TR-88-015

ANO(min[m,n,n'loglogn]) MAXIMUM WEIGHT CLIQUE ALGORITHM FOR CIRCULAR-ARC GRAPHS
AN $O(\min[m \cdot n, n^2 \log n])$ MAXIMUM WEIGHT CLIQUE ALGORITHM FOR CIRCULAR-ARC GRAPHS

Wei-Kuan Shih¹ and Wen-Lian Hsu²

Key Words: graph, clique, independent set, algorithm

ABSTRACT

Let $F$ be a family of arcs in a circle, where each arc is associated with a weight. The maximum weight clique problem is to find a subset of pairwise overlapping arcs in $F$ such that their total weight is maximum. This problem was originally solved by Hsu in $O(m \cdot n)$ time. Recently, a $O(n^2 \log n)$ algorithm was discovered by Apostolico and Hambrusch for the unweighted case, whereas the weighted case was posed as an open problem. We extend their approach to solve the weighted case in $O(\min[m \cdot n, n^2 \log n])$ time. Our algorithm is based on a characterization of maximum weight independent sets in bipartite graphs, which is not related to bipartite matching.

¹ This research was supported in part by the National Science Council of the Republic of China while the second author visited the Academia Sinica.

¹ Institute of Information Sciences, Academia Sinica, Republic of China

² Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois 60208
An $O(\min[m \cdot n, n^2 \log n])$ Maximum Weight Clique Algorithm
for Circular–Arc Graphs

1. Introduction

A circular arc family $F$ is a collection of arcs on a circle. A graph $G$ is a circular–arc graph if there is a circular arc family $F$ and a one–to–one mapping of the vertices of $G$ and the arcs in $F$ such that two vertices in $G$ are adjacent if and only if their corresponding arcs in $F$ overlap. Circular–arc graphs have applications in compiler design and traffic light sequencing [2]. Various characterization and optimization problems on circular–arc graphs have been studied [2]. In this paper, we consider the maximum weight clique (MWC) problem for circular–arc graphs. A clique in a graph is a set of pairwise adjacent vertices. Let $n$ be the number of vertices in $G$ and $m$ be the number of edges in $G$. An earlier result of Hsu [3] provides an $O(m \cdot n)$ algorithm for this problem (erroneously, this was quoted as $O(n^2 \log n + m \cdot n)$ in [1]). Later, Apostolico and Hambrusch [1] gave an $O(n^2 \log n)$ algorithm for the unweighted case based on bipartite matching. They posed the weighted case as an open problem. In this paper, we extend their technique to produce an $O(\min[m \cdot n, n^2 \log n])$ algorithm for the weighted case. Our algorithm is based on a characterization of maximum weight independent sets in bipartite graphs, which is not related to bipartite matching.

Consider an arc family $F$ and its intersection graph $G$. Without loss of generality, assume all arc endpoints are distinct and no arc covers the entire circle. Now, label the $n$ arcs arbitrarily from 1 through $n$. Label the endpoints from 1 to $2n$ according to their clockwise order. Denote an arc that begins at endpoint $a$ and ends at endpoint $b$ in the clockwise direction by $(a, b)$. Define $a$ to be the head (or counterclockwise endpoint) of the arc and $b$ to be the tail (or clockwise endpoint). An example is shown in Figure 1. The continuous part of the circle which begins
with endpoint c and ends with d in the clockwise direction is referred to as segment \((c,d)\) of the circle. We use "arc" to refer to a member of \(F\) and "segment" to refer to a part of the circle between two endpoints. By this definition, an arc \((a,b)\) of \(F\) is also a segment \((a,b)\).

An arc \(i\) is said to contain another arc \(j\) if both endpoints of \(j\) are contained in arc \(i\). Define a \textit{minimal arc of a clique} \(Q\) to be any arc of \(Q\) which does not contain another arc of \(Q\). For each arc \(u\) of \(F\), we determine a MWC \(Q^u\) among all cliques containing arc \(u\) as a minimal arc. Among these \(n\) cliques, a \(Q^u\) with the maximum weight is a MWC of \(G\).

Hence, we shall now consider a fixed arc \(u = (a,b)\) and search for \(Q^u\). Note that arcs which pass through both \(a\) and \(b\) will naturally be included in \(Q^u\). Therefore, our selection is among those arcs which pass through exactly one endpoint of \(u\). Let \(A\) be the set of those arcs passing through endpoint \(a\) and \(B\) the set of those passing through endpoint \(b\). Arcs in \(A\) (respectively, \(B\)) will be referred to as \(A\)-arcs (respectively, \(B\)-arcs). Obviously, arcs in \(A\) overlap with each other. The same holds for arcs in \(B\). Our algorithm is to optimally select a subset of \(A\) and one of \(B\) such that every two arcs in these subsets overlap with each other. Assume all arcs not in \(A \cup B\) have already been deleted. Starting from \(b\), one can order all endpoints in the interior of segment \((b,a)\) along the clockwise direction into a linear list \(L\). Starting from \(a\), one can order all endpoints in the interior of segment \((a,b)\) along the clockwise direction into a linear list \(R\). List \(L\) contains heads of all \(A\)-arcs and tails of all \(B\)-arcs. List \(R\) contains tails of all \(A\)-arcs and heads of all \(B\)-arcs. An example is shown in Figure 1.

Fig. 1. The two lists \(L\) and \(R\).

Define the \textit{complement} \(G\) of \(G\) to be the graph whose vertex set is the same as \(G\) and whose edges are exactly those missing in \(G\). Since both \(A\) and \(B\) induce complete subgraphs of \(G\), the subgraph \(K\) of \(\overline{G}\) induced on vertices corresponding to
arcs in $A \cup B$ is a bipartite graph. Hence, finding a MWC in $A \cup B$ in $F$ is equivalent to finding a maximum weight independent set (MWIS) of $K$. In the unweighted case, there is a close relationship between a maximum matching and a maximum independent set in a bipartite graph. However, such relationship does not exist in the weighted case. Instead, we resort to the following characterization of MWISs in bipartite graphs.

**Theorem 1.** Let $K = (X \cup Y, E)$ be a bipartite graph with weights assigned to its vertices. Then an independent set $P \subseteq X \cup Y$ is a MWIS in $K$ if and only if both of the following two conditions hold:

1. For each subset $X'$ of $X \setminus P$, the set $Y'$ of vertices in $Y \cap P$ adjacent to some vertices in $X'$ satisfies $w(Y') \geq w(X')$.
2. For each subset $Y''$ of $B \setminus P$, the set $X''$ of vertices in $X \cap P$ adjacent to some vertices in $Y''$ satisfies $w(X'') \geq w(Y'')$.

**Proof:** The "only if" part: Let $P$ be a MWIS in $K$. If there exists a subset $X'$ in $X \setminus P$ such that its corresponding $Y'$ satisfies $w(Y') < w(X')$, then $X' \cup (P \setminus Y')$ is an independent set of $K$ with weight $w(X') + w(P) - w(Y') > w(P)$, a contradiction. Hence, (1.1) holds. Similarly, one can show that (1.2) holds.

The "if" part: Let $P$ be an independent set of $K$ satisfying conditions (1.1) and (1.2). Let $P^*$ be a MWIS of $K$. Let $X'$ be $X \cap (P^* \setminus P)$ and $Y'$ be its corresponding adjacent set in $Y \cap P$. Let $Y''$ be $Y \cap (P^* \setminus P)$ and $X''$ be its corresponding adjacent set in $X \cap P$. Then we have $w(Y') \geq w(X')$, $w(X'') \geq w(Y'')$, $P \cap (X' \cup Y'') = \emptyset$ and $P^* \cap (X'' \cup Y') = \emptyset$. Since $P^*$ is the disjoint union of sets $(P \cap P^*)$, $X'$ and $Y''$ and $P$ contains the following three disjoint sets $(P \cap P^*)$, $Y'$ and $X''$, we have

$$w(P^*) = w(P \cap P^*) + w(X') + w(Y'') \leq w(P \cap P^*) + w(Y') + w(X'') \leq w(P).$$

Hence, $P$ is also a MWIS of $K$. \qed
2. The MWC Algorithm

We use $u$ to denote an arc in $F$, $j$ to denote an arc in $A$, $i$ to denote an arc in $B$ and $Q$ to denote a MWC among all arcs scanned. The idea of the algorithm can be described as follows. We scan the list $L$ from left to right. Every time the tail of a $B$-arc $i$ is encountered, arc $i$ is included in $Q$. Every time the head of an $A$-arc $j$ is encountered, we test whether there is a clique of larger weight containing $j$ and update $Q$. When all endpoints in $L$ are scanned, the current $Q$ is a MWC in $A \cup B$. The testing involves exchanging certain arcs in $Q$ with some arcs not in $Q$, which is the main part of our algorithm.

Denote the original weight of each arc $u$ by $w(u)$. Its modified weight at each iteration is denoted by $w'(u)$. We use a graph $H$ to keep track of the relationships of all arcs scanned so far and modify $H$ by adding vertices and edges (but never delete vertices or edges) along the algorithm. Each vertex of $H$ corresponds to an arc already scanned. Each edge of $H$ connects a vertex representing an $A$-arc to one representing a $B$-arc and is assigned a weight. The edges of $H$ are added so that the entire graph becomes a forest. Each component of $H$ is a tree which is associated with a root. If this root corresponds to an $A$-arc, then the tree is called an $A$-tree; otherwise, it is called a $B$-tree. The union of all $A$-trees is called the $A$-forest of $H$. The union of all $B$-trees is called the $B$-forest. Denote the vertex of $H$ corresponding to an arc $i$ in $F$ by $v_i$. Denote the arc of $F$ corresponding to a vertex $v$ in $H$ by $i_v$. A priority queue $\text{AVAIL}$ is used to store the roots of all the $B$-trees. The algorithm maintains the following conditions for $H$:

(2.1) Let $v$ be a vertex in a tree other than the root. Then $w(i_v)$ is equal to the total weight of edges incident to it.

(2.2) Let $v$ be the root of a tree. Then $w(i_v)$ is no less than the total weight of edges incident to it. Furthermore, if $i_v$ is a $B$-arc, then $w(i_v)$ is strictly greater than the total weight of edges incident to it.
Every A–arc corresponding to a vertex in the A–forest overlaps with every
B–arc corresponding to a vertex in the B–forest.

The clique \( Q \) constructed from arcs scanned so far consists of all A–arcs
corresponding to vertices in the A–forest and all B–arcs corresponding to vertices in
the B–forest. An A–tree, once formed, will never be modified in later iterations.

We now describe the modification of \( H \) at each iteration. If the tail of a B–arc
\( i \) is scanned, then we add a new vertex \( v_i \) representing \( i \) to \( H \) as a (single vertex)
B–tree. If the head of an A–arc \( j \) is scanned, then we add a new vertex \( v_j \) to \( H \)
representing \( j \) and add new edges incident to \( v_j \) according to the two cases below.

Let \( \{v_{i_1}, \ldots, v_{i_r}\} \) be the roots of the B–forest whose corresponding arcs \( i_1, \ldots, i_r \) do
not overlap with \( j \) and let them be arranged according to the R–order.

Case 1. \( w(j) \geq \sum_{t=1}^{r} w'(i_t) \). In this case, we can show that \( j \) is contained in some
MWC among all arcs scanned. Hence, it is included in the current \( Q \). Form an
A–tree \( T \) with root \( v_j \) by adding edges \((v_j,v_{i_t})\), \( t = 1, \ldots, r \) with weight \( w'(i_t) \). Set
\( w'(j) = w(j) - \sum_{t=1}^{r} w'(i_t) \).

Case 2. \( w(j) < \sum_{t=1}^{r} w'(i_t) \). In this case, we can show that \( j \) is not contained in any
MWC among all arcs scanned and our \( Q \) remains unchanged. Let \( i_s \) be the first arc
in the list \( i_1, \ldots, i_r \) such that \( w(j) < \sum_{t=1}^{s} w'(i_t) \). Form a B–tree \( T \) with root \( v_{i_s} \) by
adding edges \((v_j,v_{i_t})\), \( t = 1, \ldots, s-1 \) with weight \( w'(i_t) \) and edge \((v_j,v_{i_s})\) with weight
\( w(j) - \sum_{t=1}^{s-1} w'(i_t) \). Reduce the current weight of \( i_s \) to \( \sum_{t=1}^{s} w'(i_t) - w(j) \).

We describe the above procedure more formally in Figure 2. Let \( g = |A \cup B| \).

Procedure FIND–CLIQUE
INPUT: Two sets \( A, B \) of weighted arcs ordered according to the list \( L \)
OUTPUT: A MWC from \( A \cup B \)

begin
\( w'(u) \leftarrow w(u); \ AVAIL \leftarrow \emptyset; \) (* the priority queue storing all roots of B–trees *)
for \( k = 1 \) to \( g \) do

\( \text{— 5 —} \)
if the k-th arc is a B-arc then do
    add $v_i$ as a single vertex B-tree to $H$;
    insert arc $i$ into AVAL according to its R-order;
else do (* the k-th arc is an A-arc denoted by j *)
    exchange = "true"; add $v_j$ as a new vertex to $H$;
    while exchange is true & there is a B-arc in AVAL not overlapping
    with $j$ do
        $i_s$ ← the B-arc of the smallest R-order in AVAL not overlapping
        with $j$;
        connect $v_j$ to $v_{i_s}$ by an edge and call the new tree $T$;
        if $w'(j) < w'(i_s)$ then do (* do not exchange *)
            $w'(i_s) ← w'(i_s) - w'(j)$; $w(j,i_s) ← w'(j)$;
            exchange = "false"; make $v_{i_s}$ the root of $T$;
        else do (* continue the process *)
            $w'(j) ← w'(i_s) - w'(j)$; $w(j,i_s) ← w'(i_s)$;
            delete $i_s$ from AVAL; make $v_i$ the root of $T$;
        endwhile;
    endif;
endfor;
$Q$ ← all A-arcs corresponding to vertices in the A-forest and all B-arcs
    corresponding to vertices in the B-forest;
end;

Figure 2. The clique construction procedure

3. Correctness of the Algorithm

To facilitate our discussion, denote the graph $H$ formed at iteration $k$ by $H_k$ and the clique by $Q_k$. Let $H_k'$ be the subgraph of $G$ induced on vertices of $H_k$, $k = 1,...,g$. It is easy to verify that $H_k$ is a spanning forest of $H_k'$. Denote the complement of $H_k'$ by $H_k''$

Theorem 2. For $k = 1,...,g$, the set $Q_k$ constructed is a clique of maximum weight among all cliques in $H_k''$.

Proof. We prove this by induction. Assume it is true for $Q_1,$ ..., $Q_{k-1}$ and consider $Q_k$.

First of all, it is easy to verify that the subgraph $H_k$ satisfies properties (2.1) and (2.2). We show that $Q_k$ is a clique in $G$ and, as an independent set in $H_k'$,
satisfies (1.1) and (1.2). We do this by proving the following three claims. Denote the root of a tree T in a subgraph by r(T). Denote the R-order of the arc i corresponding to vertex v₁ of Hₖ by R(v₁).

Claim 1. Let i and i' be two B-arcs so that v₁ and v₁' are contained in two different B-trees Tₛ, Tₛ' of Hₛ, respectively with R(r(Tₛ)) < R(r(Tₛ')). If v₁ and v₁' are also contained in two different B-trees Tₚ, Tₚ' of Hₚ, respectively, with p > s, then R(r(Tₚ)) < R(r(Tₚ')).

Proof. We show this by induction on the subgraph index. Assume this is true for all subgraphs Hₜ with 1 ≤ t < p and consider Hₚ. Let the two trees containing v₁ and v₁' at iteration p-1 be Tₚ₋₁ and Tₚ₋₁', respectively. By induction, R(r(Tₚ₋₁)) < R(r(Tₚ₋₁')). If the p-th arc is a B-arc, then Tₚ = Tₚ₋₁, Tₚ' = Tₚ₋₁' and the claim is true by induction. Hence, assume the p-th arc j* is an A-arc. Let S = {i₁, ..., i₄} be the set of arcs corresponding to roots of the B-forest in Hₚ₋₁ which do not overlap with j* and let them be arranged according to the R-order. Because i and i' are contained in different B-trees at iteration p, v₁* cannot be adjacent to both r(Tₚ₋₁) and r(Tₚ₋₁') in Hₚ₋₁. If v₁* is adjacent to neither r(Tₚ₋₁) nor r(Tₚ₋₁'), then again, Tₚ = Tₚ₋₁, Tₚ' = Tₚ₋₁' and we are done. Otherwise, consider two cases:

Case (a). v₁* is adjacent to r(Tₚ₋₁) but not r(Tₚ₋₁'): then r(Tₚ) = r(Tₚ₋₁) and the arc for r(Tₚ') must be in S. By Case 2 of the algorithm, v₁ and v₁* are contained in the same B-tree with root r(Tₚ) and R(r(Tₚ)) < R(r(Tₚ')).

Case (b). v₁* is adjacent to r(Tₚ₋₁') but not r(Tₚ₋₁): then r(Tₚ) = r(Tₚ₋₁) and R(r(Tₚ')) ≥ R(r(Tₚ₋₁')) > R(r(Tₚ₋₁)) = R(r(Tₚ)).

Claim 2. Qₖ is a clique in G, namely, Hₖ satisfies (2.3).

Proof. This is obvious if the k-th arc is a B-arc i because Qₖ = Qₖ₋₁ ∪ {i} and i overlaps with all A-arcs in Hₖ. Hence, suppose the k-th arc is an A-arc j. Let i₁,
..., $i_r$ be the arcs corresponding to roots of the $B$-forest in $H_{k-1}$ which do not overlap with arc $j$ and let them be arranged according to the $R$-order. Consider the two cases discussed in the construction procedure.

Case 1. $w(j) \geq \sum_{t=1}^{r} w'(i_t)$: In this case, the algorithm connects $v_j$ to each of $i_1, ..., i_r$ and make $v_j$ the root of this new $A$-tree in $H_k$. It also adds to $Q_k$ all $A$-arcs in $T$ and remove all $B$-arcs in $T$ from $Q_{k-1}$. Hence, we need to show that the $A$-arcs corresponding to vertices in $T$ are overlap with all $B$-arcs corresponding to vertices in the $B$-forest of $H_k$. For each $A$-arc $j'$ in $T$, let $S_j' = \{i_1', ..., i_w'\}$ be the set of $B$-arcs whose corresponding vertices appear in $H_k$ which do not overlap with $j'$. All arcs in $S_j'$ must have their tails scanned before $j'$. We show that $S_j'$ is contained in the $A$-forest of $H_k$.

Suppose, to the contrary, there is an arc $i_q'$ of $S_j'$ with $v_{i_q'}$, contained in a $B$-tree of $H_k$. Let $s$ be the iteration in which the head of $j'$ is scanned. Let $T_s$ be the $B$-tree containing $v_{j'}$ and $T_s'$ be the $B$-tree containing $v_{i_q'}$ at the $s$-th iteration. Then $R(r(T_s)) < R(r(T_s'))$. Since $v_{i_p}$ is in the same $B$-tree with $v_{j'}$ in $H_{k-1}$, $v_{i_p}$ and $v_{i_q'}$ must be in different $B$-trees in $H_{k-1}$. Let $T_{k-1}$ and $T_{k-1}'$ be the subtrees in $H_{k-1}$ containing $v_{i_p}$ and $v_{i_q'}$, respectively. By Claim 1, $R(r(T_{k-1})) < R(r(T_{k-1}'))$. This implies that the arc for $r(T_{k-1})$ is among $i_1, ..., i_r$ and would have been made adjacent to $v_{j'}$ at the $k$-th iteration, a contradiction to the assumption that $i_q'$ is in a $B$-tree in $H_k$.

Case 2. $w(j) < \sum_{t=1}^{r} w'(i_t)$: in this case $Q_k = Q_{k-1}$ and the claim is trivially true.

Claim 3. $Q_k$ satisfies (1.1) and (1.2) in $H_k'$.

Proof. Let $A_k$ be the set of vertices in $H_k$ that correspond to $A$-arcs and $B_k$ be those correspond to $B$-arcs. We first show that (1.1) holds for the set $V_k$ of vertices corresponding to arcs in $Q_k$. Let $A'$ be any subset in $A_k \setminus V_k$. Because $Q_k$ contains
all A-arcs corresponding to vertices in the A-forest of H_k, the vertices in A' must be contained in the B-forest. Hence, A' does not contain any root. Let E_i be the set of edges of H_k incident to some vertices in A'. Let B_1 be the subset of vertices in V_k which are adjacent to some vertices in A' through edges in E_i. Let B' be the subset of vertices in V_k which are adjacent to some vertices in A' through edges of H_k'. Then, B_1 \subseteq B'. By property (2.1), w(A') = w(E_i) \leq w(B_1) \leq w(B'). The fact that (1.2) holds for Q_k can be proved similarly. □

4. Conclusions

We claim that Q^u can be found in O(min[m, n^4 log log n]) time for each u. There will be g iterations in the FIND-CLIQUE procedure. Since the graph H_g constructed has O(g) edges, it is easy to see that the major work involved is to process the set AVAIL. Depending on which of m and n log log n is smaller, we can adopt one of the following two strategies:

The first strategy is used in Fig. 1. Following the idea of Van Emde Boas [4], we use a priority queue over the fixed list R to process the active arcs in AVAIL. Our algorithm uses operations insert, successor, delete, each of which can be implemented in O(log log g) time. The total number of such operations is bounded by O(g) and the total time spent in finding Q^u is O(g log log g), which is bounded by O(n log log n).

The second strategy does not require any sophisticated data structure. It results in the same complexity as that in Hsu [2]. However, the approach used in this paper is simpler. For each arc (e,f), order its adjacent arcs according to their first endpoint encountered traversing from f along the clockwise direction. This can be done in O(m) time assuming all endpoints are sorted in the clockwise direction. Initially, set AVAIL to be all arcs in A \cup B, represented by a linked list ordered according to R. At iteration k, update AVAIL to be the set of all A-arcs and the
set of all $B$-arcs that are roots of the $B$-forest of $H_k$. That is, every time a root of a $B$-tree becomes a non-root, we delete it from AVAIL. If the $k$-th arc is a $B$-arc, then AVAIL is not changed. If the $k$-th arc $j$ is an $A$-arc, then root arcs of the $B$-forest in $H_{k-1}$ (whose weights are to be compared against $w'(j)$, one by one) are exactly those in AVAIL but not in the adjacency list of $j$. Since both lists are ordered according to $R$, we can find the first (and subsequent ones, if necessary) such $B$-root by a linear scan of AVAIL and the adjacency list of $j$. Thus, the total time spent in finding $Q^u$ is proportional to $O(m)$.

In the current algorithm, we have not been able to pass the maximum clique information from one arc to another. This appears to be the only approach which could lead to a more efficient algorithm for the maximum clique problem on circular-arc graphs.

References


Arcs in A: 1, 2, 3  
List L: 6, 5, 1, 2, 4, 3  
Arcs in B: 4, 5, 6  
List R: 6, 5, 1, 3, 4, 2  

Figure 1. The two lists L and R