Core and Conditional Core Path of Specified Length in Special Classes of Graphs

S. Balasubramanian, S. Harini, and C. Pandu Rangan

Department of Computer Science and Engineering,
IIT Madras, India 600036
{balu2901,harini.santhanam}@gmail.com, rangan@iitm.ernet.in

Abstract. A core path of a graph is a path P in G that minimizes $d(P) = \sum_{v \in V} d(v, P)w(v)$. In this paper, we study the location of core path of specified length in special classes of graphs. Further, we extend our study to the problem of locating a core path of specified length under the condition that some existing facilities are already located (known as conditional core path of a graph). We study both the problems stated above in vertex weighted bipartite permutation graphs, threshold graphs and proper interval graphs and give polynomial time algorithms for the core path and conditional core path problem in these classes. We also establish the NP-Completeness of the above problems in the same classes of graphs when arbitrary positive weights are assigned to edges.

Keywords: Core path, Conditional core path, Bipartite permutation graphs, Threshold graphs, Proper Interval graphs.

1 Introduction

The objective of any facility location algorithm in a network is to locate a site/facility that optimizes some criterion. The criteria that have been most generally employed are the *minimax* and *minisum* criteria. In the *minimax* criteria, the distance of the farthest vertex from the facility is minimized. In the *minisum* criteria, the sum of the distances of the vertices of the graph from the facility is minimized. Many practical facility location problems however, involve the location of several facilities rather than just one facility. In particular, the problem of locating a path-shaped or tree-shaped facility has received wide attention due to applications in metro rail routing, pipeline planning, laying irrigation ditches etc. When the path to be located is such that it satisfies the *minisum* criterion,(i.e the sum of the distances of the vertices of the graph from the path is minimized) then we call the path a core path which we will define formally in this section.

Often, it might happen that some facilities have already been located and the new facilities should be located such that the *minisum* criterion is optimized taking into account both the already existing facilities and the newly located facilities. For example, a district may already be equipped with a gas pipeline and

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 262–273, 2009.

we should plan the layout of a new pipeline. Any user can obtain a connection either from the old or the new pipeline depending on which one is closer to him. Where should we lay this new pipeline such that it minimizes the sum of the distances of the users in the district from the pipelines (old and new). This problem is called the conditional facility location problem. When the new facility to be established is a path, the problem is known as conditional-core problem (defined formally later). The network can be assigned vertex and edge weights. Vertex weights can denote the population of a locality and edge weights the distance between two localities. In this paper, we study the problem of locating the core path and conditional core path of a specified length in bipartite permutation graphs, threshold graphs and proper interval graphs. Specifically, we give polynomial time algorithms for both the problems in the above classes of graphs when vertices are assigned arbitrary positive weights and edges are assigned unit weights. When the edges are assigned arbitrary weights, we prove the NP-Completeness of both the problems in all the three classes of graphs.

Let $G = (V, E)$ be a simple, undirected, connected, weighted (positive vertex and edge weights) graph. The length of a path P is defined as sum of weights of edges in P. Let $d(u, v)$ be the shortest distance between two vertices u and v. The set of vertices of a path P is denoted by $V(P)$ and the set of vertices of $V(P) \cap X$ for any set $X \subseteq V$, is denoted by $V_X(P)$. We extend the notion of distance between a pair of vertices in a natural way to the notion of distance between a vertex and a path. The distance between a vertex v and path P is $d(v, P) = \underset{u \in V(P)}{Min}\, d(v, u)$. If $v \in V(P)$, then $d(v, P)$ is zero. The **cost of a path** P denoted by $d(P)$ is $\sum_{v \in V} d(v, P)w(v)$, where $w(v)$ is the weight of the vertex v.

Definition 1. *[9] The* ***Core path or Median path*** *of a graph G is a path P that minimizes $d(P)$.* □

Definition 2. *[8] Let $\mathcal{P}_l$ be the set of all paths of length l in G. The* ***Core path of length l*** *of a graph G is a path $P \in \mathcal{P}_l$ where $d(P) \leq d(P')$ for any path $P' \in \mathcal{P}_l$.* □

Let S denote the set of vertices in which facilities have already been deployed. The **conditional cost of a path** P denoted by
$d^c(P) = \sum_{v \in V} min(d(v, P), d(v, S))w(v)$, where $d(v, S) = \underset{u \in S}{Min}\, d(v, u)$.

Definition 3. *Let $\mathcal{P}_l^S$ be the set of all paths of length l in G such that $V(P) \cap S = \emptyset$. The* ***Conditional Core path of length l*** *of a graph G is a path $P \in \mathcal{P}_l^S$ where $d^c(P) \leq d^c(P')$ for any path $P' \in \mathcal{P}_l^S$.* □

Previous work: So far, the core-path problem has been analyzed only in trees and recently in grid graphs. In [5] Hakimi, Schmeichel, and Labb'e have given 64 variations of the above problem and have also proved that finding the core path is NP-Hard on arbitrary graphs. In [9] Morgan and Slater give a linear time algorithm for finding core path of a tree with arbitrary edge weights. In [7], [8] Minieka et al. consider the problem of finding the core path with a constraint on the length of the path. Their work considers locating path or tree shaped facilities of specified length in a tree and they present an $O(n^3)$ algorithm. In [10] Peng and Lo extend their work by giving a $O(nlogn)$ sequential and $O(log^2(n))$ parallel algorithm using $O(n)$ processors for finding the core path of a tree(unweighted) with a specified length. In [3] Becker et al. give an $O(nl)$ algorithm for the unweighted case and a $O(nlog^2n)$ for the weighted case for trees. [2] presents a study of the core path in grid graphs. In [1] Alstrup et al. give an $O(n\ min\{\log n\ \alpha(n,n), l\})$ algorithm for finding the core path of a tree. The conditional location of path and tree shaped facilities have also been extensively studied on trees. In [11], Tamir et. al. prove that the continuous conditional median subtree problem is NP-hard and they develop a fully polynomial time approximation scheme for the same. They also provide an $O(n \log^2 n)$ algorithm for the discrete conditional core path problem with a length constraint and an $O(n^2)$ algorithm for the continuous conditional core path problem with a length constraint. They also study the conditional location center paths on trees. Further, in [13], Wang et al improve the algorithms in [11] for both discrete and continuous conditional core path problem with a length constraint. For the discrete case, they presented an $O(n \log n)$ algorithm and for the continuous case they presented an $O(n \log n\ \alpha(n,n))$ algorithm.

The rest of the paper is organized as follows. In section 2, we give a polynomial time algorithm to solve the core path problem in vertex weighted bipartite permutation graphs. In section 3, we solve the conditional core problem in vertex weighted bipartite permutation graphs. We prove the NP-Completeness of the core path problem in bipartite permutation graphs with arbitrary edge weights in *Appendix*. We also briefly present our solution for core and conditional core path problems in unit edge weighted threshold graphs and proper interval graphs in *Appendix*.

2 Core Path of a Bipartite Permutation Graph with Vertex Weights

2.1 Preliminaries

Let $\pi = (\pi_1,\pi_2,\ldots\pi_n)$ be a permutation of the numbers $1, 2\ldots n$. The graph $G(\pi) = (V, E)$ is defined as follows: $V=\{1, 2\ldots n\}$ and $(i, j) \in E \iff (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ where π_i^{-1} is the position of number i in the sequence π. An undirected graph G is called a permutation graph iff there exists a permutation π such that G is isomorphic to $G(\pi)$. A graph is a bipartite permutation graph, if it is both a bipartite graph and a permutation graph. We assume that the bipartite permutation graph $G = (X, Y, E)$ has unit edge weights and arbitrary vertex weights. We present a polynomial time algorithm for solving core path problem on G. The next definition is taken from [6].

Definition 4. *A strong ordering of the vertices of a bipartite graph* $G=(X, Y, E)$ *consists of an ordering* $<_x$ *of* X *and an ordering* $<_y$ *of* Y *such that for all* (x, y), $(x', y') \in E$, *where* x, $x' \in X$ *and* y, $y' \in Y$, $x <_x x'$ *and* $y' <_y y$ *imply* (x, y') *and* $(x', y) \in E$. □

From now on, we drop the subscripts for $<_x$ and $<_y$ and denote them by $<$, interpreting the meaning from the context. For any two vertices a and b that are in the same partition of the bipartite graph, if $a < b$, we say a is *above* b and b is *below* a. Given an edge (x_i, y_j), we call the union of all the vertices *above* x_i and *above* y_j as *above* the edge (x_i, y_j).

[6] A bipartite graph B is a bipartite permutation graph iff *it admits a strong ordering.*

Ordered Paths: Any path $P = v_1v_2v_3v_4v_5v_6\ldots$ of a bipartite permutation graph G is said to be an *ordered* path *iff* $v_1 < v_3 < v_5 < \ldots$ and $v_2 < v_4 < v_6 < \ldots$. The following lemma relates every path with an ordered path of same vertex set.

Lemma 1. *In a bipartite permutation graph, for every path P, there exists an ordered path Q, such that $V(P)$=$V(Q)$.*

Proof in Appendix.

In the above lemma, since $V(P)$=$V(Q)$, it follows that $d(P)$=$d(Q)$ and $d^c(P) = d^c(Q)$. So, for every path there is an ordered path with the same cost and the same set of vertices. Therefore, we consider only ordered paths henceforth.

2.2 Algorithm

Brief overview of the algorithm: The naive technique searches the set of all paths in the graph to find the core path. Since we have proved that for every path there exists an ordered path with the same set of vertices, we can cut down on the search space for paths to ordered paths alone. The set of vertices adjacent to a vertex u is called the neighborhood of u, and is written as $N(u)$. Let $L(u)$ and $R(u)$ be the vertices with smallest and largest index in $N(u)$ according to the strong ordering. Let $P = v_1v_2v_3\ldots v_k$ be an ordered path in G. Then the edge (v_1, v_2) is called the *first edge* of P and the edge (v_{k-1}, v_k) is called the *last edge* of P. For every ordered path P, let $\alpha_r(P)$ denote the path obtained by taking the *first* r edges of the ordered path P. In case, the path does not contain r edges, then $\alpha_r(P) = \perp$. We define $d(\perp) = \infty$. Also, $d(\alpha_r(P)v) = \infty$, when $\alpha_r(P) = \perp$, where $\alpha_r(P)v$ denotes the concatenation of $\alpha_r(P)$ and vertex v.

Remark 1. For every edge $(x, y) \in E$ let $path^l_{xy}$ denote the path with (x, y) as its *last edge* such that it is the minimum cost ordered path of length l with (x, y) as its *last edge*. The $\underset{(x,y)\in E}{Min}\ d(path^l_{xy})$ will yield us the cost of core path of length l of the graph G.

Remark 2. Let G_{ij} be a graph induced by the vertices $\{x_1, x_2, \ldots, x_i\}$ and $\{y_1, y_2, \ldots, y_j\}$. The ordered path with (x_i, y_j) as the *last edge* cannot contain a vertex v such that $x_i < v$ or $y_j < v$. Hence for every edge (x_i, y_j), it is sufficient to consider the ordered paths in G_{ij} and not the entire graph G.

Finding the cost of any ordered path: Here we give a method to compute the cost of any ordered path efficiently. For all edges, $(x_i, y_j) \in E$ we define, $U(x_i, y_j) = \{v \in V | (v < x_i \text{ or } v < y_j)\}$, $W_U(x_i, y_j) = \sum_{v \in U(x_i, y_j)} w(v)$, $UAdj(x_i, y_j) = \{v \in V | (v < x_i \text{ and } (v, y_j) \in E) \text{ or } (v < y_j \text{ and } (v, x_i) \in E)\}$, $W_{UAdj}(x_i, y_j) = \sum_{v \in UAdj(x_i, y_j)} w(v)$,
$USUM(x_i, y_j) = \sum_{v \in U(x_i, y_j)} min(d(v, x_i), d(v, y_j))w(v)$.
Intuitively, $U(x_i, y_j)$ denotes the set of all vertices that are *above* (x_i, y_j) as per the strong ordering. $W_U(x_i, y_j)$ is the sum of the weight of vertices in the set $U(x_i, y_j)$. The set $UAdj(x_i, y_j)$ is the set of all vertices that are *above* (x_i, y_j) and are adjacent to either x_i or y_j. $USUM(x_i, y_j)$ denotes the sum of the costs incurred by all the vertices *above* (x_i, y_j) in either reaching x_i or y_j(whichever is closer). When all terms given above are preprocessed and stored for all the edges, the cost of any path P can be computed in $O(1)$ time. We know that $W_U(x_1, y_1)$ = 0. We can compute $W_U(x_i, y_j)$ and $W_{UAdj}(x_i, y_j)$ for all $(x_i, y_j) \in E$ in $O(|E|)$ time using the equations:

$$W_U(x_i, y_j) = \begin{cases} W_U(x_{i-1}, y_j) + w(x_{i-1}) & \text{if } (x_{i-1}, y_j) \in E \\ W_U(x_i, y_{j-1}) + w(y_{j-1}) & \text{if } (x_{i-1}, y_j) \notin E \text{ but } (x_i, y_{j-1}) \in E \end{cases}$$

$W_{UAdj}(x_i, y_j) = W_U(x_i, y_j) - W_U(L(y_j), L(x_i))$.

Lemma 2. *The following equation can be used to compute the value of* $USUM(x_i, y_j)$ *iteratively*

Let $x' = L(y_j)$ *and* $y' = L(x_i)$
$USUM(x_i, y_j) = USUM(x', y') + W_U(x', y') + W_{UAdj}(x_i, y_j)$.

Proof. We will first give an intuitive sketch of the proof. $USUM(x_i, y_j)$ is the cost incurred by all the vertices *above* (x_i, y_j) (i.e. vertices in $U(x_i, y_j)$) in reaching the nearer of x_i or y_j. This cost can be viewed as a sum of two terms : the cost due to vertices adjacent to and above (x_i, y_j) (i.e. in $UAdj(x_i, y_j)$) and the cost due to vertices *above* (x', y') (i.e. in $U(x', y')$). All the vertices in $UAdj(x_i, y_j)$ are at a distance of one from x_i or y_j and hence the cost incurred by them is $W_{UAdj}(x_i, y_j)$. All the vertices in $U(x', y')$ have to reach one of x' or y' to reach x_i or y_j. The cost incurred to reach x' or y' is $USUM(x', y')$. From (x', y'), all the vertices have to travel a distance of one to reach x_i or y_j and hence the cost incurred is $W_U(x', y')$. This completes the proof. We give an inductive proof below.

By definition, we know that $USUM(x_1, y_1) = 0$. We can verify by inspection that the expression for $USUM$ is true for $(x_1, y) \in E$ $\forall y$ satisfying $L(x_1) \leq y \leq R(x_1)$. By induction hypothesis we assume that $USUM$ is correctly computed for all (x_i, y), $1 \leq i \leq r - 1$ and $y = L(x_i)$ to $R(x_i)$. We prove the claim for the edge (x_r, y_j) where $y_j = L(x_r)$ and the proof follows similarly for the case where $L(x_r) < y_j$. By definition,

$USUM(x_r, y_j) = \sum_{v \in U(x_r, y_j)} min(d(v, x_r), d(v, y_j))w(v)$.

$USUM(x_r, y_j) = \sum_{v \in U(x', y')} min(d(v, x_r), d(v, y_j))w(v) + W_{UAdj}(x_r, y_j).$
where $x' = L(y_j)$ and $y' = L(x_r)$.

For every $v \in U(x', y')$, $min(d(v, x'), d(v, y')) \leq min(d(v, x''), d(v, y''))$ where $x' < x''$ and $y' < y''$. This statement implies each vertex $v \in U(x', y')$ can reach one of (x_r or y_j) only through one of (x' or y'). Therefore we have that,

$$\sum_{v \in U(x', y')} min(d(v, x_r), d(v, y_j))w(v) = USUM(x', y') + W_U(x', y').$$

The cost $W_U(x', y')$ is attributed to the extra distance of length one per vertex in $U(x', y')$ to reach x_r or y_j. ☐

For all edges $(x_i, y_j) \in E$ we define, $B(x_i, y_j) = \{v \in V | (x_i < v \text{ or } y_j < v)\}$, $W_B(x_i, y_j) = \sum_{v \in B(x_i, y_j)} w(v)$, $BAdj(x_i, y_j) = \{v \in V | (x_i < v \text{ and } (v, y_j) \in E)$ or $(y_j < v \text{ and } (v, x_i) \in E\ )\}$, $W_{BAdj}(x_i, y_j) = \sum_{v \in BAdj(x_i, y_j)} w(v)$,
$BSUM(x_i, y_j) = \sum_{v \in B(x_i, y_j)} min(d(v, x_i), d(v, y_j))w(v).$

Similar to $USUM$, we can compute $BSUM$ value for every $(x, y) \in E$ in $O(|E|)$ time. We define,

$$W = \sum_{v \in V} w(v) \text{ and } W(P) = \sum_{v \in V(P)} w(v) \text{ for any path } P.$$

W can be computed in $O(|V|)$ time and we give a method to calculate $W(P)$ in *Lemma* 4.

The following *Lemma* gives a method to compute cost of any ordered path.

Lemma 3. *For any ordered path P, the cost $d(P)$ can be computed in $O(1)$ time after $O(|E|)$ preprocessing.*

Proof. Let x_a and x_b be the vertices of $V_X(P)$ such that they have respectively the smallest and largest index in the strong ordering. Similarly, let y_a and y_b be the vertices of $V_Y(P)$ such that they have respectively the smallest and largest index in the strong ordering. We claim that

$$d(P) = USUM(x_a, y_a) + BSUM(x_b, y_b) + W - W(P) - W_B(x_b, y_b) - W_U(x_a, y_a)$$

Using the above formula, $d(P)$ can be computed in $O(1)$ time after $O(|E|)$ preprocessing. For $v \in \{X \cup Y\}$ - $V(P)$ - $B(x_b, y_b)$ - $U(x_a, y_a)$, we know that $d(v, P) = 1$. For all of these vertices, the total cost incurred is W - $W(P)$ - $W_B(x_b, y_b)$ - $W_U(x_a, y_a)$. Value $USUM(x_a, y_a)$ accounts $\forall v \in U(x_a, y_a)$ and $BSUM(x_b, y_b)$ accounts $\forall v \in B(x_b, y_b)$. Note that we still have not calculated $W(P)$ which is necessary for calculating $d(P)$. We specify the method to compute this in *Lemma* 4. ☐

Finding the core path: As noted by *Remark* 1, we have to find $path^l_{xy}$ $\forall xy \in E$. Let $P^l_{x_i y_j}$ denote an ordered path of length l and of minimum cost among all ordered paths of length l in G_{ij} with (x_i, y_j) as the *last edge* and y_j being the vertex of degree one in the path P. Let $P^l_{y_jx_i}$ denote an ordered path of length l and of minimum cost among all ordered paths of length l in G_{ij} with (x_i, y_j) as the *last edge* and x_i being the vertex of degree one in the path P. From $P^l_{x_iy_j}$ and $P^l_{y_jx_i}$, we can compute $path^l_{x_iy_j}$ which is defined such that $d(path^l_{x_iy_j}) = Min\{d(P^l_{x_iy_j}), d(P^l_{y_jx_i})\}$. The path corresponding to the cost $\underset{xy\in E}{Min}\, d(path^l_{xy})$ will yield a core path of length l of graph G.

$P^1_{x_iy_j} = x_iy_j$, $P^1_{y_jx_i} = y_jx_i$ and their costs can be computed using *Lemma* 3. We now give, equations to compute the path of least cost of length r from the knowledge of the same for length $r-1$. Initially we give the equations that follow from definition and later we give a dynamic programming equation to compute the same efficiently.

Lemma 4. *The following equations can be used to find the costs of* $P^r_{x_iy_j}$, $P^r_{y_jx_i}$ $\forall r \geq 2$. *In graph* G_{ij}, $\forall (x_i, y_j) \in E(G_{ij})$ *we have that,*

$$d(P^r_{x_iy_j}) = \begin{cases} \underset{\forall (y_k,x_i)\in E, k<j}{Min} d(P^{r-1}_{y_kx_i}y_j) & \textit{if such } y_k\textit{'s exist} \\ \infty & \textit{otherwise} \end{cases} \tag{1}$$

$$= \begin{cases} Min\{d(P^{r-1}_{y_{j-1}x_i}y_j),\ d(\alpha_{r-1}(P^r_{x_iy_{j-1}})y_j)\} & \textit{if } j > 1\ ,\ (x_i, y_{j-1}) \in E \\ \infty & \textit{otherwise} \end{cases} \tag{2}$$

$$d(P^r_{y_jx_i}) = \begin{cases} \underset{\forall (x_k,y_j)\in E, k<i}{Min} d(P^{r-1}_{x_ky_j}x_i) & \textit{if such } x_k\textit{'s exist} \\ \infty & \textit{otherwise} \end{cases} \tag{3}$$

$$= \begin{cases} Min\{d(P^{r-1}_{x_{i-1}y_j}x_i),\ d(\alpha_{r-1}(P^r_{y_jx_{i-1}})x_i)\} & \textit{if } i > 1\ ,\ (x_{i-1}, y_j) \in E \\ \infty & \textit{otherwise} \end{cases} \tag{4}$$

$$d(path^r_{x_iy_j}) = Min\{d(P^r_{x_iy_j}),\ d(P^r_{y_jx_i})\} \tag{5}$$

Also, for calculating the cost $d(P)$ *of a path* P, *we need* $W(P)$. *This can be calculated while we construct the path using the following equations*

$$W(P^{r-1}_{y_kx_i}y_j) = W(P^{r-1}_{y_kx_i}) + w(y_j),\ W(P^{r-1}_{x_ky_j}x_i) = W(P^{r-1}_{x_ky_j}) + w(x_i)$$

Proof. We will prove equations (1) and (2). The proofs for (3) and (4) follow analogously.

Proof for (1)

(1) states that we can the compute minimum cost path of length r having (x_i, y_j) as the last edge, by considering minimum cost paths of length $r-1$ having (y_k, x_i) as last edge $\forall k < j$. We choose the minimum cost path among them, and append it with (x_i, y_j) to get the required path. Note that y_k's are chosen such

that $k < j$ because $\forall k > j$, the path is not ordered and we have proved that it is enough to search the set of all ordered paths to find the core path. The number of operations in (1) to compute $d(P^r_{x_i y_j})$ can be as high as $degree(x_i)$

Proof for (2)
For computing $d(P^r_{x_i y_j})$, we note that it is just enough to consider the path of least cost of length $r-1$ having (y_{j-1}, x_i) as the last edge and the path of least cost of length $r-1$ having (y_k, x_i) as the last edge $\forall\ k < j-1$. The latter term is $d(\alpha_{r-1}(P^r_{x_i y_{j-1}})y_j)$. Note that this value would have already been computed while computing $d((P^r_{x_i y_{j-1}})y_j)$ and hence no special effort is required now.

Claim. $d(\alpha_{r-1}(P^r_{x_i y_{j-1}})y_j) = \underset{\forall (y_k, x_i) \in E | k < j-1}{Min} d(P^{r-1}_{y_k x_i} y_j)$

It is clear that if the above claim is established, then equations (2) and (1) will become equivalent and (2) is proven.

Let $\alpha_{r-1}(P^r_{x_i y_{j-1}})$ be such that it has (y_t, x_i) as the *last edge*.

$$\Rightarrow d(P^{r-1}_{y_t x_i} y_{j-1}) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i} y_{j-1})$$
$$\Rightarrow d(P^{r-1}_{y_t x_i}) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i})$$
$$\Rightarrow d(P^{r-1}_{y_t x_i} y_j) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i} y_j)$$

which is precisely the statement of our *claim*. Thus the number of operations if (2) is used to compute $d(P^r_{x_i y_j})$ is just two. $\square$

Theorem 1. *The Algorithm computes the core path of a bipartite permutation graph in $O(l|E|)$ time.*

Proof. The algorithm computes the values of $d(path^l_{xy})$ for each edge $(x, y) \in E$ using *Lemma* 4 and then computes the minimum cost path by finding $\underset{(x,y) \in E}{Min}$ $d(path^l_{xy})$ and hence the correctness follows.

Time complexity: For a given length and a given edge (x, y) or $(y, x) \in E$, the algorithm takes $O(1)$ time to compute the value of $d(P^{length}_{xy})$ or $d(P^{length}_{yx})$. Therefore to compute the value of $d(P^l_{xy})$ and $d(P^l_{yx})$ for all edges (x, y) and $(y, x) \in E$, it takes $O(l|E|)$ time. To compute the value of $Mincost$ from these values, the algorithm takes utmost $O(|E|)$ time. $\square$

3 Conditional Core Path of a Bipartite Permutation Graph with Vertex Weights

In this section, we give a polynomial time algorithm for the problem of finding conditional core path of a bipartite permutation graph $G = (X, Y, E)$(G has arbitrary vertex weights and unit edge weights). Let $S\ (\subset V)$ denote the subset of vertices of V where the facilities have already been located. As already defined, the **conditional cost of a path** P denoted by $d^c(P) = \sum_{v \in V} Min(d(v,P), d(v,S))w(v)$, where $d(v,P) = \underset{u \in P}{Min}\ d(v,u)$ and $d(v,S) = \underset{v' \in S}{Min}\ d(v,v')\ \forall v \in V$.

Computing d(v,S): Let $N(S) - S = \{v \in V - S | (v,u) \in E, u \in S\}$. We give here an efficient method to compute $d(v,S)\ \forall\ v \in V$.

$\forall\ v \in S$, initialize $d(v,S) = 0$.
$\forall\ v \in N(S) - S$ initialize $d(v,S) = 1$.
For all the remaining vertices, initialize $d(v,S) = \infty$.

We first give a method to find $d(x_i, S)\ \forall x_i \in X$. A similar procedure can be used to find $d(y_j, S)\ \forall y_j \in Y$.

1. For every vertex x_i, we define two vertices x_i^a, $x_i^b \in X \cap S$ such that $x_i^a < x_i < x_i^b$. Also
   - $d(x_i, x_i^a) \leq d(x_i, x')\ \forall x' \in X \cap S$ and $x' < x_i$
   - $d(x_i, x_i^b) \leq d(x_i, x')\ \forall x' \in X \cap S$ and $x' > x_i$
2. For every vertex x_i, we define two vertices x_i^c, $x_i^d \in Y \cap S$ such that $x_i^c < L(x_i) \leq R(x_i) < x_i^d$. Also
   - $d(x_i, x_i^c) \leq d(x_i, y')\ \forall y' \in Y \cap S$ and $y' < L(x_i)$
   - $d(x_i, x_i^d) \leq d(x_i, y')\ \forall y' \in Y \cap S$ and $y' > R(x_i)$

Clearly, if $d(x_i, S) \neq 0$ or 1, then $d(x_i, S) =$ Min $\{d(x_i, x_i^a)$, $d(x_i, x_i^b)$, $d(x_i, x_i^c)$, $d(x_i, x_i^d)\}$. If we find x_i^a, x_i^b, x_i^c and x_i^d, we can evaluate the above Min function and find $d(x_i, S)$. We now state two lemmas from [4] as the following *Remark*.

Remark 3

1. Suppose $i < j < k$. Then $d(x_i, x_j) \leq d(x_i, x_k)$.
2. Suppose x_i is not adjacent to y_j or y_k, $x_i < L(y_j)$, and $j < k$. Then $d(x_i, y_j) \leq d(x_i, y_k)$.

The above *Remark* characterizes x_i^a to be the maximum indexed vertex in $X \cap S$, that lie above x_i. Similarly, x_i^b is the minimum indexed vertex in $X \cap S$, that lie below x_i. Also, x_i^c is the maximum indexed vertex in $Y \cap S$, that lie above $L(x_i)$ and x_i^d is the minimum indexed vertex in $Y \cap S$, that lie below $R(x_i)$. We use the above property to find $d(x_i, x_i^a)$ efficiently for all the vertices. We can find x_i^a, $\forall x_i \in X$, using the following code.

1. $s_{up} = NIL$.
2. For $i = 1$ to $|X|$ If $x_i \in S$ then $s_{up} = x_i$. Else $x_i^a = s_{up}$.

From [4] we know that, for a bipartite permutation graph, after $O(n^2)$ preprocessing, the value of the shortest distance between any given pair of vertices can be computed in $O(1)$ time. Hence we can compute $d(x_i, x_i^a)\ \forall x_i \in X$, in $O(|X|)$ time. Since $x_i^c = (L(x_i))^a$, we note that $d(x_i, x_i^c) = 1 + d(L(x_i), (L(x_i))^a)$. Similarly we calculate all the required values and $d(v,S)\ \forall v \in V$ in $O(|V| + |E|)$ time.

Finding the cost of any ordered path: We first set $w(v) = 0, \forall v \in S$ because for any path P, cost due to these vertices is zero and setting $w(v) = 0$ simplifies some calculations that follow. We now show that for a path P, the conditional cost $d^c(P)$ can be calculated using *Lemma* 3 with a modification to the definition of $USUM$ and $BSUM$ to comply with the definition of conditional cost of a path. For all edges, $(x_i, y_j) \in E(G)$ we define,

$$USUM(x_i, y_j) = \sum_{v \in U(x_i, y_j)} min(d(v, x_i), d(v, y_j), d(v, S))w(v)$$

In order to calculate the value of this newly defined $USUM(x_i, y_j)$ efficiently, we define the following terms

$$\tau(x_i, y_j) = \{v : v \in U(x_i, y_j)\ ,\ d(v, (x_i, y_j)) < d(v, S)\}$$
$$TOADD(x_i, y_j) = \sum_{v \in \tau(x_i, y_j)} w(v)$$

Lemma 5. *The following equation can be used to compute $USUM(x_i, y_j)$ iteratively:*

$USUM(x_1, y_1) = 0$. Let $x' = L(y_j)$ and $y' = L(x_i)$.
$USUM(x_i, y_j) = USUM(x', y') + TOADD(x', y') + W_{UAdj}(x_i, y_j)$.

Proof in Appendix

In order to calculate $TOADD(x_i, y_j)$ we need some more definitions which we present below.

$Q(x_i, y_j) = \{v \in U(x_i, y_j) :\ d(v, x_i) = d(v, S)$ and $d(v, y_j) = d(v, S) + 1\}$
Also, $W(Q(x_i, y_j)) = \sum_{v \in Q(x_i, y_j)} w(v)$.

Algorithm 1. Algorithm to compute $W(Q(x_i, y_j))$

```
for all v ∈ V do
  for i = 1 to |X| do
    for j = 1 to |Y| do
      if  d(v, x_i) = d(v, S) and d(v, y_j) = d(v, S)+1  then
        W(Q(x_i, y_j)) = W(Q(x_i, y_j)) + w(v)
      else if  d(v, y_j) = d(v, S) and d(v, x_i) = d(v, S)+1  then
        W(Q(y_j, x_i)) = W(Q(y_j, x_i)) + w(v)
      end if
    end for
  end for
end for
```

Lemma 6. *Algorithm 1 computes $W(Q(x_i, y_j))$ in $O(|V|.|E|)$ time.*

Proof. From [4] we know that, for a bipartite permutation graph, after $O(n^2)$ preprocessing, the value of the distance between any given pair of vertices can be computed in $O(1)$ time. From this, it immediately follows that the time complexity of Algorithm 1 is $O(|V|.|E|)$ □

Lemma 7. *The following equation can be used to compute $TOADD(x_i, y_j)$ iteratively.*

$$TOADD(x_i, y_j) = \begin{cases} 0 & \text{if } i, j = 1 \\ TOADD(x_{i-1}, y_j) + w(x_{i-1}) - W(Q(y_j, x_i)) + W(Q(y_j, x_{i-1})) & \text{if } i \neq 1,\ j = index(L(x_i)) \\ TOADD(x_i, y_{j-1}) + w(y_{j-1}) - W(Q(x_i, y_j)) + W(Q(x_i, y_{j-1})) & \text{otherwise} \end{cases}$$

Proof in Appendix
Note that we compute $TOADD(x_i, y_j)$ along-side while computing $USUM$. We similarly calculate $BSUM$ where

$$BSUM(x_i, y_j) = \sum_{v \in B(x_i, y_j)} min(d(v, x_i), d(v, y_j), d(v, S))w(v).$$

Lemma 8. *For any ordered path P, the conditional cost can be computed in $O(1)$ time after $O(|V||E|)$ preprocessing.*

Proof. The conditional cost for a path P is given by $d(P) = USUM(x_a, y_a)$ + $BSUM(x_b, y_b)$ + W - $W(P)$ - $W_B(x_b, y_b)$ - $W_U(x_a, y_a)$. Here $USUM$ and $BSUM$ is as defined in this section. All other definitions is same as given in *Lemma* 3 and the proof also follows in exactly same fashion. □

Finding the Conditional Core Path: The conditional core path should be vertex disjoint from S by definition. Let H be the graph induced by $V(G)$ - S. We ought to search for the conditional core path in H. However, while calculating the conditional cost of the path, we must use the vertices in the entire graph G. Note that, we can modify *Remark* 1 to use conditional cost as follows.

Remark 4. For every edge $(x, y) \in E(H)$ let $path^l_{xy}$ denote the path with (x, y) as its *last edge* such that it is the minimum conditional cost ordered path of length l with (x, y) as its *last edge*.

Now, we can use the dynamic programming equations given in *Lemma* 4 on the graph H (instead of G) to find the conditional core path due to the validity of *Remark* 4. But to calculate the conditional cost, we use the definitions stated in this section and *Lemma* 8.

Theorem 2. *The conditional core path of a bipartite permutation graph can be computed in $O(|V||E|)$ time.* □

4 Conclusion

In this paper, we have presented an $O(l|E|)$ time algorithm for finding the core path of specific length l in vertex weighted bipartite permutation graphs, threshold graphs and proper interval graphs. We have extended our study of core path problem to the conditional core path problem on the same graph classes. For the conditional core path problem of specified length, we have presented $O(l|E|)$ time algorithms for threshold and proper interval graphs and $O(|V||E|)$ time algorithm for bipartite permutation graphs. In all the three classes of graphs, due to their inherent property of vertex ordering we were able to conceptualize the notion of ordered paths. However, such a notion of ordered paths (i.e. for every path there exists an ordered path of the same vertex set) is not valid on interval or permutation graphs. Also, the complexity of the longest path problem is still unresolved in interval graph [12] and thus even the existence of a path of length l in interval graphs is still unresolved. Therefore, the techniques used in this paper cannot be directly applied to interval or permutation graphs and hence the problem of finding core path in them is an interesting open problem in this direction.

References

1. Alstrup, S., Lauridsen, P.W., Sommerlund, P., Thorup, M.: Finding cores of limited length. In: Rau-Chaplin, A., Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 45–54. Springer, Heidelberg (1997)
2. Becker, R., Lari, I., Scozzari, A., Storchi, G.: The location of median paths on grid graphs. Annals of Operations Research 150(1), 65–78 (2007)
3. Becker, R.I., Chang, Y.I., Lari, I., Scozzari, A., Storchi, G.: Finding the l-core of a tree. Discrete Appl. Math. 118(1-2), 25–42 (2002)
4. Chen, L.: Solving the shortest-paths problem on bipartite permutation graphs efficiently. Inf. Process. Lett. 55(5), 259–264 (1995)
5. Hakimi, S.L., Schmeichel, E.F., Labb'e, M.: On locating path- or tree-shaped facilities on networks. networks 23(6), 543–555 (1993)
6. Brandstadt, A., Spinrad, J., Stewart, L.: Bipartite Permutation Graphs. Discrete Applied Mathematics 18, 279–292 (1987)
7. Minieka, E.: The optimal location of a path or a tree in a tree network. Networks 15, 309–321 (1985)
8. Minieka, E., Patel, N.H.: On finding the core of a tree with a specified length. J. Algorithms 4(4), 345–352 (1983)
9. Morgan, C.A., Slater, P.J.: A linear algorithm for a core of a tree. J. Algorithms 1(3), 247–258 (1980)
10. Peng, S., Lo, W.-T.: Efficient algorithms for finding a core of a tree with a specified length. J. Algorithms 20(3), 445–458 (1996)
11. Tamir, A., Puerto, J., Mesa, J.A., Rodríguez-Chía, A.M.: Conditional location of path and tree shaped facilities on trees. J. Algorithms 56(1), 50–75 (2005)
12. Uehara, R., Uno, Y.: On computing longest paths in small graph classes. Int. J. Found. Comput. Sci. 18(5), 911–930 (2007)
13. Wang, B.-F., Ku, S.-C., Hsieh, Y.-H.: The conditional location of a median path. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 494–503. Springer, Heidelberg (2002)