Efficient Algorithms for the Longest Path Problem

Ryuhei Uehara[1] and Yushi Uno[2]

[1] Department of Information Processing, School of Information Science, JAIST, Ishikawa, Japan
uehara@jaist.ac.jp
[2] Department of Mathematics and Information Science, College of Integrated Arts and Sciences, Osaka Prefecture University, Sakai, Japan
uno@mi.cias.osakafu-u.ac.jp

Abstract. The longest path problem is to find a longest path in a given graph. While the graph classes in which the Hamiltonian path problem can be solved efficiently are widely investigated, very few graph classes are known where the longest path problem can be solved efficiently. For a tree, a simple linear time algorithm for the longest path problem is known. We first generalize the algorithm, and it then solves the longest path problem efficiently for weighted trees, block graphs, ptolemaic graphs, and cacti. We next propose three new graph classes that have natural interval representations, and show that the longest path problem can be solved efficiently on those classes. As a corollary, it is also shown that the problem can be solved efficiently on threshold graphs.

Keywords: Efficient algorithms, graph classes, longest path problem.

1 Introduction

The Hamiltonian path problem (HPP, for short) is one of the most well known $\mathcal{NP}$-hard problems and has numerous applications [15]. For such an intractable problem, there are two major approaches; approximation algorithms [17, 2, 25] and parameterized complexity [13]. In both approaches, we have to change the decision problem to the optimization problem. Therefore the longest path problem (LPP) is one of the basic problems from the viewpoint of combinatorial optimization. From the practical point of view, it is also a very natural approach to find a longest path in a given graph when it does not have a Hamiltonian path. However, finding a longest path seems to be more difficult than determining whether the given graph has a Hamiltonian path or not. Even if a given graph has a Hamiltonian path, it is impossible to efficiently find a path of length $n - n^{\epsilon}$ for any $\epsilon < 1$ unless $\mathcal{P} = \mathcal{NP}$ [18]. In general, LPP does not belong to $\mathcal{APX}$ unless $\mathcal{P} = \mathcal{NP}$ [18], and the best known performance ratio of an approximation algorithm is $O(n(\log\log n/\log n)^2)$ [7] (see also [21, 1, 23, 26] for related results).

Until now, many graph classes have been proposed, and the complexity of hard problems over those graph classes have been extensively investigated [9, 16]. The classification of the graph classes by the difficulty in solving HPP gives us an insight for LPP. If HPP is $\mathcal{NP}$-hard, LPP is also intractable since HPP is a special case of LPP. Such "hard" graph classes include chordal bipartite graphs, strongly chordal split graphs

R. Fleischer and G. Trippen (Eds.): ISAAC 2004, LNCS 3341, pp. 871–883, 2004.

(and hence chordal graphs and split graphs) [22], undirected path graphs, double interval graphs, rectangle graphs [6], and circle graphs [11]. On the other hand, all proper interval graphs have a Hamiltonian path [5]. Thus we can find a longest path of length equal to the number of vertices for any given proper interval graph. Between them, there are "non-trivial" graph classes; HPP is polynomial time solvable for circular arc graphs (and hence interval graphs) [12], and bipartite permutation graphs [24]. In this paper, we focus on LPP for the "non-trivial" graph classes, and propose efficient algorithms for several graph classes.

There are few polynomial time algorithms for finding a longest path in a graph; as far as the authors know, trees are the only natural and non-trivial graph class in which LPP can be solved in polynomial time. The algorithm for trees was invented by W. Dijkstra around 1960. It runs in linear time, and the formal proof is given by R.W. Bulterman et al [10].

We first generalize Dijkstra's longest path algorithm (in this paper, we abbreviate this simply as Dijkstra's algorithm) and its proof in [10], and show that LPP is solved efficiently for (vertex/edge) weighted trees, block graphs, ptolemaic graphs, and cacti.

Next, we focus on graph classes which have natural interval representations. Although all longest paths of a connected interval graph have non-empty intersection [3, Corollary 2.2], there are no efficient algorithms for finding a specific longest path in an interval graph. We introduce three natural and non-trivial graph classes and show that LPP can be solved efficiently on those classes. As a direct corollary, LPP is solved efficiently for threshold graphs.

2 Preliminaries

A graph $G = (V, E)$ consists of a finite set V of *vertices* and a collection E of 2-element subsets of V called *edges*. The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$. For a subset U of V, we denote by $N_G(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the index G. We denote the closed neighborhood $N(v) \cup \{v\}$ by $N[v]$. Given a graph $G = (V, E)$, its *complement* is defined by $\bar{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$, and is denoted by $\bar{G} = (V, \bar{E})$. A vertex set I is an *independent set* if $G[I]$ contains no edges, and the graph $\bar{G}[I]$ is called a *clique*. A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two independent sets. A graph with a partition of its vertex set V into X and Y can be denoted by $G = (X \cup Y, E)$.

For $G = (V, E)$, 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, denoted by $d(u, v)$, is the minimum length of the paths joining u and v. A *cycle* is a path beginning and ending with the same vertex. A cycle of length i is denoted by C_i. An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle. A graph is *chordal* if each cycle of length at least 4 has a chord. Given a graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N(v)]$ is a clique in G.

2.1 Graph Classes with Interval Graph Representation

A graph (V, E) with $V = \{v_0, v_1, \ldots, v_{n-1}\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_{v_0}, I_{v_1}, \ldots, I_{v_{n-1}}\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $0 \leq i, j < n$. We call the set $\mathcal{I}$ of intervals an *interval representation* of the graph. For each interval I, we denote by $L(I)$ and $R(I)$ the left and right endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$).

A bipartite graph $(X \cup Y, E)$ with $X = \{x_0, x_1, \ldots, x_{n_1-1}\}$ and $Y = \{y_0, y_1, \ldots, y_{n_2-1}\}$ is an *interval bigraph* if there are families of intervals $\mathcal{I}_X = \{I_{x_0}, I_{x_1}, \ldots, I_{x_{n_1-1}}\}$ and $\mathcal{I}_Y = \{I_{y_0}, I_{y_1}, \ldots, I_{y_{n_2-1}}\}$ such that $\{x_i, y_j\} \in E$ iff $I_{x_i} \cap I_{y_j} \neq \emptyset$ for each i and j with $0 \leq i < n_1$ and $0 \leq j < n_2$. We also call the families of intervals $(\mathcal{I}_X, \mathcal{I}_Y)$ an *interval representation* of the graph. If no confusion can arise we sometimes unify a vertex v and the corresponding interval I_v, and write $N(I_v)$, $L(v)$. Moreover, if a vertex v corresponds to a point, the vertex v and the corresponding point $L(I_v) = R(I_v)$ are sometimes unified.

For two intervals I and J, we write $I \prec J$ if $L(I) \leq L(J)$ and $R(I) \leq R(J)$. For any interval representation $\mathcal{I}$ and a point p, $N[p]$ denotes the set of intervals that contain the point p.

Given an interval (bi)graph, an interval representation is said to be *compact* if the following conditions hold;

1. for each interval I, $R(I)$ and $L(I)$ are integers, and
2. for each pair of integers p, q with $N[p] \neq \emptyset$ and $N[q] \neq \emptyset$, $N[p] \setminus N[q] \neq \emptyset$ and $N[q] \setminus N[p] \neq \emptyset$.

Lemma 1. *For any given interval graph $G = (V, E)$, there is a linear time algorithm that constructs a compact interval representation $\mathcal{I}$ such that*

1. *each simplicial vertex v is a point; that is, $L(I_v) = R(I_v)$,*
2. *$\bigcup_{I \in \mathcal{I}} I$ is contained in $[0, |V| - 1]$, and*
3. *each integer point i with $N[i] \neq \emptyset$ corresponds to a distinct maximal clique of G.*

Proof. Given an interval graph $G = (V, E)$, we can construct an $\mathcal{MPQ}$-tree, which is a kind of labeled $\mathcal{PQ}$-tree, in linear time [19]. From the $\mathcal{MPQ}$-tree, in a natural way, we can construct a compact interval representation in linear time. This fact can be proved by the induction of the number of nodes and sections in the $\mathcal{MPQ}$-tree, but it is straightforward and tedious, and is omitted here. We show that the constructed compact interval representation satisfies the conditions.

Let v be any simplicial vertex in G. By the definition, $N(v)$ induces a clique. Thus, by the Helly property (see, e.g.,[4]), all vertices u in $N(v)$ share a common point. Therefore, I_v contains this point. If I_v is not an integer point, I_v must contain an interval $[i, i+1]$ for some integer i. Then, $N[i] = N[i+1]$, which contradicts the compactness of the interval representation. This yields claim 1.

The maximal cliques of an interval graph G can be linearly ordered so that for each vertex v, the maximal cliques containing v occur consecutively [8]. Since the representation is compact, we have claim 3 immediately. It is well known that chordal graphs, and hence interval graphs, have at most n maximal cliques. This fact with claim 3 implies claim 2. □

An interval graph is a *proper interval graph* if no two intervals I and J properly contain each other, i.e., if either $I \prec J$ or $J \prec I$ for each I and J.

Let $G = (X \cup Y, E)$ be a bipartite graph. An ordering $<$ of X in G has the *adjacency property* if for each vertex $y \in Y$, $N(y)$ consists of vertices that are consecutive in the ordering $<$ of X. A bipartite graph $G = (X \cup Y, E)$ is *biconvex* if there are orderings of X and Y that fulfill the adjacency property. G is *convex* if there is an ordering of X or Y that fulfills the adjacency property.

Given a biconvex graph $G = (X \cup Y, E)$ with $|X| = n_1$ and $|Y| = n_2$, a compact interval representation $\mathcal{I}(G)$ is constructed as follows: Let $x_0 < x_1 < \cdots < x_{n_1-1}$ and $y_0 < y_1 < \cdots < y_{n_2-1}$ be the orderings of X and Y that have adjacency property. Let x_i correspond to the integer point i with $0 \leq i < n_1$. For each j with $0 \leq j < n_2$, since each y_j contains consecutive xs, we can make y_j correspond to the interval $[L(y_j), R(y_j)]$ such that $L(y_j) = l$ and $R(y_j) = r$, where x_l and x_r are the minimum and maximum vertices in $N(y_j)$, respectively.

Lemma 2. *For the compact interval representation $\mathcal{I}(G)$ of a given biconvex graph $G = (X \cup Y, E)$, there are two indices j_t and j_b such that (1) $L(y_0) \geq L(y_1) \geq \cdots \geq L(y_{j_t})$ and $L(y_{j_t}) \leq L(y_{j_t+1}) \leq \cdots \leq L(y_{n_2-1})$, and (2) $R(y_0) \leq R(y_1) \leq \cdots \leq R(y_{j_b})$ and $R(y_{j_b}) \geq R(y_{j_t+1}) \geq \cdots \geq R(y_{n_2-1})$.*

Proof. We note that $x_0 \in N(y_{j_t})$, $x_{n_1-1} \in N(y_{j_b})$, and we can assume that $j_t \leq j_b$ without loss of generality. If the index j_t does not exist, we have three consecutive vertices, say, $y_1 < y_2 < y_3$ such that $L(y_1) > L(y_2)$ and $L(y_3) > L(y_2)$. Then there is a vertex $x \in X$ such that $y_1, y_3 \in N(x)$ and $y_2 \notin N(x)$. This contradicts the definition of biconvex graphs. Thus j_t exists. The case of index j_b is symmetric. □

A graph $G = (V, E)$ is called *threshold* if there exist nonnegative integers $w(v)$ $(v \in V)$ and an integer t such that $\sum_{v \in S} w(v) \leq t$ if and only if S is an independent set of G.

2.2 Tree-Like Graph Classes

We here introduce some graph classes that have similar structure to trees. A graph G is a *block graph* if G is connected and every maximal 2-connected subgraph is a clique. Block graphs can be seen as graphs obtained from trees by replacing each edge by a clique, and the cliques have at most one vertex in common. A graph G is *ptolemaic* if for any vertices u, v, w, x of G, $d(u,v)d(w,x) \leq d(u,w)d(v,x) + d(u,x)d(v,w)$. Ptolemaic graphs can be seen as graphs obtained from trees by replacing each edge by a clique, and the cliques have any number of vertices in common. Thus, a block graph is ptolemaic. A *cactus* is a graph whose block is either an edge or a cycle. Similarly, cacti can be seen as graphs obtained from trees by replacing a part of edges by a cycle, and the cycles have at most one vertex in common. See [9] for further details of these graph classes.

2.3 New Graph Classes

We say that an interval graph $G = (V, E)$ is said to be an *interval biconvex graph* if V can be partitioned into two sets S and Y such that

1. each vertex $x \in S$ is simplicial in G, and
2. the bipartite graph $G' := (S \cup Y, E')$ is a biconvex graph, where $E' := \{\{x, y\} \mid x \in S, y \in Y, \text{ and } \{x, y\} \in E\}$.

Intuitively, interval biconvex graphs G are interval graphs obtained from biconvex graphs G'; they have common interval representations. We have the following proper inclusion:

Lemma 3. *(Proper interval graphs ∪ threshold graphs) ⊂ interval biconvex graphs ⊂ interval graphs.*

Proof. The inclusions "proper interval graphs ⊂ interval biconvex graphs ⊂ interval graphs" are easy to follow directly from the corresponding definitions. It remains to show that any threshold graph $G = (V, E)$ is an interval biconvex graph. We assume that vertices are ordered with respect to their weights; $w(v_0) \leq w(v_1) \leq \cdots \leq w(v_{n-1})$. Let i be the smallest index with $w(v_i) \geq \frac{t}{2}$. We partition V into $V' := \{v_i, \ldots, v_{n-1}\}$ and $S := V \setminus V'$. Then we have (1) $G[V']$ is a clique, (2) for each vertex v in S, there is an index j $(> i)$ such that $N(v) = \{v_j, \ldots, v_{n-1}\}$, and (3) each vertex in S is simplicial. Thus G is an interval biconvex graph. □

Lemma 4. *Let $G = (S \cup Y, E)$ be an interval biconvex graph. Let v be a (simplicial) vertex in S. Let $N_S[v]$ be the set $\{v\} \cup (N(v) \cap S)$ in G. Then $N_S[v]$ induces a clique, and for any two vertices v_1 and v_2 in $N_S[v]$, $N_G[v_1] = N_G[v_2]$.*

Proof. Since v is simplicial, $N_S[v]$ induces a clique. Thus we show that $N[v_1] = N[v_2]$ for any v_1 and v_2 in $N_S[v]$. We first note that $v_1 \in N[v_2]$ since v is simplicial. Let w be any vertex in $N[v_1]$. Then, since v_1 is also simplicial, $\{w, v_2\} \in E$, consequently, $w \in N[v_2]$. Hence we have $N[v_1] \subseteq N[v_2]$. Symmetric argument implies $N[v_2] \subseteq N[v_1]$, which completes the proof. □

Corollary 1. *Let $G = (S \cup Y, E)$ be an interval biconvex graph, and $\mathcal{I}(G)$ be a compact interval representation of G. Then*

(1) for each vertex v in S, I_v is an integer point, and
(2) for each vertices u and v in S with $N[u] = N[v]$, I_v and I_u are the same integer point.

We say that a biconvex graph $G = (X \cup Y, E)$ with $Y = \{y_0, \ldots, y_{n_2-1}\}$ is *proper* if $y_0 \prec y_1 \prec \cdots \prec y_{n_2-1}$ and is *linearly included* if $I_{y_0} \subseteq I_{y_1} \subseteq \cdots \subseteq I_{y_{n_2-1}}$. Intuitively, a biconvex graph G is proper if $j_0 = j_t$ and $j_b = n_2 - 1$, and G is linearly included if $j_t = n_2 - 1$ in Lemma 2. The motivation for introducing these two graph classes comes from the fact that every biconvex graph can be partitioned into three graphs from these classes. This can be achieved as follows: For any biconvex graph $G = (X \cup Y, E)$ with the two indices j_t and j_b from Lemma 2, we partition Y into $Y_1 := \{y_0, \ldots, y_{j_t}\}$, $Y_2 := \{y_{j_t+1}, \ldots, y_{j_b-1}\}$, and $Y_3 := \{y_{j_b}, \ldots, y_{n_2-1}\}$. Let X_i be the set of vertices in $N(Y_i)$ and let E_i be the set of edges induced by $X_i \cup Y_i$. Then $G_1 := (X_1 \cup Y_1, E_1)$ and $G_3 := (X_3 \cup Y_3, E_3)$ are linearly included biconvex graphs, and $G_2 := (X_2 \cup Y_2, E_2)$ is a proper biconvex graph.

3 Algorithms for Tree-Like Graphs

Given a finite tree T, a longest path can be found in linear time. A simple procedure is invented by E.W. Dijkstra around 1960. The formal correctness proof of the procedure is given in [10]. In order to illustrate our algorithms for tree-like graphs, we first give a framework that allows a generalization of Dijksta's algorithm by transforming G into another graph G'. For a given graph $G = (V, E)$, suppose that we can construct a new graph $G' = (V', E')$ satisfying the following three conditions:

1. $V \subseteq V'$,
2. for each pair u, v in V, $d_{G'}(u, v)$ equals the length of a longest path joining u and v in G, and
3. for each pair u, v in V, $d_{G'}(u, v)$ is given by the unique shortest path on G'.

Then the following algorithm computes the length of a longest path in G by using the graph G' in $O(|V| + |E| + |V'| + |E'|)$ time and space.

Algorithm TR
Input: Graphs $G = (V, E)$ and $G' = (V', E')$.
Output: The length of a longest path P in G.
T1. pick any vertex u in V of G';
T2. find a vertex x in V such that $d_{G'}(u, x)$ is largest;
T3. find a vertex y in V such that $d_{G'}(x, y)$ is largest;
T4. output the length of a shortest path joining x and y on G'.

For $G' = G$, TR is in fact Dijkstra's algorithm.

Theorem 1. *Algorithm* TR *computes the length of a longest path in G in linear time.*

Proof. In the proof in [10], the correctness of Dijkstra's algorithm is based on the following three facts; (1) for any vertices u and v, the shortest path joining them gives the longest path joining them, (2) for any vertices u, v and w, we have $d(u, v) \leq d(u, w) + d(w, v)$, and (3) for any vertices u, v and w, we have $d(u, v) = d(u, w) + d(w, v)$ if and only if w is on the path joining u and v.

By condition of G', for any vertices u and v in V, the length of the shortest path joining u and v on G' is equal to the length of the longest path joining u and v on G. For any vertices u, v, and w in V, we also have $d_{G'}(u, v) \leq d_{G'}(u, w) + d_{G'}(w, v)$. By condition of G', $d_{G'}(u, v)$ is given by the unique shortest path on G'. Thus, for any vertices u, v and w in V, we have $d_{G'}(u, v) = d_{G'}(u, w) + d_{G'}(w, v)$ if and only if w is on the shortest path joining u and v. Therefore, we can use the same argument in the proof of [10] to show the correctness of Algorithm TR. Since shortest paths in G' are unique, we can use the standard BFS to implement TR in linear time. □

In the following, we investigate graph classes for which Algorithm TR computes the length of a longest path with given suitable reductions. We note that, in step T4, Algorithm TR outputs the length of a longest path. However, it is easy to modify the specialized algorithms so that they output not just the length but the longest path itself.

Lemma 5. *Let $G = (V, E)$ be a tree, and $w : V \cup E \to \boldsymbol{Z}_+$ be a weight function of vertices and edges. We define the length of a weighted path $(v_1, v_2, \ldots, v_k)$ by $\sum_{i=1}^{k} w(v_i) + \sum_{i=1}^{k-1} w(\{v_i, v_{i+1}\})$. Then the weighted longest path problem can be solved in $O(|V|)$ time and space.*

Proof. We can use Dijkstra's algorithm straightforwardly. □

Theorem 2. *Let $G = (V, E)$ be a block graph. Then the longest path problem can be solved in $O(|V| + |E|)$ time and space.*

Proof. Given block graph G, we construct a weighted tree G' as follows: For each maximal clique K of size $k > 2$, we first remove all edges in K, second create a vertex c of weight $k - 3$, and third join c and each vertex in K by an edge of weight 1. We repeat the replacement for all maximal cliques of size > 2. The resulting graph G' is an integer weighted tree. Let u and v be any two vertices in V. Then the length of a longest path between u and v on G is equal to the length of the weighted path joining u and v on G'. Thus we can use Lemma 5. □

Theorem 3. *Let $G = (V, E)$ be a ptolemaic graph. Then the longest path problem can be solved in $O(|V|\ (|V| + |E|))$ time and space.*

Proof. Let K and K' be any two maximal cliques in G with $|K \cap K'| \geq 2$. Then, for any pair of vertices u and v in $K \cup K'$, we can construct a path P of length $|K \cup K'| - 1$ such that its endpoints are u and v and P contains all vertices in $K \cup K'$. We now add edges to $G[K \cup K']$ until $G[K \cup K']$ becomes a clique. It is easy to see that the replacement does not change the length of the longest path of the graph. While G contains two maximal cliques K and K' with $|K \cap K'| \geq 2$, we repeat the above process. Let G' be the resulting graph. Then G' is a block graph and the length of a longest path in G' is equal to the length of a longest path in G. Thus, Theorem 2 completes the proof. □

Theorem 4. *Let $G = (V, E)$ be a cactus. Then the longest path problem can be solved in $O(|V|^2)$ time and space.*

Proof. We first show a reduction in $O(|V|^3)$ time and space. Let C_k be any cycle of k vertices in G. We replace C_k by a gadget C'_k defined as follows (Fig. 1): For each pair of vertices v_i and v_j on C_k, if $d_{C_k}(v_i, v_j) = h$, C'_k contains a path of length $k - h$ (with auxiliary vertices). We note that the length of the longest path joining v_i and v_j on G (or C_k) is $k - h$. We replace all cycles of G by the gadgets. The resulting graph G' has $O(|V|^3)$ vertices and edges, and the reduction can be done in $O(|V|^3)$ time and space. To show the correctness of Algorithm TR, we show that for each pair u, v in V, (1) $d_{G'}(u, v)$ equals the length of a longest path joining u and v in G, and (2) the shortest path joining u and v on G' is uniquely determined. Since G is a cactus, we show that for each pair v_i and v_j on C_k, $d_{G'}(v_i, v_j)$ equals the length of a longest path joining v_i and v_j in G. Without

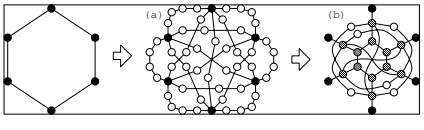

Fig. 1. Gadget for C_5

loss of generality, we assume that $i = 0$ and $0 < j \leq \frac{k}{2}$. By the construction, we have $d_{G'}(v_0, v_j)$ is at most $k - j$ by the added path joining v_0 and v_j. We assume that there is a shorter path P in G' joining v_0 and v_j to derive a contradiction. Since G is a cactus, P contains at least one vertex v' on C_k with $v' \neq v_0, v_j$. From the construction, for any pair of vertices u and u' on C'_k, we have $\frac{k}{2} \leq d_{G'}(u, u') \leq k - 1$. Thus, $k - j \leq k - 1$. On the other hand, since P contains three vertices v_0, v_j, and v' on C_k, the length of P is at least $2\frac{k}{2} = k$, which contradicts that the length of P is shorter than $k - j \leq k - 1$. Thus, for each pair u, v in V, $d_{G'}(u, v)$ is equal to the length of a longest path joining u and v in G, and the shortest path joining u and v on G' is uniquely determined. In each gadget in Fig. 1, we replace each path by the edge of weight equal to the length of the path. This reduction can be done in $O(|V|^2)$ time and space. □

4 Algorithms for Biconvex Graphs

In this section, we consider the longest path problem on two subclasses of biconvex graphs. We assume that a biconvex graph $G = (X \cup Y, E)$ is given with its compact interval representation. For given any path P, we denote by P_X the set of vertices in P and X, and by P_Y the set of vertices in P and Y. We assume that the sets P_X and P_Y are ordered; when $\mathsf{P}_X = \{x_{i_0}, x_{i_1}, \ldots\}$ and $\mathsf{P}_Y = \{y_{j_0}, y_{j_1}, \ldots\}$, then $\mathsf{P} = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \ldots)$ or $\mathsf{P} = (y_{j_0}, x_{i_0}, y_{j_1}, x_{i_1}, \ldots)$.

4.1 An Algorithm for Proper Biconvex Graphs

Lemma 6. *If the graph $G = (X \cup Y, E)$ is proper biconvex, there is a longest path P with $\mathsf{P}_X = \{x_{i_0}, x_{i_1}, \cdots\}$ and $\mathsf{P}_Y = \{y_{j_0}, y_{j_1}, \cdots\}$ s.t. $i_k < i_{k+1}$ and $j_k < j_{k+1}$ for each k.*

Proof. Let P' be any longest path $(x_{i'_0}, y_{j'_0}, x_{i'_1}, y_{j'_1}, \ldots)$ such that $x_i \in X$ and $y_j \in Y$ (the symmetric case that P' starts from a vertex in Y is omitted). We construct P from P' such that P contains the same vertices in P' and satisfies the condition. The proof is done by the induction for the length of the path. The lemma holds when the length of the path is 2. Thus we assume that the length of the path is at least 3. We have two cases:
(1) The path ends two vertices $(x_{i'_k}, y_{j'_k})$ with $x_{i'_k} \in X$ and $y_{j'_k} \in Y$. Then, by the inductive hypothesis, there is a path $\mathsf{P}'' = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \ldots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfying the condition. If either $y_{j_{k-1}} \prec y_{j'_k}$ or $y_{j'_k} \prec y_{j_0}$, the path $(x_{i_0}, y_{j_0}, \ldots, x_{i_{k-1}}, y_{j'_k}, x_{i'_k})$ or $(x_{i'_k}, y_{j'_k}, x_{i_0}, y_{j_0}, \ldots, x_{i_{k-1}})$ satisfies the condition. Thus we suppose that $y_{j_0} \prec y_{j'_k} \prec y_{j_{k-1}}$. Then since the graph is proper, there is an index h such that $y_{j_h} \prec y_{j'_k} \prec y_{j_{h+1}}$. Then $x_{i_{h+1}} \in I_{y_{j'_k}}$, and $x_{i'_k} \in (I_{y_{j_h}} \cup I_{y_{j_{h+1}}})$. If $x_{i'_k} \in I_{y_{j_h}}$, the path $(x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \ldots, y_{j_h}, x_{i'_k}, y_{j'_k}, x_{i_{h+1}}, y_{j_{h+1}}, \ldots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfies the condition. Otherwise, the path $(x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \ldots, y_{j_h}, x_{i_{h+1}}, y_{j'_k}, x_{i'_k}, y_{j_{h+1}}, \ldots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfies the condition.
(2) The path ends two vertices $(y_{j'_{k-1}}, x_{i'_k})$ with $y_{j'_{k-1}} \in Y$ and $x_{i'_k} \in X$. Then, by the inductive hypothesis, there is a path $\mathsf{P}'' = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \ldots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfying the condition. Then we have three possible cases; $y_{j_{k-1}} \prec y_{j'_{k-1}}$, $y_{j'_{k-1}} \prec y_{j_0}$, and $y_{j_0} \prec y_{j'_{k-1}} \prec y_{j_{k-1}}$. Using the same argument as in (1), we have the lemma. □

Theorem 5. *In a proper biconvex graph $G = (X \cup Y, E)$, a longest path can be found in $O(|X \cup Y| + |E|)$ time.*

Proof. (Outline.) To compute the longest path satisfying the condition in Lemma 6, we use a standard dynamic programming. We define two functions $f(x_i, y_j)$ and $g(y_j, x_i)$ such that $f(x_i, y_j)$ gives the length of a longest path starting at the edge $\{x_i, y_j\}$ and $g(y_j, x_i)$ gives the length of a longest path starting at the edge $\{y_j, x_i\}$. We define $f(x_i, y_j) = 0$ if $\{x_i, y_j\} \notin E$ or y_j does not exist, and $g(y_j, x_i) = 0$ if $\{x_i, y_j\} \notin E$ or x_i does not exist. Then, if $\{x_i, y_j\} \in E$, we have $f(x_i, y_j) = \max\{f(x_i, y_{j+1}), g(y_j, x_{i+1}) + 1\}$ and $g(y_j, x_i) = \max\{f(x_i, y_{j+1}) + 1, g(y_j, x_{i+1})\}$. Therefore, the length of a longest path is $\max\{f(x_0, y_0), g(y_0, x_0)\}$, which can be computed by a standard dynamic programming in linear time and space. Computing the longest path itself is also straightforward. □

4.2 An Algorithm for Linearly Included Biconvex Graphs

Lemma 7. *A linearly included biconvex graph $G = (X \cup Y, E)$ is proper.*

Proof. By definition, there is an index i_c such that $N(x_0) \subseteq N(x_1) \subseteq \cdots \subseteq N(x_{i_c})$ and $N(x_{n_1-1}) \subseteq N(x_{n_1-2}) \subseteq \cdots \subseteq N(x_{i_c})$, where $n_1 = |X|$. Since Y is linearly ordered in inclusion, for any pair of x_i and $x_{i'}$ with $0 \leq i \leq c$ and $c \leq i' \leq n_1 - 1$, we have $N(x_i) \subseteq N(x_{i'})$ or $N(x_{i'}) \subseteq N(x_i)$. Thus, we can linearly order both of X and Y; the ordering of Y is as it is, and the ordering of X is computed by the following algorithm in linear time:

Input: A linearly included biconvex graph $G = (X \cup Y, E)$ with $|X| = n_1$ and $|Y| = n_2$, the orderings over X and Y, and a compact interval representation $\mathcal{I}(G)$.
Output: A linear ordering of X in inclusion.
P0. $\ell := 0$; $r := n_1 - 1$;
P1. if $N(x_\ell) \subseteq N(x_r)$ then output x_ℓ and $\ell := \ell + 1$ else output x_r and $r := r - 1$;
P2. if $\ell < r$ then goto P1 else output $x_\ell (= x_r)$ and halt.

The correctness of the algorithm is easy. □

Theorem 6. *In a linearly included biconvex graph $G = (X \cup Y, E)$, a longest path can be found in $O(|X \cup Y| + |E|)$ time.*

Proof. By Lemma 7, G is proper. Since Y is linearly ordered in inclusion, the condition $N(x_\ell) \subseteq N(x_r)$ is equivalent to the condition $\min\{N(x_\ell)\} \geq \min\{N(x_r)\}$. Thus the step P1 in the algorithm in the proof of Lemma 7 can be done in $O(1)$ time, and the algorithm runs in linear time. Therefore, combining the algorithms in Lemma 7 and Theorem 5, we can construct a linear time algorithm that finds a longest path of G. □

5 An Algorithm for Interval Biconvex Graphs

Let $G = (S \cup Y, E)$ be an interval biconvex graph with the set S of simplicial vertices. Let $x_0 < x_1 < \cdots < x_{n_1-1}$ and $y_0 < y_1 < \cdots < y_{n_2-1}$ be the orderings over S and Y that have adjacency property. We here denote by $I(x_i)$ the integer point I_{x_i}, that is,

$I(x_i) = R(I_{x_i}) = L(I_{x_i})$. By Lemma 4, for each x and x' with $N[x] = N[x']$, we have $I(x) = I(x')$. For each j with $0 \leq j \leq n_2 - 1$, since each y_j contains consecutive xs, we can let y_j correspond to the interval $[L(y_j), R(y_j)]$ such that $L(y_j) = I(x_\ell)$ and $R(y_j) = I(x_r)$, where x_ℓ and x_r are the minimum and maximum vertices in $N(y_j)$, respectively. By definition and the proof of Lemma 2, we immediately have the following lemma:

Lemma 8. *For the compact interval representation $\mathcal{I}(G)$ of the interval biconvex graph $G = (S \cup Y, E)$ with the set S of simplicial vertices, there are two indices j_t and j_b such that (0) $j_t \leq j_b$, (1) $L(y_0) \geq L(y_1) \geq \cdots \geq L(y_{j_t})$ and $L(y_{j_t}) \leq L(y_{j_t+1}) \leq \cdots \leq L(y_{n_2-1})$, and (2) $R(y_0) \leq R(y_1) \leq \cdots \leq R(y_{j_b})$ and $R(y_{j_b}) \geq R(y_{j_t+1}) \geq \cdots \geq R(y_{n_2-1})$.*

The following proposition is also immediate (see also [5, Lemma 2]):

Proposition 1. *Let $G = (S \cup Y, E)$ be a connected interval biconvex graph. Let $y_0 < y_1 < \cdots < y_{n_2-1}$ be the orderings over Y that have adjacency property. Then $\{y_{i-1}, y_i\} \in E$ for each $1 \leq i \leq n_2 - 1$. That is, $(y_0, y_1, \ldots, y_{n_2-1})$ is a path of G.*

Hereafter, we denote by (S, y) a path $(x_1, \ldots, x_k, y)$ for a set $S = \{x_1, \ldots, x_k\}$ if $G[S]$ is a clique.

Lemma 9. *Given a connected interval biconvex graph $G = (S \cup Y, E)$, there is a longest path P such that (1) the vertices in $Y \cap \mathsf{P}$ appear consecutively; that is, $Y \cap \mathsf{P}$ is $y_j, y_{j+1}, \ldots, y_{j+k-1}, y_{j+k}$ for some j and k, and those vertices appear according to the ordering over Y, (2) the consecutive vertices in S of P correspond to the same integer point; that is, if P contains a subpath $(y_j, x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+h}, y_{j+1})$, we have $I(x_i) = I(x_{i+1}) = \cdots = I(x_{i+h})$, (3) the vertices in S of P appear according to the ordering over S; that is, if $S \cap \mathsf{P}$ is $x_{i_1}, x_{i_2}, \ldots, x_{i_h}$ in this order, $x_{i_1} < x_{i_2} < \cdots < x_{i_h}$, and (4) P starts and ends at the vertices in S.*

Let s and t be integers such that P starts with the vertices in $N[s] \cap S$ and ends with the vertices in $N[t] \cap S$. Let y_s and y_t be the vertices such that P starts with $(N[s], y_s)$ and ends with $(y_t, N[t])$. Then (5) y_s is the minimum vertex in Y with I_{y_s} contains s, and y_t is the maximum vertex in Y with I_{y_t} contains t.

Proof. (1) We assume that P contains y_{j_1}, y_{j_2}, and y_{j_3} in this ordering and $y_{j_1} < y_{j_3} < y_{j_2}$. Then we have five cases; (a) $I(y_{j_1}) \subseteq I(y_{j_3}) \subseteq I(y_{j_2})$, (b) $I(y_{j_1}) \subseteq I(y_{j_3})$ and $y_{j_3} \prec y_{j_2}$, (c) $y_{j_1} \prec y_{j_3} \prec y_{j_2}$, (d) $y_{j_1} \prec y_{j_3}$ and $I(y_{j_2}) \subseteq I(y_{j_3})$, or (e) $I(y_{j_2}) \subseteq I(y_{j_3}) \subseteq I(y_{j_1})$. We assume that they are minimal, that is, they are consecutive in $\mathsf{P} \cap Y$. Then, in any case, swapping y_{j_2} and y_{j_3} we have also a path. Repeating this process, we have a path such that the vertices in Y appear according to the ordering over Y. Let y_j and y_{j+k} be the first and last vertices in $\mathsf{P} \cap Y$, respectively. We next show that all vertices $y_{j+k'}$ with $0 < k' < k$ appear on P. We assume that some $y_{j+k'}$ with $0 < k' < k$ does not appear on P. Then, inserting it, we have a longer path, which contradicts that P is a longest path.

(2) Let s be any integer such that there is a vertex x in $N[s] \cap S \cap \mathsf{P}$. If there is a vertex x' such that $x' \in N[s] \cap S$ and x' does not appear in P, we have longer path by replacing a subpath (x) of P by (x, x'). Thus all vertices in $N[s] \cap S$ appear in P. It is

clear that gathering all vertices in $N[s] \cap S \cap \mathsf{P}$ has no effects on the connectivity and length of P. Thus we can assume that all vertices in $N[s] \cap S$ are consecutive in P.

(3) By (1), all vertices in $Y \cap \mathsf{P}$ appear consecutively. Thus, using the same argument in (1), we can assume that all vertices in $S \cap \mathsf{P}$ also appear consecutively.

(4) To derive a contradiction, we assume that P starts at the vertex y_j in S. Then, by the definition of a compact interval representation, we have (a) $L(y_j) = R(y_{j'})$ for some $j' < j$, or (b) $N[L(y_j)] \cap S \neq \emptyset$. In the case (a), we have longer path by adding $y_{j'}$ at the top of P. On the other hand, in the case (b), we also have longer path by adding the vertices in $N[L(y_j)] \cap S$ at the top of P. Thus, P starts at the vertex in S. Using the same argument, we can show that P ends at the vertex in S.

(5) To derive a contradiction, we assume that P starts with $(N[s], y_s)$ and there is a vertex $y_{s'}$ such that $s' \neq s$, $I_{y_{s'}}$ contains s, and $y_{s'} \prec y_s$. By (1), $y_{s'}$ does not appear in P. However, in the case, we have longer path by replacing $(N[s], y_s)$ by $(N[s], y_{s'}, y_s)$, which is a contradiction. Thus y_s is the minimum vertex in Y with I_{y_s} contains s. Using the symmetric argument, y_t is the maximum vertex in Y with I_{y_t} contains t. □

By Lemma 9, the outline of our algorithm is as follows:

0. for each integer s and t with $0 \leq s < t < n_1$, suppose $N[s] \cap S$ and $N[t] \cap S$ are the endpoints of a longest path;
1. let y_{j_s} be the smallest vertex with $L[y_{j_s}] \leq s \leq R[y_{j_s}]$, and let y_{j_t} be the largest vertex with $L[y_{j_t}] \leq t \leq R[y_{j_t}]$;
2. for each integer $i = s+1, s+2, \ldots, t-1$, determine where the vertices in $S \cap N[i]$ are inserted in the path $(N[s], y_{j_s}, y_{j_s+1}, \ldots, y_{j_t-1}, y_{j_t}, N[t])$.

Step 2 can be solved by finding a maximum weighted matching in the weighted bipartite graph $G' = (X' \cup Y', E')$ defined as follows; $X' = \{i \mid I(x) = i$ for some $x \in S$ with $s < i < t\}$, $Y' = \{\{y_j, y_{j+1}\} \mid y_j, y_{j+1} \in Y$ and $j_s \leq j \leq j_t - 1\}$, and E' consists of edges $e = \{i, \{y_j, y_{j+1}\}\}$ if $N[i]$ contains y_j and y_{j+1}. The weight of the edge $e = \{i, \{y_j, y_{j+1}\}\}$ is defined by $|S \cap N[i]|$. A weighted matching M of G' gives us a path of G as follows; if an edge $e = \{i, \{y_j, y_{j+1}\}\}$ is in M, the path of G contains $(y_j, N[i], y_{j+1})$. By Lemma 9, the maximum weighted matching of G' gives us a longest path of G. The detail of the algorithm can be described as follows:

Algorithm IBG:

Input: An interval biconvex graph $G = (S \cup Y, E)$ with $|S| = n_1$ and $|Y| = n_2$ and a compact interval representation $\mathcal{I}(G)$.

Output: A longest path P.

B0. construct the weighted bipartite graph $G' = (X', Y', E')$ for G;

B1. for each integer s and t with $0 \leq s < t < n_1$ do the following;

- B1.1. let $N[s]$ and $N[t]$ suppose the endpoints of a longest path;
- B1.2. let y_{j_s} be the smallest vertex with $L[y_{j_s}] \leq N[s] \leq R[y_{j_s}]$, and let y_{j_t} be the largest vertex with $L[y_{j_t}] \leq N[t] \leq R[y_{j_t}]$;
- B1.3. let G'' be the induced bipartite graph from G' by the vertices in S between s and t and the vertices in Y between y_{j_s} and y_{j_t};
- B1.4. find a maximum weighted matching M in G'';
- B1.5. compute a path from $(N[s], y_{j_s}, y_{j_s+1}, \ldots, y_{j_t}, N[t])$ and the weighted edges in M;

B2. find the longest path among the paths generated in step B1.5.

Theorem 7. *A longest path in a given interval biconvex graph $G = (S \cup Y, E)$ with $|S \cup Y| = n$ and $|E| = m$ can be found in $O(n^3(m + n \log n))$ time.*

Proof. The correctness of Algorithm IBG follows from Lemma 9. Thus we analyze the running time. For the weighted bipartite graph $G' = (X' \cup Y', E')$ constructed in step B0, we have $|X'| = O(n_1)$, $|Y'| = n_2 - 1$, and $|E'| = O(|E|)$. A maximum weighted matching can be found in $O(|V|(|E| + |V| \log |V|))$ time and $O(|E|)$ space for (any) given graph $G = (V, E)$ [14]. Hence step B1.4 takes $O(n(m + n \log n))$ time with $n = n_1 + n_2$ and $m = |E|$. Therefore total running time is $O(n_1^2 n(m + n \log n)) = O(n^3(m + n \log n))$. □

Corollary 2. *A longest path in a given threshold graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ can be found in $O(n + m)$ time and space.*

Proof. (Sketch.) By Lemma 3 and Theorem 7, a longest path in a threshold graph can be found in $O(n^3(m + n \log n))$ time. However, Algorithm IBG can be simplified to run in linear time and space using the properties shown in the proof of Lemma 3 and the fact that $G[S]$ is an independent set. A similar idea can be found in [20], and hence is omitted here. □

6 Concluding Remarks

The major open problem is the complexity of the longest path problem for interval graphs, convex graphs, and biconvex graphs.

References

1. N. Alon, R. Yuster, and U. Zwick. Color-Coding. *J. ACM*, 42(4): 844–856, 1995.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999.
3. P.N. Balister, E. Györi, J. Lehel, and R.H. Schelp. Longest Paths in Circular Arc Graphs. Technical report, U. of Memphis, `www.msci.memphis.edu/preprint.html`, 2002.
4. C. Berge. *Hypergraphs*. Elsevier, 1989.
5. A.A. Bertossi. Finding Hamiltonian Circuits in Proper Interval Graphs. *Info. Proc. Lett.*, 17(2): 97–101, 1983.
6. A.A. Bertossi and M.A. Bonuccelli. Hamiltonian Circuits in Interval Graph Generalizations. *Info. Proc. Lett.*, 23: 195–200, 1986.
7. A. Björklund and T. Husfeldt. Finding a Path of Superlogarithmic Length. *SIAM J. Comput.*, 32(6): 1395–1402, 2003.
8. K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *J. Comput. Syst. Sci.*, 13: 335–379, 1976.
9. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
10. R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen. On Computing a Longest Path in a Tree. *Info. Proc. Lett.*, 81: 93–96, 2002.
11. P. Damaschke. The Hamiltonian Circuit Problem for Circle Graphs is NP-complete. *Info. Proc. Lett.*, 32: 1–2, 1989.

12. P. Damaschke. Paths in Interval Graphs and Circular Arc Graphs. *Discr. Math.*, 112: 49–64, 1993.
13. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.
14. H.N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proc. 1st Ann. ACM-SIAM Symp. on Discr. Algo.*, 434–443. ACM, 1990.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
16. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, 2/e. Ann. Discr. Math., 57. Elsevier, 2004.
17. D. Hochbaum (eds.). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1995.
18. D. Karger, R. Motwani, and G.D.S. Ramkumar. On Approximating the Longest Path in a Graph. *Algorithmica*, 18: 82–98, 1997.
19. N. Korte and R.H. Möhring. An Incremental Linear-Time Algorithm for Recognizing Interval Graphs. *SIAM J. Comput.*, 18(1): 68–81, 1989.
20. N.V.R. Mahadev and U.N. Peled. Longest Cycles in Threshold Graphs. *Discr. Math.*, 135: 169–176, 1994.
21. B. Monien. How to Find Long Paths Efficiently. *Ann. Discr. Math.*, 25: 239–254, 1985.
22. H. Müller. Hamiltonian Circuit in Chordal Bipartite Graphs. *Disc. Math.*, 156: 291–298, 1996.
23. M.G. Scutellà. An Approximation Algorithm for Computing Longest Paths. *European J. Oper. Res.*, 148(3): 584–590, 2003.
24. J. Spinrad, A. Brandstädt, and L. Stewart. Bipartite Permutation Graphs. *Discr. Appl. Math.*, 18: 279–292, 1987.
25. V.V. Vazirani. *Approximation Algorithms*. Springer, 2001.
26. S. Vishwanathan. An Approximation Algorithm for Finding a Long Path in Hamiltonian Graphs. In *Proc. 11th Ann. ACM-SIAM Symp. on Discr. Algo.*, 680–685. ACM, 2000.