BIPARTITE PERMUTATION GRAPHS

Jeremy SPINRAD
Dept. of Computer Science, Vanderbilt Univ., Nashville, TN 37240, USA

Andreas BRANDSTÄDT
Sektion Mathematik der Friedrich-Schiller Universität, Jena, GDR

Lorna STEWART
Dept. of Computer Science, Univ. of Toronto, Toronto, Ontario, Canada

Received 10 May 1985
Revised 15 August 1986

This paper examines the class of bipartite permutation graphs. Two characterizations of graphs in this class are presented. These characterizations lead to a linear time recognition algorithm, and to polynomial time algorithms for a number of NP-complete problems when restricted to graphs in this class.

Bipartite graphs and permutation graphs are two well known subfamilies of the perfect graphs. Neither of these families is contained in the other, and their intersection is nonempty. This paper shows that graphs which are both bipartite and permutation graphs have good algorithmic properties. These graphs can be recognized in linear time, and several problems which are NP-complete or of unknown complexity for graphs in either of the two larger classes can be solved in polynomial time when restricted to bipartite permutation graphs. These problems include the Hamilton Circuit problem, a variant of the Crossing Number problem, and the minimum fill-in problem.

Definitions

For the most part, this paper uses standard graph theory terminology. Some of the notation and definitions will be explicitly stated in this section.

For a graph $G = (V, E)$, we will often denote $|V|$ by $n$ and $|E|$ by $m$. A linear time algorithm is an algorithm which takes $O(n + m)$ time.

The set of vertices adjacent to a vertex $u$ is called the neighborhood of $u$, and is written as $N(u)$. This does not include the vertex $u$ itself.

A graph is bipartite, or two-colorable, if its vertices can be partitioned into two independent sets. We will call the two independent sets $S$ and $T$. If there is more
than one way to partition the graph into two independent sets, the graph is not connected. For the problem in this paper, if $G$ is not connected, the problem is either trivial (no such graph can have a Hamilton Circuit, for example), or can be solved simply by combining the solutions from the different connected components. We will generally assume that our graph is connected. We will use the notation $G = (S, T, E)$ for bipartite graphs, where $S$ and $T$ are the independent sets.

A graph is a permutation graph if there is some pair $P, Q$ of permutations of the vertex set such that there is an edge between vertices $x$ and $y$ if and only if $x$ precedes $y$ in one of $\{P, Q\}$, while $y$ precedes $x$ in the other.

A permutation graph $G$ which is defined by the permutations $P, Q$ can also be described by a permutation diagram. Consider two columns, one consisting of the vertices in the order given by $P$, and the other consisting of the vertices in the order defined by $Q$. A line connects vertex $i$ in the left column with vertex $i$ in the right column. There is an edge between the vertices $x$ and $y$ if and only if the lines cross in the permutation diagram. Fig. 1 shows a permutation graph, and the corresponding permutation diagram. It should be noted that several different permutation diagrams may represent the same graph.

Other definitions are contained in the body of the paper. Notation and definitions not stated explicitly can be found in [15].

**Characterizations**

This section provides two characterizations of bipartite permutation graphs, which are used to develop the algorithms described in the rest of the paper. We need three specialized definitions to be able to describe these characterizations.

**Definition 1.** An ordering of the vertices $S$ in a bipartite graph $G = (S, T, E)$ has the adjacency property if for each vertex $x$ in $T$, $N(x)$ consists of vertices which are consecutive in the ordering of $S$. 

![Permutation Graph and Diagram](image)
Definition 2. An ordering of the vertices $S$ in a bipartite graph $G = (S, T, E)$ has the enclosure property if for every pair of vertices $u, w$ in $T$ such that $N(u)$ is a subset of $N(w)$, vertices in $N(w) - N(u)$ occur consecutively in the ordering of $S$.

Examples of orderings with and without these properties are given in Fig. 2.

Definition 3. A strong ordering of the vertices of a bipartite graph $G = (S, T, E)$ consists of an ordering of $S$ and an ordering of $T$ such that for all $(s, t), (s', t')$ in $E$, where $s, s'$ are in $S$ and $t, t'$ are in $T$, $s < s'$ and $t > t'$ imply $(s, t')$ and $(s', t)$ are in $E$.

To make the definition more clear, imagine the vertices of $S$ arranged on the north side of the unit square, with vertices of $T$ arranged on the south side. This is a strong ordering if whenever the edges $(s, t)$ and $(s', t')$ cross, then the four vertices must form a complete bipartite subgraph.

Theorem 1. The following statements are equivalent for a bipartite graph $G = (S, T, E)$:

(i) $G$ is a bipartite permutation graph.
(ii) There is a strong ordering of $S \cup T$.
(iii) There exists an ordering of $S$ which has the adjacency and enclosure properties.

Proof. (i) \Rightarrow (ii). Suppose $G = (S, T, E)$ is a bipartite permutation graph, and consider a permutation diagram $D$ for $G$.

Claim. Consider orderings of $S$ and $T$ in which vertices are ordered by their position in the first column of $D$. These constitute a strong ordering of $G$.

Consider any pair of edges $(s, t), (s', t')$ such that $s < s'$ and $t' < t$. We know that the lines corresponding to $s$ and $s'$ do not cross in $D$, so $s$ comes before $s'$ in both columns of $D$. Similarly, $t'$ comes before $t$ in both columns of $D$. There are three
possible positions for $t'$ in the first column of $D$; $t'$ may come before $s$, after $s'$, or between $s$ and $s'$. We look at each of these possible positions for $t'$.

Suppose that $t'$ precedes $s$ in the first column of $D$, as in Fig. 3(a). The line corresponding to $t'$ in $D$ must cross the line corresponding to $s'$ in $D$, so $t'$ must come after $s'$ in the second column of $D$. Since $t$ must come later than $t'$ in both columns, and the line corresponding to $t$ must cross the line corresponding to $s$, the vertices must be ordered $t's's'$ in the first column, and $ss't't$ in the second. This creates edges $(s', t)$ and $(s, t')$ in $G$. The argument is very similar if $t'$ comes after $s'$ in the first column, as in Fig. 3(b). The orders $ss't't$ and $t's's'$ are implied by the fact that the lines corresponding to $t$ and $s$ must cross in $D$, and the lines corresponding to $s'$ and $t'$ must cross in $D$. Finally, $t'$ cannot occur between $s$ and $s'$ in the first column of $D$. This would force $t'$ to come after $s'$ in the second column, and $t$ would come after $s$ in both columns, which contradicts the requirement that $s$ crosses $t$ in $D$.

(ii) ⇒ (iii). Let $G = (S, T, E)$ be a bipartite graph, and assume that $S$ and $T$ are strongly ordered. Suppose that $S$ does not have the adjacency property. Then there is some vertex $z$ in $T$ such that the vertices of $N(z)$ are not adjacent in $S$. Thus, there is a vertex $x$ in $S$ such that $(x, z)$ is not in $E$, but there exist vertices $w < x$ and $y > x$ with $(w, z)$ and $(y, z)$ in $E$. Now any edge incident with $x$ must cross one of $(w, z)$ and $(y, z)$, but because of the strong ordering this implies that $(x, z)$ is in $E$, a contradiction. Thus, $x$ has no incident edge. Any such $x$ can be moved to the beginning of $S$, resulting in an ordering of $S$ which has the adjacency property.

Now assume $S$ has the adjacency property and suppose $S$ does not have the enclosure property. There must be vertices $u, w$ in $T$ such that $N(u)$ is a subset of $N(w)$, but the vertices of $N(w) - N(u)$ do not appear consecutively in $S$. In other words, there are some vertices $a, b, c$ in $N(w)$ such that $a < b < c$ in $S$, $(a, u)$ and $(c, u)$ are not edges, but $(b, u)$ is an edge. The edge $(b, u)$ will cross either $(a, w)$ or $(c, w)$. Since this is a strong ordering, this means that either $(a, u)$ or $(c, u)$ is an edge. This is a contradiction. Therefore, $S$ must have the enclosure property. Thus, there exists an ordering of $S$ with the adjacency and enclosure properties.

(iii) ⇒ (i). Let $G = (S, T, E)$ be a bipartite graph and assume that $S$ satisfies the adjacency and enclosure properties. We will construct a pair of permutations $P, Q$ which represent $G$. 

![Fig. 3(a).](image_url)

![Fig. 3(b).](image_url)
For any $t$ in $T$ we let $\text{minadj}(t)$ be the first vertex $u$ in $S$ such that $(u, t)$ is an edge, and let $\text{maxadj}(t)$ be the last vertex $u$ of $S$ such that $(u, t)$ is an edge. Originally, set $P = S$, and $Q = S$. Vertices from $T$ are inserted into the permutations $P$ and $Q$ one at a time, using the following rules:

1. Each vertex $t$ is inserted into $P$ after $\text{maxadj}(t)$, and before any vertex from $S$ which appears after $\text{maxadj}(t)$ in $P$.
2. If $\text{maxadj}(t) = \text{maxadj}(t')$ for some $t, t'$, then $t$ is placed before $t'$ in $P$ if $\text{minadj}(t)$ appears before $\text{minadj}(t')$ in $S$.
3. Each vertex $t$ is inserted into $Q$ before $\text{minadj}(t)$, and after any vertex from $S$ which appears before $\text{minadj}(t)$ in $Q$.
4. If $\text{minadj}(t) = \text{minadj}(t')$ for some $t, t'$ in $T$, then $t$ is placed before $t'$ in $Q$ if and only if $t'$ appears after $t$ in $P$.

We show that the relationship between each pair of vertices in $G$ is represented correctly by $P$ and $Q$.

Suppose $b$ and $c$ are both in $S$. The vertices are nonadjacent, and the order of the vertices in $Q$ is the same as their order in $P$, so $P$ and $Q$ properly represent the relationship between $b$ and $c$.

Suppose $b$ is in $S$, $y$ is in $T$. If $b$ is adjacent to $y$, then $b$ precedes $y$ in $P$, since $\text{maxadj}(y)$ is the last vertex in $P$ which is adjacent to $y$, and $y$ is inserted into $P$ after $\text{maxadj}(y)$. Similarly, $y$ appears before $b$ in $Q$, since $y$ is inserted before $\text{minadj}(y)$ in $Q$. If $b$ and $y$ are nonadjacent, by the adjacency property, all vertices adjacent to $y$ occur consecutively in $S$. Therefore, $b$ either precedes all the vertices of $N(y)$ in both $P$ and $Q$, or $b$ succeeds all vertices of $N(y)$ in $P$ and $Q$. In the first case, $y$ is inserted after $b$ in both of the permutations, while in the latter case, $y$ is inserted before $b$ in $P$ and $Q$. Thus, $P$ and $Q$ will properly represent the relationship of $b$ and $y$.

Let $x$ and $y$ be vertices of $T$. The vertices must be nonadjacent. Suppose that there are vertices $b$ and $c$ such that $x$ is in $N(b) - N(c)$, and $y$ is in $N(c) - N(b)$. Without loss of generality, assume that $b$ comes before $c$ in $P$ and $Q$. By the adjacency property, $b$ will come before every vertex of $N(y)$ in both $P$ and $Q$, while $c$ comes after every vertex of $N(x)$. Therefore, $x$ will precede $y$ in both permutations.

If $N(x)$ is a proper subset of $N(y)$, the enclosure property tells us that vertices in $N(x) - N(y)$ occur consecutively. This means that either $\text{minadj}(x) = \text{minadj}(y)$, or $\text{maxadj}(x) = \text{maxadj}(y)$. If $\text{minadj}(x) = \text{minadj}(y)$, then $x$ precedes $y$ in $P$ and $Q$. If $\text{maxadj}(x) = \text{maxadj}(y)$, then $y$ precedes $x$ in $P$ and $Q$.

Finally, if $N(x) = N(y)$, the fourth rule of the placement procedure guarantees that the vertex which appears first in $P$ will also appear first in $Q$.

Therefore, if $S$ has the adjacency and enclosure properties, then $G$ is a permutation graph. □
Recognition algorithm

The fastest known methods for recognizing general permutation graphs require $\Omega(n^2)$ time, where $n$ is the number of vertices in the graph [26,27]. These techniques use a complicated algorithm for finding the transitive orientation of a graph. Other recognition algorithms, such as [9,13,14], require $\Omega(n^3)$ time. In this section, we develop a simple algorithm which recognizes bipartite permutation graphs in linear time.

There are simple linear time algorithms to divide a graph into connected components, determine whether a graph is bipartite, and divide a bipartite graph into its two color classes. It is easy to see that a graph is a bipartite permutation graph if and only if every connected component is a bipartite permutation graph, so we can restrict our attention to connected graphs for the rest of this section.

The general idea of the algorithm is to maintain a data structure which allows us to represent all possible permutations of $S$ which have the adjacency and enclosure properties. Vertices of $T$ are examined one at a time, each one forcing more restrictions on the permutation.

We need some new terminology in order to describe the algorithm.

A block $B$ is a set of vertices from $S$ which must occur consecutively in any ordering of $S$ which has the adjacency and enclosure properties, but vertices in $B$ may occur in any order.

A block list is a sequence of blocks $B_1, B_2, \ldots, B_k$. Vertices within each block will occur consecutively in our permutation of $S$, and the last vertex of $B_i$ will immediately precede the first vertex of $B(i+1)$, $1 \leq i \leq k-1$.

The methodology of the algorithm is similar to the algorithm of Booth and Lueker for the consecutive arrangement problem [4], where enclosure adds another type of constraint. For readers familiar with that paper, a block is like the children of a $P$-node in [4], and a block list is like the children of a $Q$-node.

As each vertex from $T$ is examined, we have a current block list, which represents a minimum set of restrictions on any ordering of $S$ which has the adjacency and enclosure properties. This current block list is called $BLIST$. Vertices from $S$ are called new if they are not in any block of $BLIST$, and are old if they are already in some block of $BLIST$. The leftmost block of $BLIST$ will be called $B_L$, and the rightmost block of $BLIST$ will be called $B_R$.

For each vertex $t$ in $T$, let $ADJ(t)$ be the set of blocks in $BLIST$ which contain at least one vertex from $N(t)$. Let $MIX(t)$ be the set of blocks which contain at least one vertex from $N(t)$, and at least one vertex from $S - N(t)$.

At the beginning of the algorithm, we find a vertex $y$ in $T$ such that $|N(y)|$ is minimal. We create a block containing the members of $N(y)$, and let our initial $BLIST$ be this block.

At each remaining step of the algorithm, we consider some vertex $t$ from $T$ which is adjacent to at least one old vertex. We modify $BLIST$ so that the vertices adjacent to $t$ will occur consecutively, and will not ‘enclose’ the neighbors of any other vertex.
or be 'enclosed' by the neighbors of any other vertex, if this is possible. We divide the description of the algorithm into four cases, depending on the composition of \( N(t) \).

**Case 1:** \( N(t) \) contains a new vertex.

**Case 1.1:** Some old vertex is not in \( N(t) \).

**Case 1.2:** \( N(t) \) contains every old vertex.

**Case 2:** All vertices in \( N(t) \) are old.

**Case 2.1:** \( N(t) \) contains vertices from two distinct blocks of BLIST.

**Case 2.2:** \( N(t) \) is completely contained in a single block of BLIST.

**Case 1:** \( N(t) \) is a new vertex.

For both cases 1.1 and 1.2, create a block \( B_{new} \) consisting of all new vertices in \( N(t) \).

**Case 1.1:** Some old vertex is not in \( N(t) \).

We first decide where the new vertices belong, and then modify the old vertices to reflect the constraints imposed by \( t \).

If BLIST has only one block, we arbitrarily decide to place \( B_{new} \) at the end of BLIST. If BLIST has more than one block, it is easy to determine which side of BLIST \( B_{new} \) must be added to. The blocks of BLIST must occur consecutively, and the vertices of \( N(t) \) must occur consecutively. Thus, \( B_{new} \) belongs on either the right or left end of BLIST. If \( B_L \) is not in \( \text{ADJ}(t) \), or if \( B_L \) is in \( \text{MIX}(t) \) and \( B_R \) is in \( \text{ADJ}(t) \), then \( B_{new} \) is placed at the end of BLIST; otherwise, \( B_{new} \) is placed at the beginning of BLIST. This placement is forced by the adjacency property for neighbors of \( t \).

After \( B_{new} \) has been placed, we must modify the old blocks of BLIST so that vertices of \( N(t) \) are forced to occur consecutively. If there is any block \( B \) in \( \text{MIX}(t) \), remove the vertices of \( B \) which are in \( N(t) \) from \( B \), creating a new block \( C \). \( B \) cannot come between \( C \) and \( B_{new} \) if the adjacency property is satisfied, so \( C \) is placed on the side of \( B \) which is closest to \( B_{new} \).

**Case 1.2:** Every old vertex is adjacent to \( t \).

In this case, the algorithm simply places \( B_{new} \) at the right end of BLIST.

If there is only one block in BLIST at this time, this choice is completely arbitrary. If there is more than one block in BLIST, the choice is forced so that \( N(t) \) does not enclose \( N(y) \), where \( y \) was the first vertex we used to create the initial block. The algorithm always places at least one vertex \( b \) which is not in \( N(y) \) at the right of BLIST when the second block of BLIST is created. Since \( N(t) - N(y) \) includes both \( b \) and \( B_{new} \), \( B_{new} \) must also be placed to the right of BLIST.

**Case 2:** All vertices in \( N(t) \) are old.

**Case 2.1:** \( N(t) \) contains vertices from at least two blocks of BLIST.
We need to guarantee that vertices of $N(t)$ occur consecutively. For any block $B$ which is in MIX($t$), remove the vertices of $N(t)$ from $B$ to form a new block $C$. Place $C$ between $B$ and the block which is next to $B$ in BLIST which is also in ADJ($t$). If $B$ has two neighboring blocks in ADJ($t$), it is easy to see that there is no possible arrangement which has the adjacency property, so that this is the only possible way to refine the block list.

*Case 2.2: $N(t)$ is completely contained in a single block $B$ of BLIST.*

If all vertices in $B$ are in $N(t)$, do nothing. Otherwise, remove the vertices of $N(t)$ from $B$, forming a new block $C$. If $B = B_L$, then put $C$ at the end of BLIST. If $B = B_R$, place $C$ at the front of BLIST. If $B$ is not $B_L$ or $B_R$, then the graph is not a permutation graph; this will be explained in the following paragraph.

We first note that if $B$ was split, there must be more than one block in BLIST, since the initial block corresponded to $N(y)$ and $|N(y)|$ was minimal. If $C$ is not placed on the outside of BLIST, then $N(t)$ will be enclosed by the neighbors of some other vertex. It is easy to see that any pair of blocks which are next to each other in BLIST are contained in $N(x)$ for some $x$ in $T$; otherwise, they would not have been placed together. If $C$ is placed between $B$ and $D$, where $D$ is next to $B$ in BLIST, then every vertex in both $B$ and $D$ is in $N(x)$ for some $x$ in $T$, and vertices in $N(x) - N(t)$ would not occur consecutively.

For each vertex $t$, the rearrangement caused by that vertex can be done in time proportional to the neighbors of $T$, as long as each block is stored as a doubly linked list.

All these refinements of BLIST are forced if the ordering of $S$ has the adjacency and enclosure properties, with the exception of the placement of the second block to the right of the first block. If two vertices are in the same block at the end of this algorithm, they are adjacent to the same set of vertices, and are interchangeable in any ordering of $S$. Therefore, we divide any block containing more than one vertex into subblocks containing a single vertex, arranged in any order.

If the graph is a permutation graph, this ordering will have the adjacency and enclosure properties. In fact, up to renaming of vertices with the same neighbors, it is one of two orderings which have the adjacency and enclosure properties; the other is simply the reverse, and would have been the ordering constructed if we choose the second block to precede the first using the algorithm described above. On the other hand, if the graph is not a permutation graph, this algorithm may still produce a candidate ordering. Therefore, we need a way to verify whether this ordering has the adjacency and enclosure properties. We will construct the two permutations of $\{SU T\}$ which represent the graph if the adjacency and enclosure property hold, and test to see whether they represent the graph properly.

We construct the permutations $P$ and $Q$ from the ordering of $S$ using the same method described in Theorem 1, when we proved that (iii) $\Rightarrow$ (i). Initially, $P$ and $Q$ will be the ordering of $S$.

We first insert all vertices of $T$ into $P$. As in Theorem 1, each vertex $t$ will be in-
serted after maxadj(t) in P, and before the next vertex of S in P. If two vertices t, t' of T are inserted between the same pair of vertices of S, we want to let t precede t' whenever minadj(t) precedes minadj(t') in P. We first sort the vertices of T into a list ordered by decreasing position of minadj(t) in P; this can be accomplished in O(n + m) time by using a bucket sort. We then go through the sorted list in order, inserting each vertex t immediately after maxadj(t) in P.

We then insert all vertices of T into Q. Each vertex t should be inserted into Q before minadj(t), and after any previous vertices of S in Q. If two vertices t, t' of T are inserted between the same pair of vertices of S, t should precede t' in Q if and only if t precedes t' in P. These insertions can be performed in O(n f m) time by creating a list of vertices of T ordered by their position in P, and inserting each vertex t immediately before minadj(t).

The two permutations need to be checked to see whether they represent the relations between all vertex pairs correctly. Let u be the first vertex of P. If N(u) is not equal to the set of vertices which appear before u in Q, the graph is not represented properly. This can be checked in O(|N(u)|) time by marking all vertices in N(u), stepping through Q to check that all vertices before u are marked, and checking that |N(u)| = the number of vertices which occur before u in Q. Vertex u is then deleted from the lists P and Q. We then repeat this process for the new vertex at the front of P; the only difference is that we must not include the vertices which have already been used in our count of |N(u)}. If P is exhausted, then the graph is a permutation graph. Since we do work proportional to the size of the adjacency list of u at each step, this is an O(n + m) algorithm.

Other algorithms

In this section, we use the characterizations developed earlier to show that a number of problems which are NP-complete on arbitrary graphs can be solved in polynomial time for bipartite permutation graphs. These problems include the Hamilton circuit and Hamilton path problems, a variant of the crossing number problem, and the minimum fill-in problem. Other results of this type appear in [28,29], where it is shown that the jump number problem can be solved in polynomial time for bipartite permutation graphs, and [26], where a number of vertex deletion problems are solved in polynomial time for graphs in this class. Results on general permutation graphs are contained in [15, 2, 3, 10, 29, 20, 8].

We begin with a variant of the crossing number problem, as described by Johnson [18]. Given a bipartite graph G = (S, T, E) and a number k, can G be embedded in a unit square so that all vertices from S are on the North boundary, all vertices from T are on the South boundary, and the number of crossings is k or less? This problem is NP-complete for arbitrary bipartite graphs; we will show that it can be solved in polynomial time for permutation graphs.

Arrange the vertices of S and T as in the strong ordering of G. Each crossing of
(b, x) and (c, y) is contained in a cycle b, x, c, y, b, and is the only crossing of edges in this cycle. Therefore, the number of crossings is at most the number of 4-cycles in G. However, there must be at least one crossing for any cycle of length four in G, and no distinct pair of 4-cycles in a bipartite graph can share the same crossing pair, so the number of 4-cycles is also a lower bound on the number of crossings in G.

Therefore, this variant of the crossing number problem can be solved in polynomial time for permutation graphs by constructing the strong ordering, and counting the number of crossings.

The next problems we examine will be the Hamilton circuit and Hamilton path problems. Formal definitions of these problems are given in [12]. A Hamilton circuit in a graph is a simple cycle which contains every vertex in G, and a Hamilton path is a simple path which contains every vertex in G. These problems remain NP-complete when restricted to several special classes of bipartite graphs [1, 22, 17, 23], and have been mentioned as open problems with respect to general permutation graphs [20].

We will say that a Hamilton path in a bipartite permutation graph begins at a vertex v if v has the earlier index of the two endpoints in the strong ordering of S U T. Let S_i = {s_1, s_2, ..., s_i}, and let T_i = {t_1, t_2, ..., t_i}.

**Claim.** Let G = (S, T, E) be a bipartite permutation graph which contains a Hamilton path H beginning at vertex s in S. Then s_1, t_1, s_2, t_2, ..., s_k, t_k (followed by s_k+1 if it exists) is also a Hamilton path H' in G, where S U T is ordered as in a strong ordering of G.

**Proof.** Consider the first edge of H' which does not occur in G. Suppose that the missing edge is of the form (s_i, t_j). Clearly, i cannot be equal to 1, since any edge out of s_1 must cross any edge out of t_1. Some edge of H must leave S_i to a vertex t_j, where j > i - 1. This must be true since at least 2i - 2 edges leave S_i in H, and at most 2i - 2 edges leave T_{i-1}; there would be no edges to connect S_i U T_{i-1} to the rest of the graph otherwise. Any such edge which is not (s_i, t_j) itself must cross (t_{i-1}, s_i), so (s_i, t_j) is also an edge of G. Any edge out of t_i crossed either (t_{i-1}, s_i) or (s_{i-1}, t_j), so (s_i, t_i) is an edge of G for every value of i.

Suppose that the first edge of H' which does not occur in G is (t_j, s_{j+1}). Some edge of H must go from T_j to a vertex s_j, j > i. This must be true since the number of edges of H leaving T_j is at least as large as the number of edges of H leaving S_i (the path begins in S), and T_j U S_j must be connected to the rest of the graph. If the edge to s_j is not from t_i, it must cross (s_i, t_j), so (t_j, s_j) is an edge of G. Any edge other than s_{i+1} must cross either (t_j, s_i) or (t_i, s_j), so (t_j, s_{j+1}) is also an edge of G.

Therefore, H' is also a Hamilton path in G. □

It is now simple to determine whether a bipartite permutation graph has a Hamilton path. There are only two paths to check for, depending on whether the
path starts in $S$ or $T$. If $G$ has a Hamilton path, then either $s_1, t_1, s_2, t_2, \ldots, t_{k-1}, s_k, (t_k)$
or $t_1, s_1, t_2, s_2, \ldots, s_{k-1}, t_k, (s_k)$ is a Hamilton path.

**Claim.** A bipartite permutation graph $G = (S, T, E)$ with $|S| = |T| = k \geq 2$ has a
Hamilton circuit if and only if $s_i, t_i, s_{i+1}, t_{i+1}$ is a cycle of length four for $1 \leq i \leq k - 1$

**Proof.** If all the cycles of length four exist, the Hamilton circuit consists of the
edges $(s_1, t_1), (s_k, t_k), (s_{i}, t_{i+1})$ for $1 \leq i \leq k - 1$, and $(s_i, t_{i-1})$ for $2 \leq i \leq k$. If $G$ has a
Hamilton circuit, then $G$ has a Hamilton path which begins in $S$, and another
Hamilton path which begins in $T$. By the previous claim, this forces each edge
$(s_i, t_i), (s_{i}, t_{i+1})$ and $(s_{i+1}, t_{i})$ to occur in $G$. $\square$

This gives us linear time algorithms to determine whether a graph is a bipartite
permutation graph with a Hamilton path/circuit. In fact, if we are actually given
the permutation rather than the graph, the algorithms run in $O(n)$ time, which may
be less than linear. This time bound relies on the fact that we can divide a permuta-
tion which corresponds to a bipartite permutation graph into its two color classes
in $O(n)$ time, using the canonical coloring algorithm for permutations described in
[9, 15].

For the remainder of this section, we will deal with the minimum fill-in problem,
which is also known as the chordal graph completion problem.

A graph is **chordal** if every cycle of length greater than three has a chord. For an
arbitrary graph $G = (V, E)$, a minimum fill-in for $G$ refers to a minimum cardinality
set $E'$ from $(V \times V)$ such that $G' = (V, E \cup E')$ is chordal.

The minimum fill-in problem has applications to the solution of sparse systems
of linear equations by Gaussian elimination. We refer the interested reader to Rose
[25] for this aspect of the problem. For general graphs, the problem of computing
the minimum fill-in is NP-complete [30]. The complexity of the problem is unknown
for bipartite graphs, and for permutation graphs. We will show that it can be solved
in polynomial time for bipartite permutation graphs.

For a bipartite graph $G = (S, T, E)$, a set $E'$ from $(V \times V)$ is said to be $C_k$-
destroying if for every cycle of length four $s, t, s', t'$ in $G$, the edge $(s, s')$ or
$(t, t')$ is in $E'$. Any edge of a minimum $C_k$-destroying set $E'$ has both endpoints
in $S$ or both endpoints in $T$, so we can partition $E'$ into $E'_S = \{(s, s') \in E' | s, s' \in S\}, E'_T = \{(t, t') \in E' | t, t' \in T\}$.

We will prove that the minimum fill-in set for bipartite permutation graphs is
equal to the smallest $C_k$-destroying set. This problem is then reduced to a vertex
cover problem on bipartite graphs, which can be solved in polynomial time. Potential
edges of $E'$ are the vertices of the graph in the vertex cover problem, and the
edges are the cycles of length four in $G$; an edge corresponding to $(s, t, s', t')$ can be
covered by the two potential edges of $E'$ which can break the cycle, that is, $(s, s')$ and $(t, t')$. 

In the rest of this section, we assume that $G = (S, T, E)$ is a strongly ordered bipartite permutation graph, and that $E'$ is a minimum $C_4$-destroying set.

**Lemma 1.** If $(u, w)$ is in $E'_S (E'_T)$, then for all $v$ in $S (T)$ such that $u < v < w$, the edges $(u, v)$ and $(v, w)$ are also in $E'_S (E'_T)$.

**Proof.** Suppose that this is not the case. Without loss of generality, let $u, v, w$ be in $S$, $(u, w)$ be in $E'$, and let $(u, v)$ not be in $E'$. Since $(u, w)$ is in $E'$ and $E'$ is a minimum $C_4$-destroying set, there must be $x, y$ in $T$ such that $u, x, w, y, u$ is a cycle in $G$, and $(x, y)$ is not in $E'$. By the adjacency property of $S$, $v$ must also be adjacent to $x$ and $y$, so $u, x, u, v, y, u$ is a cycle in $G$. Neither $(x, v)$ nor $(u, v)$ is in $E'$, which contradicts the fact that $E'$ is a $C_4$-destroying set. □

**Theorem.** Let $G = (S, T, E)$ be a strongly ordered bipartite permutation graph, and let $E'$ be a minimum cardinality $C_4$-destroying set. Then the graph $G' = (S \cup T, E \cup E')$ is a chordal graph.

**Proof.** Suppose that $G'$ is not a chordal graph; let $C$ be a chordless cycle in $G'$. $G$ cannot contain a chordless cycle of length $> 4$, some pair of edges in the cycle must cross in the strong ordering, which will create chords in the cycle. In fact, no permutation graph can contain an induced cycle of length $> 4$ [2, 29]. Therefore, since $E'$ destroys all cycles of length four in $G$, $C$ contains at least one edge from $E'$. By Lemma 1, $C$ must contain at least one edge from $E$.

Assume without loss of generality that some edge of $C$ is in $E'_S$. Let $c$ be the last vertex of $S$ in the strong ordering which is an endpoint of an edge in $C$ from $E'_S$; this edge is of the form $(b, c)$, where $b$ is in $S$. The other edge out of $c$ in $C$ must be of the form $(c, x)$, where $x$ is in $T$. Note that no vertex of $C$ can come between $b$ and $c$ in $S$; by Lemma 1, this would create a third edge to vertex $c$.

Since $(b, c)$ is in $E'$, there must be some cycle $c, u, b, u, c$ in $G$, where $(u, v)$ is not in $E'$, and $u$ precedes $v$ in the strong ordering of $T$. Vertex $x$ must come after $v$ in the strong ordering of $T$; otherwise, $(x, c)$ would cross $(u, b)$, which would create a chord $(x, b)$ in $C$. Some edge of $C$ must 'return' from the set of vertices which come after $c$ in $S$ or after $v$ in $T$ to the rest of the graph. See Fig. 4.

This edge cannot be of the form $(d, t)$, where $d$ comes after $c$ in $S$ and $t$ comes before $v$ in $T$, since this would cross $(c, v)$ and create a chord $(t, c)$. The returning edge cannot be of the form $(a, w)$, where $a$ comes before $c$ in $S$ and $w$ comes after $v$ in $T$. This edge would cross $(c, v)$, creating an edge $(w, c)$, which is possible only

![Fig. 4.](image-url)
if \( w = x \). However, since \( a \) cannot come between \( b \) and \( c \), the edge also crosses \((b, u)\), and there would be a chord \((x, b)\). The returning edge cannot be in \( E'_c \), since \( c \) is the last vertex of \( S \) which is connected to an edge of this form in \( C \). Therefore, the returning edge of \( C \) must be of the form \((t, z)\), where \( t \) and \( z \) are in \( T \), and \( t < v < z \). Vertex \( t \) cannot come between \( u \) and \( v \) in \( T \); any edge of \( G \) from such a vertex would cross \((c, u)\) or \((c, v)\), and there would be a chord \((c, t)\) in \( C \). However, if \( t \) comes before \( u \) and \( z \) comes after \( v \), Lemma 1 tells us that \((u, v)\) is also in \( E' \), contradicting an earlier assumption.

Therefore, \( G' \) must be chordal. \( \square \)

Since the union of \( G \) and any minimum \( C_d \)-destroying set is chordal, and any fill-in set must be a \( C_d \)-destroying set, the minimum cardinality \( C_d \)-destroying set in a bipartite permutation graph is also a minimum fill-in.

Given a bipartite graph \( G = (S, T, E) \), we can represent the \( C_d \)-structure of \( G \) with another graph \( H(G) \), which we define as follows.

\[
H(G) = (V_H, E_H), \text{ where } V_H = S_H \cup T_H,
\]

\[
S_H = \{ss' \mid s, s' \in S, z < s'\},
\]

\[
T_H = \{tt' \mid t, t' \in T, t < t'\},
\]

\[
E_H = \{(ss', tt') \mid ss' \in S_H, tt' \in T_H \text{ and } (s, t), (s', t), (s', t'), (s, t') \text{ are in } E\}.
\]

Note that \( H(G) \) is a bipartite graph. Vertices of \( H(G) \) correspond to possible edges of a \( C_d \)-destroying set, and edges correspond to cycles of length four in \( G \). Every edge of \( H(G) \) has as endpoints vertices corresponding to the two possible edges which can destroy the corresponding cycle in \( G \). Thus, a set of edges is a minimum cardinality \( C_d \)-destroying set in \( G \) if and only if the corresponding set of vertices in \( V_H \) is a minimum vertex cover of \( H(G) \). The minimum vertex cover of a bipartite graph can be constructed in \( O(n + m) \) time from the maximum matching of the graph [27,5], and a maximum matching in a bipartite graph can be found in \( O(mn^{0.5}) \) time [16]. Since \( H(G) \) has \( O(n^2) \) vertices and \( O(n^4) \) edges, a minimum fill-in can be calculated in \( O(n^5) \) time.

The time bound of the algorithm described here, though polynomial, is still quite large, and the question of whether a more efficient algorithm exists is as yet unanswered.

Conclusions

This paper presents several characterizations of bipartite permutation graphs, and uses them to develop algorithms for a number of problems on this class of graphs. The main open questions are to determine whether similar algorithms can be found for general permutation graphs. In particular, the complexity of the Hamilton circuit and minimum fill-in problem are unknown for permutation graphs, and no linear time recognition algorithm has been developed.
References