ALGORITHMS FOR FINDING A MAXIMUM BIPARTITE SUBGRAPH FOR SPECIAL CLASSES OF GRAPHS

Susan S. Yeh
Andrea S. LaPaugh

CS-TR-149-88

April 1988

Algorithms for Finding a Maximum Bipartite Subgraph for Special Classes of Graphs

Susan S. Yeh
Andrea S. LaPaugh

Department of Computer Science
Princeton University
Princeton, NJ 08544

ABSTRACT

In this paper we present efficient polynomial algorithms for finding a maximum bipartite subgraph for special classes of graphs. In general this problem is NP-complete, and it remains NP-complete even when the graph is restricted to be planar. However, putting further restrictions on the graph makes this problem tractable. We have developed algorithms for finding a maximum bipartite subgraph for proper circular-arc, circular-arc, permutation, and split graphs. In numerous occasions, we used one technique, namely dynamic programming, for the development of the algorithms. Furthermore, the algorithms we designed, with slight modifications, can be adapted to solve the maximum bipartite subgraph problem which includes a nontrivial subset of vertices.

Algorithms for Finding a Maximum Bipartite Subgraph for Special Classes of Graphs

Susan S. Yeh
Andrea S. LaPaugh

Department of Computer Science
Princeton University
Princeton, NJ 08544

I. Introduction

In this paper we are interested in finding a *maximum induced bipartite subgraph* of a graph, that is, 2 independent sets that cover maximum number of vertices. (For the rest of the paper, when we say subgraph we mean *induced* subgraph.) In general this problem is NP-complete as shown by Lewis and Yannakakis[1980]. Furthermore, even when restricted to planar graphs, the maximum bipartite subgraph problem remains NP-complete [Lewis and Yannakakis, 1980]. This makes one wonder if additional restrictions on graphs affect the complexity of this problem. In particular, it is interesting to identify special classes of graphs for which finding a maximum bipartite subgraph is solvable in polynomial time. The focus of this paper is in showing efficient polynomial algorithms for this problem on proper circular-arc, circular-arc, permutation, and split graphs. More specifically, we present $O(n\delta^2)$ algorithm for finding a maximum bipartite subgraph for circular-arc graphs, when the graphs are given in the form of a family of intervals, where δ is the minimum number of arcs covering any point on the circle. For proper circular-arc graphs, we give an $O(n^2)$ algorithm that solves the same problem. Using dynamic programming, we show how to find a maximum bipartite subgraph for permutation graphs in time $O(n\overline{e}_t)$, where $\overline{e}_t$ is the number of edges in the complement of a permutation graph after transitive reduction. For split graphs, we use matrix multiplication and obtain an $O(n^{2.376})$ algorithm for the maximum bipartite subgraph problem.

In this paper we will also consider finding a maximum bipartite subgraph which includes a nontrivial subset (size > 0) of vertices. We shall call this the *maximum bipartite subgraph with a chosen subset* problem, with the understanding that the size of the chosen subset is greater than 0. If we allow the size of the chosen subset to be 0, then the maximum bipartite subgraph problem is a special case of the maximum bipartite

subgraph with a chosen subset problem. This means the second problem is as hard as the first, if not harder. However, if we only consider the cases where the size of the chosen subset is greater than 0, then, as we will show, in most cases the complexity of the algorithms for finding a maximum bipartite subgraph remain the same, and in one case – circular-arc graphs, the complexity of the algorithm is reduced from $O(n\delta^2)$ to $O(n\log n + n\Delta)$, where Δ is the maximum number of arcs covering any point on the circle.

Finding a maximum bipartite subgraph falls in the general frame work of finding a maximum k-colorable subgraph of a graph. Two well-known problems: maximum independent set and the chromatic number, are also special cases of this problem. Interestingly, the maximum independent set problem, though shown to be NP-complete for general graphs and planar graphs, is polynomial solvable for a wide range of other well-known classes of graphs, including perfect (which properly contain chordal, comparability, permutation, and interval), circle, circular-arc, series-parallel, and k-outerplanar graphs; see Johnson[1985] for references. The chromatic number problem, however, is NP-complete for a larger number of classes of graphs, including planar, circular-arc, and circle graphs, but is polynomial for perfect, series-parallel, and k-outerplanar graphs; see Johnson[1985] for references. The existence of polynomial algorithms for the maximum independent set and the chromatic number problems on various classes of graphs may lead one to believe the maximum bipartite subgraph problem is solvable in polynomial time for these classes of graphs as well. In fact, the complexity of the maximum k-colorable subgraph problem on special classes of graphs has been investigated by a number of researchers, including Frank[1980] who has presented a polynomial algorithm for this problem on comparability graphs and their complements, and Yannakakis and Gavril[1987] who have shown for chordal graphs this problem is polynomially solvable when k is fixed, but NP-hard when k is not fixed. One implication of these results is that the maximum bipartite subgraph problem is polynomial solvable for comparability and chordal graphs.

Table 1 summarizes the current status of polynomial algorithms and NP-complete results for the maximum bipartite subgraph problem and the maximum bipartite subgraph with a chosen subset problem on various classes of graphs. We have analyzed the running time of algorithms presented by Frank for comparability graphs and by Yannakakis and Gavril for chordal graphs [Yannakakis, 1987], and included our findings in the table.

Graph Class	Running Time		Discoverer	Transformation Time†
	Max. Bipartite	with Subset		
1. Interval	$O(n\log n)$	$O(n\log n)$	Yannakakis and Gavril[1987]	$O(n + e)$
2. Proper Circular-Arc	$O(n^2)$	$O(n\log n + n\Delta)$‡	Yeh and LaPaugh	$O(n^2)$
3. Circular-Arc	$O(n\delta^2)$‡‡	$O(n\log n + n\Delta)$	Yeh and LaPaugh*	$O(n^3)$
4. Permutation	$O(n\bar{e_t})$‡‡‡	$O(n\bar{e_t})$	Yeh and LaPaugh**	$O(de + n^2)$‡‡‡‡
5. Comparability	$O(ne)$	open	Frank[1980]	$O(de)$
6. Split	$O(n^{2.376})$	$O(n^{2.376})$	Yeh and LaPaugh	NA
7. Chordal	$O(ne)$	$O(ne)$	Yannakakis and Gavril[1987]	$O(n + e)$
8. Series-Parallel	$O(n + e)$††	open	Takamizawa et al[1982]	NA
9. Planar	NP-complete	NP-complete	Lewis and Yannakakis[1980]	NA
10. Circle	NP-complete	NP-complete	Sarrafzadeh and Lee[1987]	NA
11. General	NP-complete	NP-complete	Lewis and Yannakakis[1980]	NA

† Transformation time corresponds to the time needed to transform a graph G = (V, E) to the proper representation required by the algorithm.

†† Takamizawa, Nishizeki, and Saito[1982] have claimed this result in their paper but a proof was not given.

* An independent study of this problem on circular-arc graphs has been done by Manber and Narasimhan[1987] where they obtained an $O(n^3)$ time algorithm.

** Recently Sarrafzadeh and Lee[1987] have developed an $O(n^2\log n)$ algorithm for this problem on permutation graphs. Their algorithm will be the preferred algorithm when the complement of the permutation graph after transitive reduction is dense.

‡ Δ is the maximum number of arcs covering any point on the circle, $\Delta \leq d \leq n$.

‡‡ δ is the minimum number of arcs covering any point on the circle, $\delta \leq \sqrt{e}$.

‡‡‡ $\bar{e_t}$ is the number of edges in the complement of a permutation graph after transitive reduction.

‡‡‡‡ d is the maximum degree of a vertex.

Table 1. Polynomial algorithms and NP-complete results for the maximum bipartite subgraph problem and the maximum bipartite subgraph with a chosen subset problem. Algorithms for graphs 1, 2, 3, 4, 6 are presented in this paper.

The remaining portion of this paper is devoted to presentation of the algorithms. We start with some preliminaries in section II. In section III we describe a greedy algorithm for interval graphs; in section IV we introduce the notion of placing intervals on tracks and present algorithms for track assignment problems. The discussions of sections III and IV will be used to develop the algorithms for proper circular-arc and circular-arc

graphs. We present algorithms for proper-circular-arc, circular-arc, permutation, and split graphs in sections V, VI, VII, and VIII, respectively. In section IX we close with some concluding remarks and suggestions for future research.

II. Preliminaries

In this paper G = (V, E) will denote a simple, finite, and undirected graph with n vertices and e edges. When we consider a subgraph G_A (A $\subseteq$ V) of G, we mean the *induced subgraph* of G, that is, G_A consists of the set of vertices A, and two vertices are adjacent in G_A if and only if they are adjacent in G. A graph G = (V, E) is called an *intersection* graph for a family F of sets if each set in F is represented by a vertex and two vertices are adjacent if and only if their corresponding sets intersect. The intersection graph of a family of intervals on a real line is called an *interval* graph. The intersection graph of a family of arcs on a circle is called a *circular-arc* graph. A *proper circular-arc* graph is a circular-arc graph such that no arc is contained within another. The intersection representation for interval, proper circular-arc, and circular-arc graphs can be constructed in time $O(n+e)$ [Booth and Lueker, 1976], $O(n^2)$ [Tucker, 1971], and $O(n^3)$ [Tucker, 1980], respectively.

A graph G = (V, E) is called *perfect* if for every subgraph G_A of G (A $\subseteq$ V), the chromatic number of G_A equals its maximum clique size. Perfect graphs properly contain comparability, permutation, chordal, split, and interval graphs. *Comparability* graphs, also known as *transitively orientable* and *partially orderable* graphs, are undirected graphs G = (V, E) that can, by appropriate orientation of their edges, be turned into transitive directed graphs (directed graphs such that if (a,b) and (b,c) are arcs, then so is (a,c)). *Permutation* graphs are comparability graphs whose complements are also comparability graphs (an alternate characterization of permutation graphs is given in section VII). An undirected graph G is called *chordal* if every cycle of length greater than 3 has a chord, i.e., an edge joining two nonconsecutive vertices on the cycle. Chordal graphs are also known as *triangulated, perfect elimination, rigid-circuit,* and *monotone transitive* graphs. *Split* graphs are chordal graphs whose complements are also chordal graphs (an alternate characterization of split graphs is given in section VIII). Comparability and chordal graphs are incomparable, i.e., neither is properly contained within another.

III. Interval Graphs

Assume an interval graph I is given as a set of intervals, $I=\{f_1, f_2, \cdots, f_n\}$, on the real line. Each interval is in the form of (left_endpoint, right_endpoint), and is represented by a vertex. Without loss of generality we can assume the 2n endpoints of the intervals are distinct.† The algorithm for finding a maximum bipartite subgraph for interval graphs goes as follows:

Input. An interval graph I given as a set of intervals, $I=\{f_1, f_2, \cdots, f_n\}$.

Output. A maximum bipartite subgraph of I.

1. Sort the 2n endpoints in ascending order.
2. Number the n intervals (vertices) according to their right_endpoints in ascending order. Let β, β = {1, 2, ..., n}, be the collection of n intervals.
3. While β is nonempty do
    4. Pick an interval with the smallest right_endpoint from β and add it to a set called S (originally empty).
    5. Delete all intervals from β which overlap two intervals already in S.

    end while
6. Return S.

Algorithm 1. Maximum bipartite subgraph algorithm for interval graphs.

This is a greedy algorithm; at each step we are picking an interval that is as good as, if not better than, any other interval that overlaps it. We will need the following definition for the proof of the correctness of Algorithm 1:

Let $G_a=(V_a, E_a) \subseteq$ G and $G_b=(V_b, E_b) \subseteq$ G. G_a *disagrees* with G_b at vertex v_a if v_a is the smallest numbered vertex in V_a but not in V_b. Similarly if an interval graph I is given in the form of a family of intervals, and $I_a \subseteq I$, $I_b \subseteq I$, then we say I_a *disagrees* with I_b at interval f_a if f_a is the smallest numbered interval in I_a but not in I_b.

Theorem 1 [Yannakakis and Gavril]. Algorithm 1 finds a maximum bipartite subgraph of an interval graph I given in the form of a family of intervals.

Proof. Assume S is not a maximum bipartite subgraph of I.

† If the endpoints are not distinct, then we can make them distinct by placing one endpoint just to the left or right of the other, depending on whether we consider intervals to be overlapping if they only overlap at one endpoint. It is important that we do not change the "overlap relationship" among the intervals during this process.

Step 5 of the algorithm ensures that S is an bipartite subgraph of I. Let I_B be a maximum bipartite subgraph of I with which S disagrees the latest, i.e., S disagrees with I_B at the highest numbered interval among all maximum bipartite subgraphs of I. Without loss of generality let's say S disagrees with I_B at interval f_s, and I_B disagrees with S at interval f_b. By Step 4 of the algorithm, we know the right_endpoint of interval f_s is less than that of interval f_b. So $f_s < f_b$ by the way intervals are numbered (Step 2). Knowing I_B is bipartite, the number of intervals in I_B numbered larger than f_b and overlapping f_b is at most 1, and let's call it f_c, if it exists. Thus f_b and f_c are the only two possible intervals numbered larger than f_s in I_B that can overlap f_s. Now we have two cases to consider: 1. $f_c \in S$; 2. $f_c \notin S$. In the first case, we can substitute f_s for f_b in I_B without violating the bipartiteness of I_B, because all intervals that overlap f_s in I_B are also in S. In the second case, we observe that at most one of f_b, f_c overlaps intervals numbered less than f_s in I_B. Hence we can substitute f_s for f_b or f_c whichever has a smaller left_endpoint in I_B without violating the bipartiteness of I_B. In either case, S disagrees with this new maximum bipartite subgraph at an interval numbered larger than f_s. Contradiction.

■

The most time consuming part of Algorithm 1 is sorting (Step 1) which takes O($n$log$n$) time. We determine which intervals to delete (Step 5) by keeping track of the interval with the second largest right_endpoint that overlaps another interval in S, and deleting all intervals in β that overlap this right_endpoint. The total amount of time spent in Step 5 is O(n). Hence the whole algorithm runs in time O($n$log$n$).

We will discuss the maximum bipartite subgraph with a chosen subset problem for interval graphs in section IV.

IV. Placing Intervals on Tracks

The maximum bipartite subgraph problem is also known as the maximum 2-colorable subgraph problem. The vertices in each color induce an independent set. In the case of interval graphs, the vertices in each color are non-overlapping intervals. If we place these non-overlapping intervals on a "track", then the maximum bipartite subgraph problem for interval graphs can be viewed as a *2-track assignment* problem:†

† In this paper we shall always sort the 2n endpoints; hence without loss of generality we can assume the left and right endpoints of intervals take on values between 0 and 2n − 1.

Given 2 tracks numbered from 0 through 2n−1, and n interval of the form (l_i, r_i); l_i, r_i take on values between 0 and 2n−1 inclusive; all 2n values of l_i and r_i are distinct.

Pack as many intervals on the 2 tracks as possible without overlapping.

The algorithm that solves the 2-track assignment problem is the same as Algorithm 1, except in Step 4 we actually place intervals on the tracks: Pick an interval with the smallest right_endpoint from β and place it on the "tighter" fitting track, i.e., the track having an interval with the largest right_endpoint, if it fits on both. Ties are broken arbitrarily.

Now let's consider a variation of the 2-track assignment problem - *left-end-limited 2-track assignment* problem:

Given 2 tracks - track 1 (2) is numbered from p_1 (p_2) through 2n−1, and n interval of the form (l_i, r_i); p_1, p_2, l_i, r_i take on values between 0 and 2n−1 inclusive; all 2n values of l_i and r_i are distinct.

Pack as many intervals on the 2 tracks as possible without overlapping.

We solve the left-end-limited 2-track assignment problem by first deleting all those intervals which can not be placed on the limited tracks, i.e., deleting all intervals that either contain the point $\min\{p_1,p_2\}-1$ or end before $\min\{p_1,p_2\}$, and then calling the 2-track assignment algorithm on the remaining intervals with one modification: after Step 2 and before Step 3, place dummy intervals $(0, p_1-1)$ and $(0, p_2-1)$ on track1 and track2, respectively. It is clear the complexity of the algorithm for the left-end-limited 2-track assignment problem is also $O(n\log n)$.

We can define the *right-end-limited 2-track assignment* problem accordingly. The right-end-limited 2-track assignment problem is actually the left-end-limited 2-track assignment problem in reverse. Hence the right-end-limited 2-track assignment problem can also be solved in time $O(n\log n)$.

Another variation of the 2-track assignment problem is the *two-end-limited 2-track assignment* problem:

Given 2 tracks - track 1 (2) is numbered from p_1 (p_2) through q_1 (q_2), and n interval of the form (l_i, r_i); p_1, p_2, q_1, q_2, l_i, r_i take on values between 0 and 2n−1 inclusive; all 2n values of l_i and r_i are distinct.

Pack as many intervals on the 2 tracks as possible without overlapping.

The approach we take in solving the two-end-limited 2-track assignment problem is similar to that of Algorithm 1 with one difference: in Algorithm 2 we look for opportunities to reduce this problem to a left-end-limited 2-track assignment problem plus a right-end-limited 2-track assignment problem. The algorithm goes as follows:

Input. 2 tracks numbered from p_1 (p_2) through q_1 (q_2); an interval graph I given as a set of intervals, $I = \{f_1, f_2, \cdots, f_n\}$.

Output. 2-track assignment – a placement of the maximum number of intervals on the 2 tracks without overlapping.

1. Sort the 2n endpoints in ascending order.
2. Number the n intervals according to their right_endpoints in ascending order.
   Let β, β = {1, 2, ..., n}, be the collection of n intervals.
3. Delete all intervals from β that either contain the point $\min\{p_1, p_2\}-1$ or end before $\min\{p_1, p_2\}$.
4. Delete all intervals from β that either contain the point $\max\{q_1, q_2\}+1$ or begin after $\max\{q_1, q_2\}$.
5. Place dummy intervals $(0, p_1-1)$ and $(0, p_2-1)$ on track1 and track2, respectively.
6. While β is nonempty do
   7. Determine f – the interval from β with the smallest right_endpoint.
   8. If the right_endpoint of f is greater than $\min\{q_1, q_2\}$ (the track with the smaller right end is filled), then place a dummy interval $(\min\{q_1, q_2\}+1,\ 2n-1)$ on the track with the smaller right end, if one is not already placed there.
   9. If there exists an interval f_i, on track1 or track2, such that f_i is the left-most interval which overlaps f, f_i is not a dummy interval, and f_i does not overlap any other interval already placed on the tracks, then do
      (we have found a cut at a point just to the left of f or f_i; see Figure 1 for illustration.)
      10. Add f, f_i, and all intervals on the same track as f_i and to the right of f_i (if they exist) back to list β.
      11. Solve the right-end-limited 2-track assignment problem on track1 and track2 with intervals remaining in β.
      12. Exit Algorithm 2.

   Now we know no interval satisfies the condition described in Step 9.

   13. Place f on the tighter track, if it fits on both.
   14. Remove all intervals which overlap two intervals already placed on track1 and track2 from β.

   end while

Algorithm 2. Two-end-limited 2-track assignment algorithm.

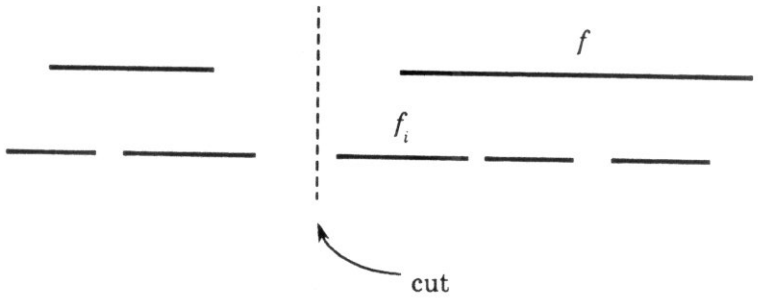

Figure 1. An example where a cut is found just to the left of f_i.

Theorem 2. Algorithm 2 correctly solves the two-end-limited 2-track assignment problem.

Proof. Algorithm 2, like Algorithm 1, is basically a greedy algorithm. If the condition described in Step 9 is never true, then we can use similar arguments as those in the proof of Algorithm 1 to show that there must exist a maximum two-end-limited 2-track assignment T such that T agrees with intervals picked by Algorithm 2.

On the other hand, if the condition described in Step 9 is true at some point, then using similar arguments as above, we know there exists a maximum two-end-limited 2-track assignment T_1 such that T_1 agrees with intervals picked by Algorithm 2 up to that point of the algorithm. This means that intervals f and f_i of Step 9 are in T_1. Since f and f_i do not overlap intervals that begin to their left in T_1, there must exist an unit interval gap (a_i,a_{i+1}) just to the left of f or f_i that is not covered by any interval in T_1. Knowing T_1 is a maximum two-end-limited 2-track assignment, we can cut the 2 tracks at the unit interval (a_i,a_{i+1}) and reduce the two-end-limited 2-track assignment problem to one left-end-limited 2-track assignment problem plus one right-end-limited 2-track assignment problem.

■

Algorithm 2 not only selects but also places the intervals on tracks subject to the constraints of the limited tracks. Note in the above proof, the reason the arguments in the proof of Theorem 1 apply when the condition described in Step 9 is never true is because, in this case, the bipartite subgraph corresponding to those intervals picked by

Algorithm 2 which fit between the overlapping part of the two tracks (between $\max\{p_1,p_2\}$ and $\min\{q_1,q_2\}$) is connected; hence there is exactly one way to place the intervals on tracks – basically placing an interval on the tighter track when it fits on both is the correct strategy. However when the condition described in Step 9 is true at some point, due to the limited right-end of the tracks, the proper choice for placing intervals f and f_i of Step 9 on the tracks to the right of the gap (a_i,a_{i+1}) is not apparent. (Note the strategy for selecting the intervals is still correct.) In this case, cutting the tracks at (a_i,a_{i+1}) and approaching the problem from the right-end will give us the proper placement of intervals on tracks.

The complexity of Algorithm 2 is still dominated by sorting (Step 1) which is $O(n\log n)$. All other steps take linear time. Hence the total running time of Algorithm 2 is $O(n\log n)$.

Now we will consider the maximum bipartite subgraph with a chosen subset problem for interval graphs. Without loss of generality we can assume the intervals in the chosen subset are bipartite. It is easier to view the maximum bipartite subgraph problem on interval graphs as a 2-track assignment problem when a subset is included. So in what follows, we shall discuss only the 2-track assignment problem. If two of the intervals in the chosen subset overlap, then we can reduce the original problem to two subproblems by cutting the tracks at a point of overlap. Hence without loss of generality we can assume all the intervals in the chosen subset are non-overlapping. There are 3 varieties of the 2-track assignment problem with a chosen subset: 1). un-limited, 2). left-end-limited (or right-end-limited), and 3). two-end-limited.

At the first sight, adding a chosen subset to the 2-track assignment seems to increase the complexity of this problem. However the greedy approach of Algorithm 1 and Algorithm 2 can still be employed to solve the 2-track assignment problem with a chosen subset; although some care must be taken.

We will use Algorithm 1 (or its variation) to solve the first 2 varieties of the 2-track assignment problem with a chosen subset, and Algorithm 2 to solve the third variety, and in both cases we will apply the following modifications:

Delete all the intervals in the chosen subset from β.

Place the intervals in the chosen subset, one at a time, on the tighter fitting track, in increasing order of their right_endpoint at the appropriate times.

By the appropriate times, we mean: 1. an interval f_i in the chosen subset is placed on the track right after the first interval that overlaps and ends before f_i is placed on the track, if such an interval exists, and 2. all intervals that end after f_i, for some f_i in the chosen subset, are placed on the tracks after f_i. These modifications can be done with some simple book keeping. Note Step 5 of Algorithm 1, and similarly Step 14 of Algorithm 2, is carried out even when an interval in the chosen subset is placed on the track. This is done to ensure that the 2-track assignment is always valid, i.e., the corresponding subgraph is always bipartite.

The correctness of the above approach is validated by proofs of Theorem 1 and 2, because any maximum 2-track assignment must include all the intervals in the chosen subset; hence substitution, the method used in showing the correctness of Algorithm 1, is applied only to those intervals not in the chosen subset, and remains valid.

V. Proper Circular-Arc Graphs

In the following we will assume that a proper circular-arc graph C is given in the form of a family of arcs, $C = \{c_1, c_2, \cdots, c_n\}$, on a circle, where each arc is denoted by two endpoints – left and right, in clockwise direction. Again without loss of generality we can assume the 2n endpoints of the arcs are distinct. As pointed out in the preliminaries, if a graph G is given in the form of vertices and edges, then we can use Tucker's algorithm to construct a circular-arc representation of G in $O(n^2)$ time.

We will consider the following two possibilities when finding a maximum bipartite subgraph for a proper circular-arc graph C:

1. C has a maximum bipartite subgraph that is also an interval graph.
2. Not case 1, that is, the union of arcs of every maximum bipartite subgraphs of C covers the circle.

If case 1 is true, then we can reduce this problem on a proper circular-arc graph to 2n problems on interval graphs by cutting the circle at each of the 2n endpoints, apply Algorithm 1, and take the maximum bipartite subgraph we find. Recall in a proper circular-arc graph no arc is properly contained within another. Thus any arc in a bipartite subgraph overlaps at most two other arcs in the subgraph. Now if case 2 is true, then any maximum bipartite subgraph of C must be an even cycle (in the degenerate case, it consists of 2 arcs or vertices), since each arc in the maximum bipartite subgraph must

overlap exactly one arc to the right and one arc to the left of it. The maximum independent set of an even cycle consists of half of the arcs in the cycle. Note the other half is also a maximum independent set of the even cycle. As we shall show in the proof of Theorem 3, if all maximum bipartite subgraphs of C are even cycles, then there exists a maximum bipartite subgraph S of C such that S consists of the union of maximum independent set of interval graphs C_a and C_b, for some a, b, $1 \leq a, b \leq 2n$, where C_i is the set of arcs in C except all those that contain interval (a_i, a_{i+1}) as defined in Step 2 of Algorithm 3.

The following algorithm describes the details of the above approach.

Input. A proper circular-arc graph C given as a set of arcs, $C = \{c_1, c_2, \cdots, c_n\}$.

Output. A maximum bipartite subgraph of C.

1. Sort the 2n endpoints in clockwise direction and output them in a list α, $\alpha = \{a_1, a_2, \cdots, a_{2n}\}$, where $a_{2n+1} = a_1$.
2. For i = 1 to 2n, determine C_i.
   C_i is defined as the set of arcs in C except all those that contain interval (a_i, a_{i+1}). Note C_i is an interval graph.
3. For i = 1 to 2n obtain a maximum bipartite subgraph of C_i by running Algorithm 1.
4. For i = 1 to 2n obtain a maximum independent set M_i of C_i by the following steps:†
   5. Reindex the endpoints in C_i so a_{i+1} becomes a_1, a_{i+2} becomes a_2, ... , a_i becomes a_{2n}.
   6. Number the n_i intervals in C_i according to their right_endpoints in ascending index order. Let β_i, β_i = {1, 2, ..., n_i), be the collection of n_i intervals.
   7. While β_i is nonempty do
      8. Pick an interval with the smallest right_endpoint from β_i and add it to a set called M_i (originally empty).
      9. Delete all intervals from β_i which overlap any interval already in M_i.
      end while
   10. Let $first_i$ ($last_i$) denote the first (last) arc placed in set M_i.
11. For each set M_i, find a set M_j, if one exists, such that $|M_i| = |M_j|$ and $M_i + M_j$ is an even cycle. Let M_{ij} denote the set containing such an even cycle.
12. Return the maximum bipartite subgraph produced in Steps 3 and 11.

Algorithm 3. Maximum bipartite subgraph algorithm for proper circular-arc graphs.

† This maximum independent set algorithm is presented by Gupta, Lee, and Leung[1982].

Theorem 3. Algorithm 3 finds a maximum bipartite subgraph of a proper circular-arc graph C.

Proof. If C has a maximum bipartite subgraph that is also an interval graph, then Algorithm 3 finds it at Step 3.

Now assume all maximum bipartite subgraphs of C are even cycles. Let B be one of them. B consists of two independent sets, say X and Y, of equal size. Without loss of generality let (a_i, a_{i+1}) be an unit interval on the circle such that exactly one arc in B covers it. Without loss of generality let Y be the set that contains the arc, say y_1, covering interval (a_i, a_{i+1}). We shall label the arcs in Y, Y = $\{y_1, y_2, \cdots, y_r\}$, according to their right_endpoints in clockwise direction. Similarly we shall label the arcs in X, X = $\{x_1, x_2, \cdots, x_r\}$, according to their right_endpoints in clockwise direction such that x_1 overlaps y_1 and y_2. In general, x_i overlaps y_i and y_{i+1}. Note X is an independent set of interval graph C_i, and M_i (obtained through Steps 4-10) is a maximum independent set of C_i. Let's label the arcs in M_i, $M_i = \{m_{i1}, m_{i2}, \cdots, m_{il}\}$, according to the order the arcs are placed in M_i. Observe m_{ij} never ends after x_j ends (i.e., the right_endpoint of $m_{ij} \leq$ the right_endpoint of x_j) due to the way m_{ij} is picked (Step 8). Since C is a proper circular-arc graph, m_{ij} never begins after x_j begins (i.e., the left_endpoint of $m_{ij} \leq$ the left_endpoint of x_j). So m_{ij} either equals or proceeds x_j in clockwise direction. Figure 2 illustrates the relationship among the arcs in sets X, Y, and M_i.

Now we claim the union of two independent sets M_i and Y is also a maximum bipartite subgraph of C. First we note that the union of M_i and Y is a bipartite subgraph of C. Second we observe that the size of M_i is no less than size of X. Thus the only way M_i + Y is not a maximum bipartite subgraph of C is when the two sets have arc(s) in common. Without loss of generality let m_{ij} be the first arc in M_i that is in Y, i.e., $m_{ij} = y_k$, for some k. In this case we know m_{ij} proceeds x_j in clockwise direction. Since x_j overlaps y_j and y_{j+1} and $m_{ij} = y_k$, the index j must be equal to or greater than k, i.e., $j \geq k$. Now let's consider the set Z = $\{m_{i1}, m_{i2}, \cdots, m_{ik-1}, x_k, x_{k+1}, \cdots, x_r\}$. Z is an independent set; |Z| = |X|; Z is disjoint from Y. Thus Z + Y is a maximum bipartite subgraph of C. Note m_{ik-1} does not overlap y_k, because $m_{ij} = y_k$, and $j \geq k$. Then Z + Y is an interval graph, since the gap between y_{k-1} and y_k is not completely covered by any of the arcs in Z. This is a contradiction because we assumed C does not have any maximum bipartite subgraph that is an interval graph. Therefore M_i + Y must be a

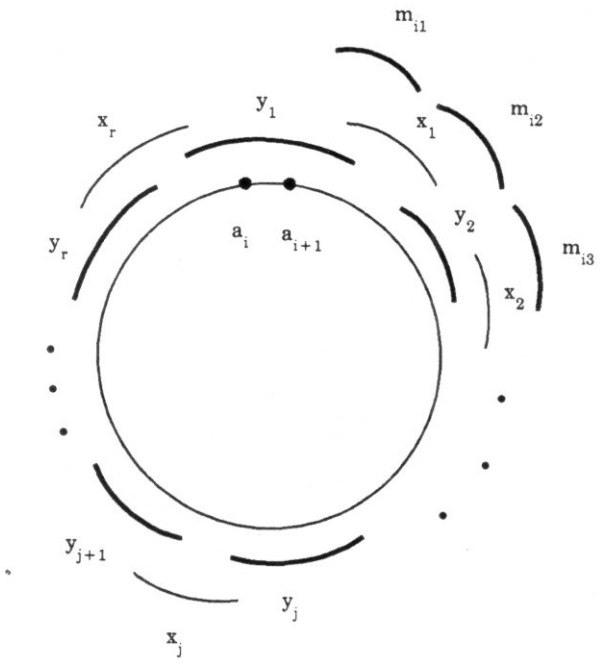

Figure 2. Arcs in sets X, Y, and M_i.

maximum bipartite subgraph of C.

Using similar arguments as above we can show that if (a_j, a_{j+1}) is an unit interval on the circle such that none of the arcs in Y covers it (this implies there exists an arc in M_i that covers it), then $M_i + M_j$ is also a maximum bipartite subgraph of C. Therefore we just proved that if all maximum bipartite subgraph of C are even cycles, then there exists a maximum bipartite subgraph M_{ij} of C such that M_{ij} consists of the union of M_i and M_j, for some $i, j, 1 \leq i,j \leq 2n$. Algorithm 3 finds $M_i + M_j = M_{ij}$ at Step 11.

■

Before analyzing the running time of Algorithm 3, we need to specify how to perform Step 11. First we prove the following:

Lemma 4. Given two independent sets M_i and M_j (as defined in Step 4 of Algorithm 3) of the same size, if either

1. $last_i$ overlaps $last_j$, and $first_i$ overlaps $last_j$ and $first_j$, or

2. $last_j$ overlaps $last_i$, and $first_j$ overlaps $last_i$ and $first_i$,

then $M_i + M_j$ forms an even cycle.

Proof. Without loss of generality let's assume case 1 is true, i.e., 1. M_i and M_j are two independent sets, 2. $|M_i| = |M_j|$, 3. $last_i$ overlaps $last_j$, and 4. $first_i$ overlaps $last_j$ and $first_j$. To prove $M_i + M_j$ forms an even cycle, it suffices to show that I. M_i is disjoint from M_j, and II. arcs in M_i and M_j cover the whole circle.

I. M_i is disjoint from M_j.

Suppose not. Let c be the first arc that is placed in both M_i and M_j. According to the maximum independent set algorithm (Steps 5-9) all the arcs placed in sets M_i and M_j after c are the same, with the possible exception of the last arc that is placed in M_j. Thus $last_i$ is also placed in M_j. Since $first_i$ overlaps $last_j$, $last_j$ is not the same as $last_i$. The above two statements imply $last_i$ does not overlap $last_j$. Contradiction.

II. Arcs in M_i and M_j cover the whole circle.

Suppose not. Without loss of generality let (a_k, a_{k+1}) be an interval on the circle not covered by arcs in either set. The maximum independent set algorithm (Steps 5-9) would have placed the same arcs in both sets M_i and M_j after the point a_{k+1}, with the possible exception of the last arc that is placed in M_j. Using similar arguments as above, we again reach a contradiction.

■

Now we are justified to restate Step 11 of Algorithm 3 as: For each set M_i, find a set M_j, if one exists, such that $|M_i| = |M_j|$, and either $last_i$ overlaps $last_j$, and $first_i$ overlaps $last_j$ and $first_j$, or $last_j$ overlaps $last_i$, and $first_j$ overlaps $last_i$ and $first_i$.

After the initial sort (Step 1) which takes $O(n \log n)$ time, Steps 3, 4-10, and 11 all can be carried out in time $O(n^2)$. Therefore we can find a maximum bipartite subgraph of a proper circular-arc graph in $O(n^2)$ time.

The complexity of the maximum bipartite subgraph with a chosen subset problem for proper circular-arc graphs is the same as that for circular-arc graphs, i.e., $O(n \log n + n\Delta)$, where Δ is the maximum number of arcs covering any point on the circle. We will discuss the algorithm for finding a maximum bipartite subgraph with a chosen subset for circular-arc graphs in section VI.

VI. Circular-Arc Graphs

Assume a circular-arc graph is given in the form of a family of arcs $C = \{c_1, c_2, \cdots, c_n\}$ on a circle, where each arc is denoted by two endpoints – left and right, in clockwise direction. Again without loss of generality we can assume the 2n endpoints of the arcs are distinct.

The approach we take in finding a maximum bipartite subgraph of a circular-arc graph is as follows: For any unit interval (a_i, a_{i+1}) on the circle, exactly one of the following three cases is true of any maximum bipartite subgraph C_B of C:

1. None of the arcs in C_B covers the interval (a_i, a_{i+1}).
2. Exactly one arc in C_B covers the interval (a_i, a_{i+1}).
3. Exactly two arcs in C_B cover the interval (a_i, a_{i+1}).

In the first case, C_B is an interval graph. To obtain C_B we can cut the circle at the interval (a_i, a_{i+1}), and find a maximum bipartite subgraph on the corresponding interval graph. The second and third cases can be handled by considering all possible combinations of arcs covering interval (a_i, a_{i+1}), cutting the circle at the interval (a_i, a_{i+1}), and solving the corresponding two-end-limited 2-track assignment problem (the exact details are given in Algorithm 4). Not knowing which of the three cases holds for any particular interval (a_i, a_{i+1}), we shall solve them all and take the maximum bipartite subgraph we find. An unit interval (a_i, a_{i+1}) best suited for the above approach is one with the least number of arcs covering it in C. All these steps are described in Algorithm 4.

Input. A circular-arc graph C given as a set of arcs, $C = \{c_1, c_2, \cdots, c_n\}$.

Output. A maximum bipartite subgraph of C.

1. Sort the 2n endpoints in clockwise direction and output them in a list α, $\alpha = \{a_1, a_2, \cdots, a_{2n}\}$, where $a_{2n+1} = a_1$.
2. For i = 1 to 2n, determine D_i.
   D_i is defined as the set of arcs covering the interval (a_i, a_{i+1}).
3. Determine k such that $|D_k| = \min\{|D_i|, 1 \le i \le 2n\}$.
4. Obtain a maximum bipartite subgraph of the interval graph $C - D_k$ by running Algorithm 1.
5. For each arc $c_i = (a_{l_i}, a_{r_i})$ in D_k do
   (c_i is the only arc covering the interval (a_k, a_{k+1}) in the bipartite subgraph.)
   6. Solve the corresponding two-end-limited 2-track assignment problem of C (Algorithm 2) by mapping endpoint a_{k+1} to 0, a_{k+2} to 1, ... , a_k to $2n-1$, and

numbering track1, track2 from 0 to 2n−1, $|r_i - k|$ to $|2n - k + l_i|$, respectively.

7. Let G_i be the set of arcs placed on track1 and track2 in Step 6 plus arc c_i.

8. For each pair of arcs $c_i = (a_{l_i}, a_{r_i})$ and $c_j = (a_{l_j}, a_{r_j})$ in D_k do
(both arcs c_i and c_j cover the interval (a_k, a_{k+1}) in the bipartite subgraph.)

9. Solve the corresponding two-end-limited 2-track assignment problem of C (Algorithm 2) by mapping endpoint a_{k+1} to 0, a_{k+2} to 1, ... , a_k to 2n−1, and numbering track1, track2 from $|r_i - k|$ to $|2n - k + l_i|$, $|r_j - k|$ to $|2n - k + l_j|$, respectively.

10. Let G_{ij} be the set of arcs placed on track1 and track2 in Step 9 plus arcs c_i and c_j.

11. Return the maximum bipartite subgraph produced in Steps 4, 7, and 10.

Algorithm 4. Maximum bipartite subgraph algorithm for circular-arc graphs.

Let's define δ to be the minimum number of arcs covering any particular interval (a_i, a_{i+1}) on the circle; $\delta = |D_k|$ in Algorithm 4. We shall parametrize the running time of Algorithm 4 in terms of δ. After the 2n endpoints are sorted (Step 1), the second *for* loop (Steps 5-7) takes $O(n\delta)$ time, since each call to Algorithm 2 takes linear time. The third *for* loop (Steps 8-10) takes $O(n\delta^2)$ time, since there are δ^2 number of pairs of arcs in D_k to consider. Thus the total running time of Algorithm 4 is $O(n\delta^2)$. Note δ is upper bounded by the minimum degree of vertices in C. Clearly δ^2 is upper bounded by e − the number of edges in circular-arc graph C. δ^2 is also upper bounded by the number of edges in a minimum maximal clique of C.

Circular-arc graphs is one class of graphs where the complexity of finding a maximum bipartite subgraph is reduced, from $O(n\delta^2)$ to $O(n\log n + n\Delta)$, when a chosen subset is included (the size of the chosen subset is greater than 0), where Δ is the maximum number of arcs covering any point on the circle. The reason is that if an arc is included in the bipartite subgraph, then we can narrow the searching space for optimal solutions. First, we observe that if there exists two overlapping arcs in the chosen subset, then we can cut the circle at a point of overlap and reduce the original problem on a circular-arc graph to a two-end-limited 2-track assignment problem with the chosen subset (the chosen subset will not include the two overlapping arcs). The complexity of the algorithm in this case is $O(n\log n)$. Second, we note that given that an arc (a_i, a_j) is included in the bipartite subgraph, then in any maximum bipartite subgraph the unit interval (a_i, a_{i+1}) is covered with exactly one arc or exactly two arcs on the circle. We have dealt with this situation previously in circular-arc graphs, but now we know arc (a_i, a_j) is in the bipartite subgraph. Thus we only have to consider all possible arcs (other than (a_i, a_j))

covering or not covering the unit interval (a_i, a_{i+1}), and solve the corresponding two-end-limited 2-track assignment problem with a chosen subset. The number of arcs we need to consider is at most Δ. Note $\Delta \leq d \leq n$, where d is the maximum degree of vertices in C. Hence the complexity of finding a maximum bipartite subgraph with a chosen subset for circular-arc graphs and for proper circular-arc graphs is $O(n\log n + n\Delta)$.

VII. Permutation Graphs

Suppose $\Pi = [\pi_1, \pi_2, \cdots, \pi_n]$ is a permutation of n numbers. Let π_i^{-1} denote the position in the sequence where the number i can be found. A *permutation* graph G[Π] is an undirected graph on n vertices such that vertex i is adjacent to vertex j if the larger of $\{i, j\}$ appears to the left of the smaller in Π. More formally, the graph G[Π] = (V, E) is defined as follows:

$$V = \{1, 2, \ldots, n\}$$

and

$$ij \in E \iff (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0.$$

An undirected graph G is a permutation graph if there exists a permutation Π such that G is isomorphic to G[Π].

As mentioned in the preliminaries, permutation graphs are also characterized as comparability graphs whose complements are also comparability graphs. Permutation graphs are a proper subclass of comparability (or transitively orientable) graphs which are a proper subclass of perfect graphs. To obtain a transitive orientation $\vec{G}[\Pi]$ of a permutation graph G[Π], we simply orient each edge toward the larger vertex. Note after orientation, the resulting directed graph must be acyclic. A *transitively reduced* graph $\vec{G}_t[\Pi]$ of $\vec{G}[\Pi]$ is a directed graph which consists of vertices in $\vec{G}[\Pi]$, and an arc (a,c) is in $\vec{G}_t[\Pi]$ if and only if (a,c) is in $\vec{G}[\Pi]$ and there exists no vertex b such that (a,b) and (b,c) are in $\vec{G}[\Pi]$. Note $\vec{G}_t[\Pi]$ is a subgraph (not induced) of $\vec{G}[\Pi]$. An interesting property of permutation graphs is that the complement of a permutation graph is another permutation graph. If we let Π^r be the reverse of permutation Π, then it is clear $\overline{G[\Pi]} = G[\Pi^r]$.

Given a permutation graph G, a suitable permutation Π can be constructed in time $O(de + n^2)$ [Golumbic, 1980], where d is the maximum degree of a vertex. The transitively reduced graph $\vec{G}_t[\Pi]$ can be constructed from Π in $O(n^2)$ time. In the following, we will assume a permutation graph G is given as G[Π].

Two elementary properties of permutation graphs G[Π] are: 1. the increasing subsequences of Π and the independent sets of G[Π] are in one-to-one correspondence, and 2. the decreasing subsequences of Π and the cliques of G[Π] are in one-to-one correspondence. Thus a maximum bipartite subgraph of G[Π] corresponds to a longest subsequence Π_1 of Π such that Π_1 is composed of two increasing subsequences of Π.

In developing an algorithm for finding a maximum bipartite subgraph of G[Π], we will be working with the complement of the permutation graph, i.e., $G[\Pi^r]$. More specifically, we will be dealing with the transitive directed graph $\vec{G}[\Pi^r]$, which is obtained from $G[\Pi^r]$ by orienting each edge toward the larger vertex. Now if $(\pi_a{}^r, \pi_b{}^r)$ is a directed edge in $\vec{G}[\Pi^r]$, then $\pi_b{}^r > \pi_a{}^r$ and $b < a$. Note directed paths in $\vec{G}[\Pi^r]$ and cliques of $G[\Pi^r]$ are in one-to-one correspondence; equivalently directed paths in $\vec{G}[\Pi^r]$ and independent sets of of G[Π] are in one-to-one correspondence. Our strategy for finding a maximum bipartite subgraph of G[Π] is to look for 2 directed paths in $\vec{G}[\Pi^r]$ containing the maximum number of vertices. Note the 2 directed paths need not be disjoint. To find 2 such directed paths in $\vec{G}[\Pi^r]$, we first compute, for each pair of vertices u and v, a *best-pair-of-paths* from u and v in $\vec{G}[\Pi^r]$, $1 \le u,v \le n$. A best-pair-of-paths from vertices u and v in $\vec{G}[\Pi^r]$ is defined as 2 directed paths starting from vertices u and v and containing the maximum number of vertices in $\vec{G}[\Pi^r]$. A best-pair-of-paths which contains the maximum number of vertices among all best-pair-of-paths in $\vec{G}[\Pi^r]$ corresponds to a maximum bipartite subgraph of G[Π].

An interesting observation we make is that all best-pair-of-paths in $\vec{G}[\Pi^r]$ use only edges in the transitively reduced graph $\vec{G}_t[\Pi^r]$ of $\vec{G}[\Pi^r]$. For if there exists a directed path from u to v in $\vec{G}[\Pi^r]$, then there exists a directed path from u to v traversing only transitively reduced edges that is as long as any other directed path from u to v in $\vec{G}[\Pi^r]$. Thus we only have to deal with the transitively reduced graph $\vec{G}_t[\Pi^r]$ when computing a maximum bipartite subgraph of G[Π].

We now present a simple algorithm, based on dynamic programming, for finding a maximum bipartite subgraph of a permutation graph. It goes as follows:

Input. A permutation graph G[Π].

Output. A maximum bipartite subgraph of G[Π].

1. Let Π^r, the reverse of Π, be indexed as: $\Pi^r = [\pi_1^r, \pi_2^r, \cdots, \pi_n^r]$.
2. Construct the transitively reduced graph $\vec{G}_t[\Pi^r]$ of $\vec{G}[\Pi^r]$.
   The edges in $\vec{G}_t[\Pi^r]$ are always oriented toward the larger vertex.
3. For i = 1 to n, do
   4. For j = 1 to i, compute a *best-pair-of-paths* from vertices π_i^r and π_j^r in $\vec{G}_t[\Pi^r]$.
5. Return a best-pair-of-paths from π_a^r and π_b^r in $\vec{G}_t[\Pi^r]$, for some π_a^r and π_b^r, $1 \leq \pi_a^r, \pi_b^r \leq$ n, which contains the maximum number of vertices among all best-pair-of-paths computed in Step 4.

Algorithm 5. Maximum bipartite subgraph algorithm for permutation graphs.

Algorithm 5 correctly solves the maximum bipartite subgraph problem because the best-pair-of-paths computed in Step 5 corresponds to a longest subsequence Π_1 of Π such that Π_1 is composed of two increasing subsequences of Π which corresponds to a maximum bipartite subgraph of G[Π].

The dynamic programming part of Algorithm 5 comes into play in Step 4, and it proceeds as follows:

1. A best-pair-of-paths from π_i^r and π_j^r (j < i) = a best of {best-pair-of-paths from π_k^r and π_j^r, where π_k^r is a vertex pointed to by π_i^r} plus the vertex π_i^r.
2. A best-pair-of-paths from π_i^r and π_i^r, = a best of {best-pair-of-paths from π_k^r and π_i^r, where π_k^r is a vertex pointed to by π_i^r}.

Basically a best-pair-of-paths from π_i^r and π_j^r must go through some vertex, say π_k^r, pointed to by π_i^r. See Figure 3 for illustration. Note π_j^r may also be pointed to by π_i^r. Due to dynamic programming, a best-pair-of-paths from π_k^r and π_j^r, k, $j < i$, has been determined earlier, and that information is readily available. If there exists more that one best-pair-of-paths from π_i^r and π_j^r, then we can freely choose one. The amount of work needed to compute a best-pair-of-paths from π_i^r and π_j^r, for some $j < i$, is bounded by the outdegree of vertex π_i^r in the transitively reduced graph $\vec{G}_t[\Pi^r]$. Thus the total running time of Algorithm 5 is $O(\sum_v nd_v) = O(n\overline{e_t})$, where d_v is the outdegree of vertex v and $\overline{e_t}$ is the number of edges in $\vec{G}_t[\Pi^r]$.

Due to the nature of dynamic programming, we can easily adapt Algorithm 5, with some simple modifications, to solve the maximum bipartite subgraph with a chosen subset problem for permutation graphs. Let S be the chosen set, S = $\{\pi_a^r, \pi_b^r, \cdots, \pi_s^r\}$,

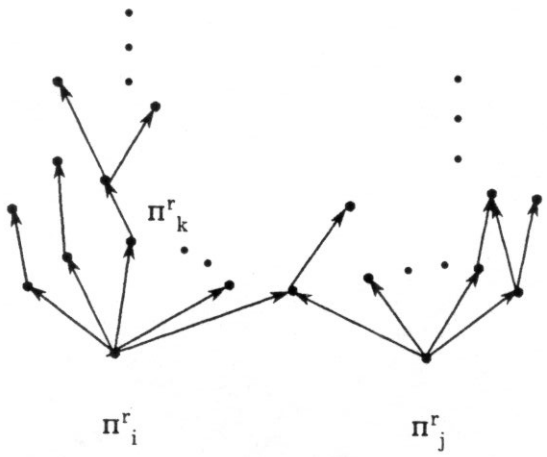

Figure 3. A best-pair-of-paths from Π^r_i and Π^r_j through Π^r_k.

$a < b < \cdots < s$. Because all directed paths $\pi_{v_1} \rightarrow \pi_{v_2} \rightarrow \cdots \rightarrow \pi_{v_l}$ in $\vec{G}_t[\Pi^r]$ have the property that $\pi_{v_1} < \pi_{v_2} < \cdots < \pi_{v_l}$ and $v_1 > v_2 > \cdots > v_l$, a best-pair-of-paths from π^r_i and π^r_j, $j \leq i$, must include all those vertices π^r_h of S such that $h \leq i$, if they exist. For if a vertex π^r_h of S, $h \leq i$, is not included in a best-pair-of-paths from π^r_i and π^r_j, $j \leq i$, then all best-pair-of-paths that build on top of best-pair-of-paths from π^r_i and π^r_j will not include the vertex π^r_h either. To find a maximum bipartite subgraph which includes S, we need to modify Step 4 of Algorithm 5 as follows: For j = 1 to i, compute a *best-pair-of-paths* from vertices π^r_i and π^r_j in $\vec{G}_t[\Pi^r]$ which includes all those vertices π^r_h of S with $h \leq i$. This modified Step 4 can be computed by considering only those vertices π^r_k which are pointed to by π^r_i such that a best-pair-of-paths from π^r_k and π^r_j plus the vertex π^r_i include all those vertices π^r_h of S with $h \leq i$, if such a best-pair-of-paths exists. Of course the best-pair-of-paths determined in Step 5 of Algorithm 5 must include the whole set S. The complexity of this modified algorithm is still $O(n\bar{e}_t)$, where $\bar{e}_t$ is the number of edges in $\vec{G}_t[\Pi^r]$.

VIII. Split Graphs

A graph G = (V, E) is called a *split* graph if V can be partitioned into two subsets I and C such that I is an independent set and C induces a complete graph. Split graphs are also characterized as chordal graphs whose complements are also chordal graphs. Split

graphs are a proper subclass of chordal graphs. It is clear that the complement of a split graph is also a split graph.

In this section we present a simple algorithm for finding a maximum bipartite subgraph of a split graph. First we make the following observation: If we partition the vertex set V of a split graph into two subsets I and C such that I is an independent set and C induces a complete graph (for example, if we use a maximum independent set algorithm to find I), then any maximum bipartite subgraph of G contains at least one and at most two vertices from C. Thus our task becomes: Find a bipartite subgraph of size |I| + 2, if it exists. Otherwise the set I plus any vertex in C is a maximum bipartite subgraph of G. The algorithm for finding a maximum bipartite subgraph of G goes as follows:

Input. A split graph G = (V,E).

Output. A maximum bipartite subgraph of G.

1. Find a maximum independent set of G using Gavril's[1972] algorithm.
   Let $I_{\max}$ denote the maximum independent set, and C denote the set of vertices which induces a complete graph, $I_{\max}$ + C = V.
2. Determine M which is defined to be the adjacency matrix with $(M)_{u,v} = 1$ if and only if (u,v)$\in$E, $u \in I_{\max}$, and $v \in C$.
3. Calculate M^2 which is the boolean multiplication of M with itself.
4. If there exist $v_i, v_j \in C$ such that $(M^2)_{v_i,v_j} = 0$, then
   5. Let B = $I_{\max} + v_i + v_j$.
   Otherwise
   6. Let B = $I_{\max} + v$, for any v in C.
7. Return B.

Algorithm 6. Maximum bipartite subgraph algorithm for split graphs.

To see Algorithm 6 correctly finds a maximum bipartite subgraph of a split graph G, we note that two vertices v_i, v_j in C form a bipartite subgraph with $I_{\max}$ only if there exists no vertex u in $I_{\max}$ such that u is adjacent to both v_i and v_j, in other words, only if the length of the shortest path between v_i and v_j in the subgraph (actually a bipartite subgraph) determined by the adjacency matrix M (Step 2) is more than two, or $(M^2)_{v_i,v_j} = 0$.

Gavril's maximum independent set algorithm on chordal graphs takes $O(n + e)$ time when it is combined with a perfect vertex elimination scheme of the same time bound (an

$O(n + e)$ algorithm of Rose, Tarjan, and Lueker[1976] will do). Thus the running time of Algorithm 6 is dominated by matrix multiplication (Step 3), in other words, the complexity of Algorithm 6 is equivalent to that of matrix multiplication. Presently the most efficient, asymptotically speaking, matrix multiplication algorithm runs in time $O(n^{2.376})$ [Coppersmith and Winograd, 1987]. (To be more precise, an nxn matrix may be multiplied using $O(n^{2.376})$ arithmetical operations; 2.376.. is the *matrix exponent*. For a more practical matrix multiplication algorithm – $O(n^{2.807})$, see Strassen[1969]). Therefore we can find a maximum bipartite subgraph for a split graph in time $O(n^{2.376})$.

Adding the constraint that a maximum bipartite subgraph must include a chosen subset of vertices may simplify the problem when we are dealing with split graphs. For we know a bipartite subgraph can include at most two vertices from C, and even knowing one vertex in C must be included in the bipartite subgraph will narrow down our choices. The complexity of finding a maximum bipartite subgraph with a chosen subset for split graphs, however, remains the same as the complexity of matrix multiplication, since the chosen subset may contain only vertices that are in I, and in which case we will have to use Algorithm 6.

In algorithm 6 we have reduced the problem: find a bipartite subgraph of size $|I| + 2$, if it exists, to the problem: determine if there exists two vertices in C such that the shortest path between those two vertices is more than 2, and we solved the latter by matrix multiplication. Now we observe that the second problem falls in the framework of a more general question: is the diameter of a bipartite graph larger than 2? The diameter of a graph is defined as the length of any longest shortest path between 2 vertices in the graph. The complexity of this problem is the same as that of matrix multiplication. An open problem is whether we can improve the running time of this algorithm, or whether we can use a more combinatorial approach and achieve a time bound better than $O(ne)$ (i.e., without using matrix multiplication). Interestingly, the complexity of determining whether the diameter of a bipartite graph is greater than k, for any fixed k, is the same as that of matrix multiplication; however, when k is not fixed, the complexity becomes $O(ne)$ using breadth first search or $O(n^3(\log\log n/\log n)^{1/3})$ using an algorithm of Fredman[1976].

IX. Concluding Remarks and Suggestions for Future Research

In this paper we have used the dynamic programming technique in numerous occasions to solve the maximum bipartite subgraph problem, including interval graphs, 2-track assignment problems, circular-arc graphs, and permutation graphs (the greedy approach of Algorithms 1 and 2 can be considered as dynamic programming). In fact, dynamic programming is also employed to solve the same problem for chordal graphs, by Yannakakis and Gavril[1987]. We believe the approaches of Bern, Lawler, and Wong[1987] and Baker[1983], both of which use dynamic programming, can be applied to solve the maximum bipartite subgraph problem for outerplanar and k-outerplanar graphs, respectively; although we have not checked the details. Dynamic programming has proven itself to be a powerful technique. However, it is not powerful enough (at least it has not been shown to be) to solve the maximum bipartite subgraph problem for all classes of graphs when such polynomial time algorithms exist, as demonstrated by Frank[1980] for comparability graphs. Another interesting observation we make is that the decomposition approach in dynamic programming in each class of graphs is different. Usually we have put the graph in proper representation before the decomposition approach becomes apparent. One such example is permutation graphs; in Algorithm 5 the permutation graph is built as a transitively reduced transitive directed graph to be operated on by dynamic programming. Another example is chordal graphs; in the algorithm of Yannakakis and Gavril[1987], a chordal graph is first represented as an intersection graph of a family of subtrees in a tree before the algorithm proceeds.

Due to the nature of dynamic programming, we can easily adapt the algorithms based on dynamic programming for finding a maximum bipartite subgraph, with some simple modifications, to solve the maximum bipartite subgraph with a chosen subset problem, as we have done for interval graphs, 2-track assignment problems, circular-arc graphs, and permutation graphs. In the same spirit, the algorithm of Yannakakis and Gavril[1987] for chordal graphs can also be modified to solve the maximum bipartite subgraph with a chosen subset problem by considering only those subgraphs (cliques) such that if a vertex is in the chosen subset, then that vertex is colored in the subgraph, i.e., included in the maximum bipartite subgraph.

In this paper we have presented efficient algorithms for finding a maximum bipartite subgraph for various classes of graphs; however, the complexity of this problem for some

important classes of graphs still remain open, including perfect graphs. For perfect graphs, both independent set and chromatic number problems are known to be polynomial time solvable via the ellipsoid method (see Grotschel, Lovasz, and Schrijver[1981]). These results may or may not provide a clue to the complexity of the maximum bipartite subgraph problem for perfect graphs. In any event, determining its complexity can be interesting and challenging, and is left for future research. In addition, the complexity of the maximum bipartite subgraph with a chosen subset problem for comparability graphs and series-parallel graphs are open, and deserve further studies.

In this paper we have restricted our attention to those classes of graphs which have polynomial algorithms for the maximum bipartite subgraph problem. Another possible direction of study is in the development of approximation algorithms or heuristics for this problem, meaning algorithms which produce a bipartite subgraph of a certain size, possibly parameterized by the chromatic number, for all graphs. This is another interesting and challenging problem left for future studies.

Acknowledgements

We thank Mihalis Yannakakis for helpful discussion on the runtime of the maximum bipartite subgraph algorithm for chordal graphs. We thank Rachel Manber for numerous discussions and for pointing to us the relevant references.

References

Baker, B. S. [1983]

Approximation algorithms for NP-complete problems on planar graphs, *Proc. 24th Annual Symposium on Foundations of Computer Science,* pp.265-273.

Bern, M. W., E. L. Lawler, and A. L. Wong [1987]

Linear-time computation of optimal subgraphs of decomposable graphs *J. of Algorithms* **8**, pp.216-235.

Booth, K. S. and G. S. Lueker [1976]

Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *J. Comput. System Sci.* **13**, pp.335-379.

Coppersmith, D. and S. Winograd [1987]

Matrix multiplication via arithmetic progressions, *Proc. 19th Annual ACM Symposium on Theory of Computing,* pp.1-6.

Frank, A. [1980]

On chain and antichain families of a partially ordered set, *J. Combin. Theory Ser. B* **29**, pp.176-184.

Fredman, M. L. [1976]

New bounds on the complexity of the shortest path problem, *SIAM J. Comput.* **5**, pp.83-89.

Gavril, F. [1972]

Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM J. Comput.* **1**, pp.180-187.

Golumbic, M. [1980]

Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York.

Grotschel, M., L. Lovasz, and A. Schrijver [1981]

The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* **1**, pp.169-197.

Gupta, U. I., D. T. Lee, and Y.-T. Leung [1982]

Efficient algorithms for interval graphs and circular-arc graphs, *Networks* **12**, pp.459-467.

Johnson, D. [1985]

The NP-completeness column: an ongoing guide, *J. Algorithms* **6**, pp.434-451.

Lewis, J. M. and M. Yannakakis [1980]

The node-deletion problem for hereditary properties is NP-Complete, *J. of Comp. System Science* **20**, pp.219-230.

Manber, R. and G. Narasimhan [1987]

Algorithms for the maximum induced bipartite subgraph problem on interval and circular-arc graphs, U. of Wisconsin - Madison CS Technical Report #696, April, 1987.

Rose, D. J., R. E. Tarjan, and G. S. Lueker [1976]

Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, pp.266-283.

Sarrafzadeh, M. and D. T. Lee [1987]

A new approach to topological via minimization, Northwestern University, Center for Integrated Microelectronic Systems Technical Report CIMS-87-01, November, 1987.

Strassen, V. [1969]

Gaussian elimination is not optimal, *Numerische Mathematik* **13**, pp.354-356.

Takamizawa, K., T. Nishizeki, and N. Saito [1982]

Linear-time computability of combinatorial problems on series-parallel graphs, *JACM* vol 29, no 3, pp.623-641.

Tucker, A. [1971]

Matrix characterization of circular-arc graphs, *Pacific J. Math.* **39**, pp.535-545.

Tucker, A. [1980]

An efficient test for circular-arc graphs, *SIAM J. Comput.* **9**, pp.1-24.

Yannakakis, M. [1987]

Personal communication.

Yannakakis, M. and F. Gavril [1987]

The maximum k-colorable subgraph problem for chordal graphs, *IPL* **24**, pp.133-137.