$.

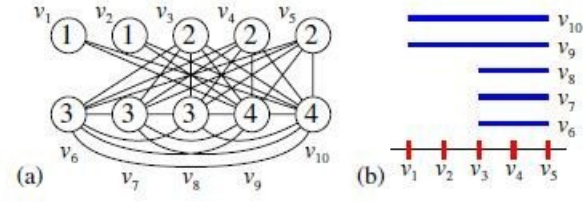


Figure 4.1: [8](a)Threshold graph and (b) the interval representation

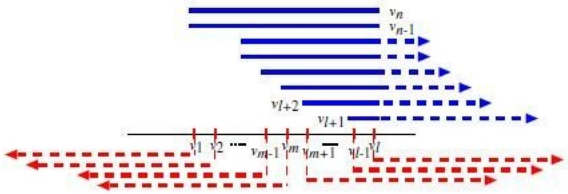


Figure 4.2: [8] Minimum completion construction

An example is given to have a better understanding of a threshold graph and its interval representation. In the example given in Figure 4.1(a), the threshold is assigned as 5 and the weight is the number given the circle. We give the interval representation in Figure 4.1(b).

Theorem 5. *When an interval representation $\mathcal{I}(G)$ is given as discussed above for a graph G = (V, E), the bandwidth b(G) can be computed in O(n) time.*

Proof. Note that $L(I_{v_i}) < L(I_{v_{i+1}})$ and $R(I_{v_i}) < R(I_{v_{i+1}})$ for each i such that $1 \leq i < l$, and $L(I_{v_i}) \geq L(I_{v_{i+1}})$ and $R(I_{v_i}) = R(I_{v_{i+1}}) = l$ for each i with $l < i < n$. Basically, G consists of two proper interval graphs induced by $\{v_1, v_2, \ldots, v_l\}$ and $\{v_l, v_{l+1}, \ldots, v_n\}$. The proper interval representations appear in $\mathcal{I}(G)$ as well. Therefore using lemma 9, an optimal layout π of $V = \{v_1, \ldots, v_n\}$ exists such that $\pi(v_1) < \pi(v_2) < \ldots < \pi(v_l)$ and $\pi(v_l) > \pi(v_{l+1}) > \ldots > \pi(v_n)$. So, an optimal layout is obtained by merging the two sequences of vertices.

We now use Observation 1 to obtain an optimal layout by constructing a minimum completion of G from the two sequences. $[L(I_{v_n}), R(I_{v_n})] = [1, l]$ is the longest interval as G is connected. So we extend all the intervals, except I_{v_n}, to length *l - 1* and construct a minimum completion. We denote each extended interval I_{v_i} by I'_{v_i} . The extension of intervals for $i > l$ is straightforward from Figure 4.2. We just extend them to right. Here, the maximum clique size is not increased. So, we now concentrate on the points $I_{v_i} = [i, i]$ with $i \leq l$. They are also extended to I'_{v_i} with length *l - 1*. We see that I'_{v_i} contains either 1 or *l*. If it contains 1, set $R(I'_{v_i}) = 1$

whereas if it contains l we set $L(I'_{v_i})$.

Algorithm 2 Algorithm for computing bandwidth of threshold graph [8]

Input:Threshold graph $G = (V, E)$ with $w(v_1) \leq w(v_2) \leq \ldots \leq w(v_n)$ and t
Output:*b(G)*

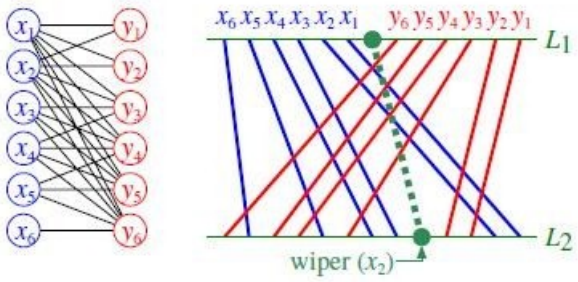
1. let l be the minimum index with $w(v_l) + w(v_{l+1}) \geq t$;
2. set $b(G) := \infty$.
3. **for** $m = 1, 2, \ldots, l-1$ **do**
4. set $lc := 0$;
5. **for** $i = 1, 2, \ldots, m$ **do**
6. let j be the minimum index with $w(v_l) + w(v_{l+1}) \geq t$;
7. **if** $lc < (m - i + 1) + (n - j + 1)$ then set $lc := (m - i + 1) + (n - j + 1)$;
8. **end**
9. **if** $max\{lc, n - m\} < b(G)$ then $b(G) := max\{lc, n - m\}$;
10. **end**
11. **return** (*b(G)* - 1).

Thus, the following proper interval representation (Fig. 4.2) gives a minimum completion of n intervals of length l - *1* for some m such that $1 \leq m < l$:
$L[I_{v_i}] = j$ if $i > l$. Here j is the minimum index with $w(v_i) + w(v_j) \geq t$;
$R[I_{v_i}] = i$, if $1 \leq i \leq m$, and
$L[I_{v_i}] = i$ if $m < i \leq l$.
Basically for the construction of of a minimum completion, we look for the index m that minimizes a maximum clique in the proper interval graph represented by above proper interval representation determined by m.
On the minimum completion, at each point i with $1 \leq i \leq l$, l distinct cliques C_i are induced. Now we consider the maximum clique of the corresponding proper interval graph for a fixed $m \in [1..l]$.
We see that $N[m+1] \subseteq \ldots \subseteq N[l]$ and hence $N[l]$ induces a maximum clique of size n - m at points in $[m+1, l]$ whereas at each point i in $[1, m]$, $N[i]$ induces a clique

that consists of $\{v_i, v_{i+1}, \ldots, v_m\}$ and $\{v_j, v_{j+1}, \ldots, v_n\}$. Hence, we have a clique of size $(m - i + 1) + (n - j + 1)$ for each point i in $[1, m]$.
Thus, for a fixed m, we compute $[1..m]$ and $[m + 1..l]$, compare them, and obtain the maximum. We then compute the minimum size of the maximum cliques for all m, which gives $b(G) + 1$. Hence, we can compute $b(G)$ by Algorithm 2. The correctness of Algorithm 2 follows from Observation 1, Lemma 9 and the above discussions. □

4.3 Bandwidth of Chain Graphs

Chain graphs are biconvex graphs that follow certain rules. We will discuss a linear time algorithm to solve the bandwidth problem in chain graphs. We are given a chain graph with vertex ordering $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_{n'}\}$ according to the neighbour inclusion $N(x_n) \subseteq N(x_{n-1}) \subseteq \ldots \subseteq N(x_1) = Y$ and $N(y_1) \subseteq N(y_2) \subseteq \ldots \subseteq N(y_{n'}) = X$. Every vertex $y \in Y$ stores two endpoints 1 and $d(y)$ such that $N(y) = \{x_1, x_2, \ldots, x_{d(y)}\}$, and each $x \in X$ stores two endpoints n' and $n' - d(x) + 1$ such that $N(x) = \{y_{n'}, y_{n'-1}, \ldots, y_{n'-d(x)+1}\}$.



(a) Chain Graph (b) Line representation

Figure 4.3: [8] A chain graph and its line representation

Figure 4.3 shows the intersection model of a chain graph; all $x \in X$ are the horizontal line segments with all left endpoints having the same coordinates, and all $y \in Y$ are the vertical line segments with all the top endpoints having the same coordinates. It also can be transformed to the line representation of a bipartite permutation graph in a natural way with the endpoints sorted as shown in the above figure.

Definition 11. *In a chain graph G = (X, Y, E), we define a supergraph* $H_i = (X \cup Y, E_i)$ *as follows: For* $1 \leq i \leq n-1$, *we take the set* $\{x_1, x_2, \ldots, x_i\} \cup N(x_{i+1})$. *We make a clique* C_i *using the above set. This clique is the supergraph.*

Lemma 10. *(1) Hi is an interval graph for each i.*

(2) $b(G) = min_i\ b(Hi)$.

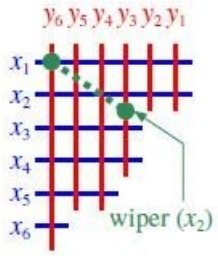


Figure 4.4: [8] Interval representation of a chain graph

We can see that H_i is a threshold graph that has an interval representation as shown in Figure 4.4 with x's as intervals and y's as points. Thus by Theorem 5 $b(H_i)$ can be computed in $O(n+n')$ time. We can hence also construct the minimum completions of H_i as discussed in the previous section.
We now introduce a new notation called $wiper(x_i)$. A wiper is a line segment joining two points p_1 on L_1 and p_2 on L_2 of the line representation as shown in Figure 4.3b. We choose p_1 and p_2 as follows: p_1 lies between x_1 and y'_n on L_1 and p_2 lies on L_2 between x_{i+1} and q. Here q is the lowest indexed neighbour among all $y \in Y$. In a way, the $wiper(x_i)$ cuts through the vertices of H_i if $N(x_i) \backslash N(x_{i+1}) \neq \phi$.

We see that the interval representation of H_i is a combination of the two interval representations of two threshold graphs because of which we can run the algorithm to find bandwidth of threshold graphs discussed in the previous section. We give the formal construction of H_i as given by Uehara [8]:

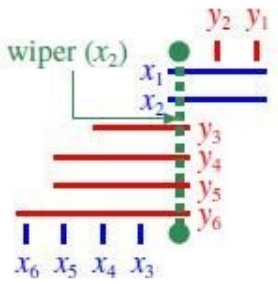


Figure 4.5: [8] Symmetric interval representation of chain graph

By Helly property, the intervals in the clique C_i share a common point 0, corresponding to wiper(x_i). For the point, we can construct a symmetric interval representation as in Figure 4.5 below:

(1) each $x'_i \in X$ with $i' \leq i$ corresponds to an interval $[0, (d(x'_i) - d(x_i))]$,

(2) each $x'_i \in X$ with $i' > i$ corresponds to the point $i - i'(< 0)$,

(3) each $y_j \in Y$ with $j > n' - d(x_{i+1})$ corresponds to an interval $[(i - d(y_j)), 0]$, and

(4) each $y_j \in Y$ with $j \leq n' - d(x_{i+1})$ corresponds to the point i - j + 1.

Let $X_i^R = \{x'_i \in X | i' \leq i\}, X_i^L = \{x'_i \in X | i' > i\}, Y_i^L = \{y_j \in Y | j > n' - d(x_{i+1})\}$, and $Y_i^R = \{y_j \in Y | j \leq n' - d(x_{i+1})\}$. $H_i[X_i^L \cup Y_i^L]$ and $H_i[X_i^R \cup Y_i^R]$ are the two induced subgraphs of H_i and are threshold graphs.

Theorem 6. *Given a chain graph G = (X, Y, E), we can compute b(G) in O(n + n') time.*

Proof. The outline of proof is fairly simple. From Lemma 10, we can compute *b(G)* by computing the minimum $b(H_i)$ for $i = 0, 1, \ldots, n-1$. We can compute $b(H_i)$ as the lemma also states that each H_i is a threshold graph and by Theorem 5, we can compute $b(H_i)$ in linear time.

Algorithm 3 Algorithm for computing bandwidth of Chain graph [8]

Input :Chain graph $G = (X, Y, E)$ with $N(x_n) \subseteq N(x_{n-1}) \subseteq \ldots \subseteq N(x_1)$. and $N(y_1) \subseteq N(y_2) \subseteq \ldots \subseteq N(y'_n)$.

Output:*b(G)*

1. $b := b(H_i)$ // by Algorithm 2;
2. **for** $i = 1, 2, \ldots, n-1$ **do**
3. construct the interval representation $\mathcal{I}(H_i)$ of the graph H_i with $wiper(x_i)$;
4. for $l = n, n-1, \ldots, i+1$ **do**
5. for $r = 1, 2, \ldots, n' - d({}_x i + 1)$ **do**
6. **if** $\max\{|RC_i(l,r)|, |CC_i(l,r)|, |LC_i(l,r)|\} < b$ **then**
7. $b = \max\{|RC_i(l,r)|, |CC_i(l,r)|, |LC_i(l,r)|\}$;
8. **end**
9. **end**
10. **end**
11. **return** (*b(G)* - 1).

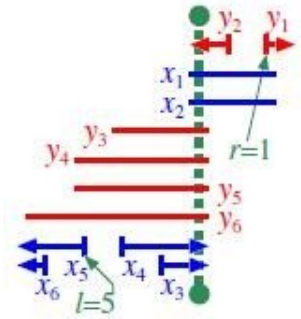


Figure 4.6: Example used in Theorem 6

The catch here is to combine the algorithm for the threshold graphs H_i. In threshold graphs, we have seen that we put point m during the construction of minimum completion. Here we have seen that $H_i[X_i^L \cup Y_i^L]$ and $H_i[X_i^R \cup Y_i^R]$ are the two in-

duced subgraphs of H_i are threshold graphs. So, we put two points l in $H_i[X_i^L \cup Y_i^L]$ and r in $H_i[X_i^R \cup Y_i^R]$. And now we make the proper interval representation using l and r as constructed by Uehara in [8].

(1) for each $x'_i \in X_i^L$ with $i' \geq l, I'_{x_{i'}} = [i-n, i-i']$

(2) for each $x'_i \in X_i^L with l < i', I'_{x_{i'}} = [i-i', 0]$

(3) for each $y_j \in Y_i^R$ with $r < j (\leq n' - d(x_{i+1})), I'^{y}_{j} = [0, i-j+1]$

(4) for each $y_j \in Y_i^R$ with $j - r, I'_{y_j} = [i-j+1, i]$

Figure 4.6 gives an example with $l = 5$ and $r = 1$. For every possible pair of *(l, r)* with $i+2 \leq l \leq n$ and $1 \leq r \leq n' - d(x_{i+1}) - 1$, we compute the size of a maximum clique in the proper interval representation. We have three possible maximum cliques at the left, center, and right parts of the proper interval representation. For fixed *i, l,* and *r*, we define three maximum cliques $RC_i(l,r), CC_i(l,r)$, and $L_i^C(l,r)$ in three proper interval graphs induced by $\{x_l, x_{l+1}, \ldots, x_n\} \cup \{y_j, y_{j+1}, \ldots, y'_n\}$ where y_j is the minimum vertex in $N(x_l)$, $\{x_1, x_2, \ldots, x_{l-1}\} \cup \{y_{r+1}, y_{r+2}, \ldots, y'_n\}$, and $\{x_1, x_2, \ldots, x'_i\} \cup \{y_1, y_2, \ldots, y_r\}$ where x'_i is the maximum vertex in $N(y_r)$, respectively. For each pair *(l, r)*, $max\{|R_i^C(l,r)|, |C_i^C(l,r)|, |L_i^C(l,r)|\}$ is computed, and we take the minimum value of $max\{|R_i^C(l,r)|, |C_i^C(l,r)|, |L_i^C(l,r)|\}$ for all pairs, which equals $b(H_i) + 1$ for the fixed i. Then we compute the minimum over i which gives us $b(G) + 1$. □

4.4 Polynomial time Algorithm for Bipartite Permutation Graphs

As we have discussed earlier, Heggernes et al., [5] first gave a polynomial time algorithm for bipartite permutation graphs. They have used a new approach to solve the bandwidth problem. They have made use of a property called strong ordering. But Uehara [8] gave a faster algorithm to solve the bandwidth problem making use of a charecterization based on graph decomposition. In our study, we go through Uehara's work in detail.

The following lemma has been observed by Heggernes et al., [5]:

Lemma 11. *For a bipartite permutation graph G = (V, X, Y) given in line representation with $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y'_n\}$ ordered from left to right in this represetation, we have an optimal layout π such that $\pi(x_i) < \pi(x_{i+1})$ for each $1 \leq i < n$ and $\pi(y'_i) < \pi(y_{i'+1})$ for each $1 \leq i' < n'$.*

So we have to merge these to sequences to get an optimal layout. We will now define a new partition of vertex sets X and Y by $V_0 = \{x_1\}$ and $V_j = \{v | dist(x_1, v) = j\}$. That is V_1 corresponds to all the vertices that are adjacent to x_1 and V_2 corresponds to all the vertices adjacent to the vertices of v_1 and so on. In this construction, we can also see that there is no edge between V_j and V'_j if $|j - j'| > 1$. So we can also write $V_0 \cup V_2 \cup V_4 \cup \ldots = X$ and $V_1 \cup V_3 \cup V_5 \cup \cdots = Y$. Let m denote the index with $V_m \neq \phi$ and $V_{m+1} = \phi$. The partitions can be found in $O(n + n')$ time as it runs once over each vertex. We will denote the induced subgraph $G[V_j \cup V_{j+1}]$ for each $j = 0, 1, \ldots, m-1$ by $G_j = (V_j \cup V_{j+1}, E_j)$. The following lemma follows from the the construction of G_j and the definition of chain graphs:

Lemma 12. *For a bipartite permutation graph $G = (X, Y, E)$ with partitioning $V_0, V_1, V_2, \ldots, V_m$ of $X \cup Y$, every induced subgraph $G_j = (V_j \cup V_{j+1}, E_j)$ is a connected chain graph with $0 \leq j < m$.*

So we have a sequence of chain graphs. For each $j = 0, 1, \ldots, m-1$, we compute an optimal layout of $G_0 \cup G_1 \cup \ldots \cup G_j$ with constraint $b(G_0 \cup \ldots\ldots G_j) \leq k$. We will use Algorithm 3 for each G_j as they are chain graphs. In each G_j we have we have several H_i^j, each of them being a combination of two threshold graphs. We then consider two midpoints l_j and r_j for H_i^j, and obtain three candidates of maximum cliques $LC_i^j(l_j, r_j)$, $CC_i^j(l_j, r_j)$, and $RC_i^j(l_j, r_j)$. The place at which we face difficulty is when we switch to $G_{j+1} = (V_{j+1}, V_{j+2}, E_{j+1})$. We observe that the vertices in V_{j+1} have already been used in the graph $G_j = (V_j, V_{j+1}, E_j)$, and some vertices in V_j are placed among them on the current layout. That is, when we deal with $G_{j+1} = (V_{j+1}, V_{j+2}, E_{j+1})$ some vertices of V_j occur in this through V_{j+1}. So we define a *carry set* S_i^j to tackle the vertices of the carried over vertices of V_j in G_{j+1}. We now observe the following regarding the carry set:

Observation 2. *The carry set S_i^j is equal to $CC_i^j \cap V_j$.*

Using Lemma 12, we compute $V_0, V-1, \ldots, V_m$ in $O(n+n')$ time. We compute a layout up to $b(G_0 \cup \ldots \cup G_j) \leq k$ in that order. We modify Algorithm 3 to deal with the carry set. The modification is not discussed here as it is very tedious. We are ready to prove the following theorem:

Algorithm 4 Algorithm to compute bandwidth of bipartite permutation graph [8]

Input :Bipartite permutation graph $G = (X, Y, E)$ in a line representation with $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y'_n\}$, and positive integer k
Output: Layout with $b(G) \leq k$ if it exists, or otherwise "No"

1. compute the partition $V_0 = \{x_1\}, V_1, V_2, \ldots, V_m$ of $X \cup Y$;
2. let $S_i^{-1} = \phi$ for each i
3. for $j = 0, 1, \ldots, m-1$ **do**
4. construct chain graph $G_j = (V_j, V_{j+1}, E_j)$;
5. **for** $i = 0, 1, \ldots, |V_j|$ **do**
6. construct an interval graph H_i^j;
7. compute a layout of H_i^j with carry S_i^{j-1} satisfying $b(H_i^j) \leq k$ by a modified algorithm of Algorithm 3;
8. **end**
9. **if** there is no layout that achieves $b(H_i^j) \leq k$ **then**
10. **return** ("No")
11. **end**
12. **end**
13. **return** (layout obtained).

Theorem 7. *Let $G = (X, Y, E)$ be a bipartite permutation graph and let k be a positive integer. Then we can find a layout that attains $b(G) \leq k$, if it exists, in $O((|X| + |Y|)^2)$ time.*

Proof. If Algorithm 4 gives an output, then it achieves $b(G) \leq k$. So, we will assume that $b(G) \leq k$ and then show that the algorithm gives an output. By Lemma

11, it is sufficient to consider the orderings that satisfy $\pi(x_1) < \pi(x_2) < \ldots < \pi(x_n)$ and $\pi(y_1) < \pi(y_2) < \ldots < \pi(y'_n)$. Each chain graph G_i is a subgraph of G. Hence we have $b(G_i) \leq b(G)$. When all carry sets are empty, the correctness follows from the proof of Theorem 6. So, from now we assume that there exists a nonempty carry set S_i^j. During the computation, each carry set S_i^j achieves the best possible layout for each pair of i and j; LC_i^j and CC_i^j are saturated for given k in general, and the vertices in $S_i^j = CC_i^j \cap V_j$ (by Observation 2) are put at as left-side as possible. Intuitively, the "margin" in RC_i^j is maximized for each i and j, and we can move no vertex in S_i^j to left any more. Therefore, there exists no better layout than the output of Algorithm 4 under the constraint of k. We note that for each i, the time complexity to perform the modified Algorithm 3 is linear. In Algorithm 4, in steps 6 and 7, when H_i^j is constructed, we deal with the table for V_j which is fixed in H_i^{j-1}. This step consumes only $O(|Vj-1||V_j|)$. So, the total running time can be bounded by $O(\Sigma_{j=0}^{m-1}|Vj-1||V_j|) = O((|X|+|Y|)^2)$. So, we are ready to give the final theorem regarding bandwidth of bipartite permutation graphs whose proof follows from the above discussions.

Theorem 8. *The bandwidth $b(G)$ and an optimal layout of a bipartite permutation graph G = (X, Y, E) can be computed in $O((|X|+|Y|)^2 log b(G))$ time.*

4.5 Biconvex Trees

Definition 12. *The 2-claw is a graph obtained from complete bipartite graph $K_{1,3}$ by replacing each edge by a path of length 2.*

Lemma 13. *2-claw is not a biconvex tree.*

The proof of the above lemma can be verified easily.

Lemma 14. *A tree is biconvex iff it is a caterpillar.*

Proof. Suppose we have a caterpillar T, we will first show that it is a biconvex tree. Suppose v_1, v_2, v_3 and v_4 is the body of the caterpillar and u_1 and u_2 are hairs of v_2 and v_3 respectively. We first put v_1 and v_3 in X and v_2 and v_4 in Y. Now we insert hair of v_i in the set other than where v_i is, and in between v_{i-1} and v_{i+1}. We can see

that it is biconvex tree. Now we have to prove necessity. Say T is a biconvex tree. Let P be a longest path in T . We prove by contradiction. Suppose there exists a vertex not in P having degree greater than 1. This implies that T contains the 2-claw as a subtree (which is not biconvex by the Lemma 13), contradicting the assumption that T is biconvex graph. Hence T is a caterpillar.

□

Assmann et. al., [7] proved the following:

Lemma 15. *The bandwidth of an n-vertex generalized caterpillars of hair length at most two can be computed in O(n log n) time.*

From Lemma 14 and Lemma 15 we can see that bandwidth of a n-vertex biconvex tree can be computed in *O(n log n)* time.

Chapter 5

Bandwidth of Biconvex Graphs

In this chapter, we hypothesize that bandwidth problem cannot be NP-Complete in biconvex bipartite graphs. Though we dont have a complete proof of the hypothesis, we will discuss the irreducibilty of the multiple processor scheduling problem to the bandwidth problem in this particular graph class.
In case of convex trees (Section 3.1), we have seen that removing the degree-1 vertices from the turning point makes it a C a caterpillar and that all catrepillars are biconvex. Therefore, both partitions of C has an ordering satisfying the adjacency property. Now, when we restore the degree-1 vertices, they do not disturb the adjacency property of the other partition (Anish et al., [3]). We can clearly see that it disturbs the adjacency property of partition where it is inserted. We illustrate this with an example shown in Figure 5.1.

In the graph shown in figure, $X = \{1, 8, 3, 10, 5\}$ and $Y = \{2, 9, 11, 12, 4, 6, 7\}$ following the adjacency property. Once we remove the vertices of degree greater than 2 from the turning point (turning point here is $G_T = (\{4, 10, 11, 12\}, E)$) then Y becomes $Y' = \{2, 9, 4, 6, 7\}$. Here $G = (X, Y', E')$ is the caterpillar. So we can see that introducing vertices *11* and *12* will disturb the adjacency property of the original Y in G.
This shows that we cannot reduce the multiple processor scheduling problem to the bandwidth problem in case of biconvex bipartite graphs. Since, most of the graph classes use reduction multiple scheduling problem to bandwidth problem to prove NP-Completeness, we can also safely assume that bandwidth problem in biconvex bipartite cannot be NP-Complete.

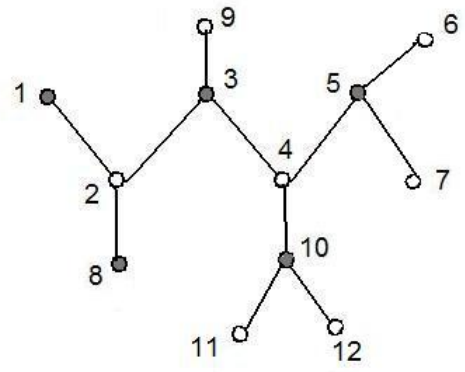


Figure 5.1: Example of biconvex graph

Also, Anish et al., [3] have proved that bandwidth problem can be solved in polynomial time for biconvex trees (Refer Section 4.5). Biconvex trees are very closely related to biconvex graphs. So we expect to get a polynomial time algorithm to solve bandwidth problem in biconvex bipartite graphs.

Bibliography

[1] Christos H. Papadimitriou: The NP-Completeness of the bandwidth minimization problem. Computing 16(3):263-270 (1976)

[2] Monien, B.: The Bandwidth Minimization Problem for Caterpillars with Hair Length 3 is NP-complete. SIAM J. Algebraic Discrete Methods 7(4), 505512 (1986)

[3] Anish Man Singh Shrestha, Satoshi Tayu, Shuichi Ueno: Bandwidth of Convex Bipartite Graphs and Related Graphs. COCOON 2011: 307-318

[4] Brandstadt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. Society for Industrial and Applied Mathematics (1999)

[5] Heggernes, P., Kratsch, D., Meister, D.: Bandwidth of Bipartite Permutation Graphs in Polynomial Time. J. of Discrete Algorithms 7(4), 533544 (2009)

[6] Shrestha, A.M.S., Tayu, S., Ueno, S.: On Orthogonal Ray Graphs. Discrete Appl. Math. 158, 16501659 (2010)

[7] Assmann, S.F., Peck, G.W., Sys lo, M.M., Zak, J.: The Bandwidth of Caterpillars with Hairs of Length 1 and 2. SIAM Journal on Algebraic and Discrete Methods 2(4), 387393 (1981)

[8] Uehara, R.: Bandwidth of Bipartite Permutation Graphs. 19th Annual International Symposium on Algorithms and Computation, Lecture Notes in Computer Science 5369, 824835 (2008)

[9] Sprague, A.P.: An O(n log n) Algorithm for Bandwidth of Interval Graphs. SIAM J. Discrete Math 7(2), 213220 (1994)

[10] Kaplan, H., Shamir, R.: Pathwidth, Bandwidth, and Completion Problems to Proper Interval Graphs with Small Cliques. SIAM J. on Comp. 25(3), 540561 (1996)

[11] D. J. Kleitman and R. V. Vohra. Computing the bandwidth of interval graphs. SIAM J. Disc. Math., 3:373375, 1990

[12] T. Kloks, D. Kratsch, and H. Muller. Bandwidth of chain graphs. Inf. Proc. Lett. 68:313-315, 1998

[13] T. Kloks, D. Kratsch, and H. Muller. Approximating the bandwidth for AT-free graphs. J. Alg.32:4157, 1999

[14] Mahesh, R., Rangan, C.P., Srinivasan, A.: On Finding the Minimum Bandwidth of Interval Graphs. Information and Computation 95, 218224 (1991)

[15] J. Spinrad, A. Brandstadt, and L. Stewart. Bipartite permutation graphs. Disc. Appl. Math., 18:279292, 1987