Planar embedding of hamiltonian graphs via efficient bipartation of circle graphs

Christoph Hundack
Forschungsinstitut für Diskrete Mathematik
Universität Bonn

Hermann Stamm-Wilbrandt
Institut für Informatik III
Universität Bonn

June 1994

Abstract

We describe an easy way to check whether a hamiltonian graph of order n with a given hamiltonian cycle is planar. This is done by solving the bipartation problem for the corresponding circle graph. If the graph is planar an embedding is constructed. The algorithm runs in $O(n)$ time and space.

1 Introduction

From an abstract point of view planarity testing in linear time has been solved by the seminal paper of J.E. Hopcroft, R.E. Tarjan [4]. However many technical questions are answered in a rather unsatisfactory manner (compare [1]). Therefore simple planarity testing/embedding algorithms are of interest for their own, even if they cover only restricted classes of graphs (eg [7]). We describe an easy way for planarity testing/embedding of any graph of order

n containing a hamiltonian cycle if this hamiltonian cycle is given. As this testing is equivalent to deciding whether the corresponding circle graph is bipartite we substitute the latter for the former.
The algorithm first constructs a representation of the circle graph wrt the input graph and its hamiltonian cycle. From the circle graph it tries to derive a so-called conflict-graph inheriting all necessary information on crossing of chords. This graph has size $O(n)$ even if the circle graph has size $O(n^2)$. If the conflict-graph cannot be constructed or if the conflict-graph is not bipartite then the circle graph is not bipartite. If the conflict-graph is bipartite then the same applies to the circle graph. Any bipartation of the circle graph then results in a feasible embedding of the input graph. The algorithm requires $O(n)$ time and space.
Conflict-graphs have been introduced in a more general way by G.J. Fisher, O. Wing [3]. Their algorithm for the generation of the conflict-graph, different to the one presented, runs in $O(n^3)$ time requiring $O(n^2)$ space. Similar problems have been solved by the planarity algorithms described in [4], [6]. Bipartation of circle graphs is extended to circle hypergraphs in [5]. In this form it is a major tool for the simple and efficient planarity testing and embedding algorithm to be published in a forthcoming paper.

2 Basic definitions and data structures

2.1 Definitions and lemmas

The terminology used for graphs in this paper follows that of S. Even [2]. Let G be a graph. The set of vertices of G is denoted by $V(G)$, the set of edges of G by $E(G)$. The *order* of G is $|V(G)|$, the *size* of G is $|E(G)|$. For each $v \in V(G)$ denote by $INC[v]$ the incidence list of v. We consider an *embedding* of a planar graph G to be an ordering of each incidence list of G, such that for each $v \in V(G)$ the order of the edges in $INC[v]$ corresponds to a counterclockwise traversal of the edges in a fixed *embedding of G in the plane*. A graph H is called a *(weak) subgraph* of graph G, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Next a lemma from [2] without proof:

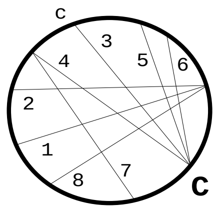
Lemma 1 Let G be a simple planar graph with $|V(G)| > 2$. Then the following holds: $|E(G)| \leq 3|V(G)| - 6$. $\square$

A *chord* of a circle $\mathring{C}$ is a straight line segment connecting two points on $\mathring{C}$. Let CH be a set of (different) chords of circle $\mathring{C}$. The so-called *circle graph* C wrt $\mathring{C}$ and CH is defined by $V(C) = CH$ and $E(C) = \{\, \{ch, ch'\} \in CH \times CH \mid ch$ and ch' intersect (cross) inside $\mathring{C}\,\}$.

Denote by c any endpoint of a chord of CH. The chord $ch \in CH$ possesses a first/second endpoint wrt the counterclockwise order they appear on $\mathring{C}$ beginning with c. Define the *length* of a chord ch to be the length of the arc from ch's first endpoint to its second wrt the counterclockwise order on $\mathring{C}$. Then number the chords arbitrarily by $1, \ldots, |CH|$. Starting with an empty list L traverse the endpoints of the chords on $\mathring{C}$ in counterclockwise order beginning with c. For each endpoint p visited this way collect all chords having p as their second endpoint sorted by increasing length and append them to L; then collect all chords having p as their first endpoint sorted by decreasing length and append them to L, too. After visiting all endpoints list L has size $2|CH|$. Now replace each chord $ch \in CH$ (being numbered by i) by list $\{-i\}$ at ch's first occurrence in L and by $\{i\}$ at its second occurrence. This results in a list of $2|CH|$ single element lists, called a *standard representation* of the circle graph C wrt $\mathring{C}$ and CH.

A standard representation of the circle graph C in the example is given by:

$$L = \{\{-3\}, \{-4\}, \{-7\}, \{-2\}, \{-1\}, \{-8\},$$
$$\{7\}, \{4\}, \{3\}, \{-5\}, \{-6\}, \{8\}, \{1\}, \{2\}, \{6\}, \{5\}\}$$



Note that a standard representation may be viewed as a well-bracketed sequence of different pairs of brackets. Now we examine an important property of standard representations of circle graphs.

Lemma 2 Denote by L a standard representation of the circle graph C wrt circle $\mathring{C}$ and set of chords CH. Denote by $N[ch]$ the number of a chord $ch \in CH$ in L. Now two chords ch and ch' of CH cross (ch being added to L before ch' in the construction above), if and only if, the following pattern appears in L:

$$\{\ldots, \{-N[ch]\}, \ldots, \{-N[ch']\}, \ldots, \{N[ch]\}, \ldots, \{N[ch']\}, \ldots\}.$$

(This is called a *crossing configuration* of ch and ch'.)

Proof: We prove the lemma by examining all possible configurations of chords ch and ch' in C. First look at the case that ch and ch' do not have endpoints in common. If they do not cross then L contains either (a) the pattern

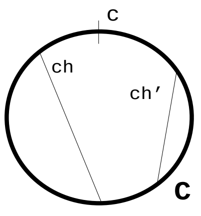
$$\{\ldots,\{-N[ch]\},\ldots,\{N[ch]\},\ldots,\{-N[ch']\},\ldots,\{N[ch']\},\ldots\}$$

or (b) the pattern

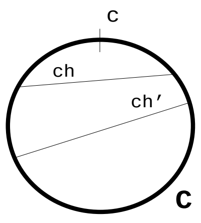
$$\{\ldots,\{-N[ch]\},\ldots,\{-N[ch']\},\ldots,\{N[ch']\},\ldots,\{N[ch]\},\ldots\}.$$

Otherwise (c) L looks like

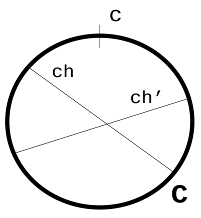
$$\{\ldots,\{-N[ch]\},\ldots,\{-N[ch']\},\ldots,\{N[ch]\},\ldots,\{N[ch']\},\ldots\}.$$



(a)



(b)

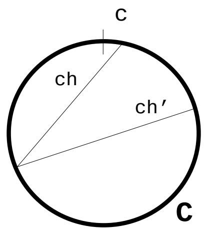


(c)

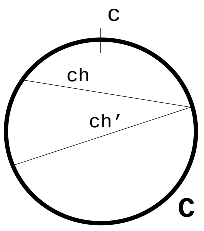
Now if two chords have one endpoint in common, they do not cross due to the definition of a circle graph. We have to show that in this case no crossing configuration is generated during the construction of a standard representation. In case the chords have (d) their first endpoints or (e) their second endpoints in common, the ordering by length prevents a crossing configuration. If (f) the second endpoint of chord ch is the first endpoint of ch' , then L contains the pattern

$$\{\ldots,\{-N[ch]\},\ldots,\{N[ch]\},\ldots,\{-N[ch']\},\ldots,\{N[ch']\},\ldots\}$$

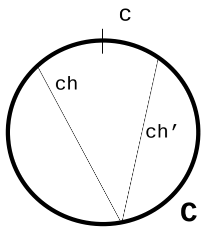
because second endpoints are appended first.



(d)



(e)



(f)

Since all possible cases have been verified the lemma is proven. □

2.2 Data structures

A (doubly linked) *list* L is a sequence of *items*. For each item it of L its *content* is denoted by $L[it]$; $L[it]$ is also called an *element* of L. The number of items in L is called the *size* of L and is denoted by $|L|$. If $L = \epsilon$, ie L has size zero, then it is called the *empty list*. The predecessor of the first item of L and the successor of the last item of L are denoted by $undef$. The type of an element is arbitrary, eg it may be a list itself. For simplicity list $L = it_1, \ldots, it_k$ is also denoted by $\{L[it_1], \ldots, L[it_k]\}$ with $\{\}$ in case $L = \epsilon$. Appending list $Y = it'_1, \ldots, it'_j$ to list $X = it_1, \ldots, it_i$ results in $Y = \epsilon$ and $X = it_1, \ldots, it_i, it'_1, \ldots, it'_j$. This model allows the following constant time operations on list L:

- get the content $L[it]$ of an item it of L;
- get the first/last item of L;
- get the successor/predecessor of a given item in L;
- get the size of L;
- append/delete elements or items to/from L;
- append list L' to L.

Now we describe the implementation of datatype *graph*. We represent the vertices and edges of a graph by positive numbers. Each vertex has information on its *predecessor* and *successor* in the doubly linked list of vertices of the graph. This allows iteration on the vertices as in eg the statement $forall\_vertices(v, G)$. Each vertex v additionally knows about its (doubly linked) incidence list $INC[v]$. Each edge $e = \{u, v\}$ knows about its incident vertices $G\_source(e) = u$ and $G\_target(e) = v$ without the usual directed interpretation. Additionally each edge $e = \{u, v\}$ knows about its positions in $INC[u]$ and $INC[v]$ and its predecessor and successor in the doubly linked list of edges of the graph (eg for iteration purposes as in $forall\_incident\_edges(e, v)$ and in $forall\_edges(e, G)$). The identification of vertices/edges with numbers allows in an easy way to associate information with them by using ordinary arrays[1], here called vertex_array/edge_array. Newly created vertices and edges get the lowest "unused" number available in each case starting with 1. The creation of a new edge e between u and w allows to specify the positions of e in $INC[u]$ and $INC[w]$; the default

[1]This is exactly the way in which *source*, *target*, ... can be realized.

position is at the ends of the lists. The non-existence of a vertex/edge as a result of some function is indicated by value 0. The *cyclic successor/cyclic predecessor* of an edge $e = \{u, v\}$, eg in $INC[v]$, is either given by its successor/predecessor (see above) if this is not 0, or it is given by the first/last edge in $INC[v]$. The operation $G\_move\_edge(e, v, f, dir)$ allows to move edge e of $INC[v]$ to the position before/after f in $INC[v]$ depending on dir. It can be implemented as constant time operation and is the main operation for modifying (planar) graphs to embeddings.

3 Informal description of the algorithm

Let G be the input graph on n vertices and m edges. We number the edges not lying on the hamiltonian cycle HC from 1 to $m - n$. Via two bucket sorts we receive incidence lists sorted descendingly according to the "cyclic length" of the edges. The incidence lists are concatenated wrt their order to one single list. We change the entries of this list in the following way: Replace each entry $\{u, v\} \in E(G) - HC$ numbered by i by $\{-i\}$ at its first appearance in the list and by $\{+i\}$ at its second. This list L is now a standard representation of a circle graph C, whose bipartiteness is equivalent to G being planar.

Now the main procedure of the algorithm (testing whether C is bipartite) is to traverse the generated list and to detect dependencies between the chords (Section 4). During the course of this procedure the so-called *conflict graph* $Conflict$ is built up. In this graph the vertices represent the chords, the edges are the detected conflicts, ie chords of $V(C)$ which have to be in different classes of a bipartation. Non-bipartiteness of C is either tracked down during the main procedure or if $Conflict$ is not bipartite.

If C is not bipartite, then G is not planar. If C is bipartite, then a two-colouring of C is derived from a bipartation of $Conflict$. Maintaining the order given in L the non-cycle edges of G corresponding to chords in one colour class of C are placed on one side of the hamiltonian cycle while the chords corresponding to the second class are placed on the other side. This leads to a feasible embedding of the graph G, thereby proved to be planar.

4 Circle Graph Bipartation

In order to determine a feasible bipartation of the circle graph C we have to know which chords of C cross and therefore must be in different partition classes. Obviously we cannot examine all conflicts as there might be $O(|V(C)|^2)$ of them. Thus we have to find a linear number of conflicts which inherit all partition information on the chords as well as possible non-bipartiteness. For this purpose the standard representation of C is altered by deleting chords and by concatenating sublists. At the same time the conflict graph $Conflict$ is built up. Note that sublists containing more than one element are used to represent that its chords have to be in the same partition class. The necessary information is retrieved by Generate_Conflict_Graph.

```
Generate_Conflict_Graph(list_of_list_of_integer $L$)
     item $actual$,$search$,$aux$; graph $Conflict$;
     initialize $Conflict$ by $E(Conflict) = \emptyset$ and
       $V(Conflict) = \{[-i, i] \mid i \in \{1, \ldots, |L|/2\}\}$;
     let $actual$ denote the first item of $L$;
1    while ($L$ is not empty)
       {
2      while (last entry of $L[actual]$ is negative)
         { set $actual$ to successor of $actual$ in $L$; }
       let $i$ be the single (positive) element of $L[actual]$;
       set $search$ to predecessor of $actual$ in $L$;
3      while (($search \neq undef$) and (last entry $j$ of $L[search] \neq -i$))
         {
         add new edge $\{[-i, i], [-j, j]\}$ to $Conflict$;
         if (successor $aux$ of $search$ in $L \neq actual$)
           { append list $L[aux]$ to $L[search]$; remove $aux$ from $L$; }
             /* now the successor of $search$ in $L$ is $actual$ again */
         set $search$ to predecessor of $search$ in $L$;
         }
4      if ($search == undef$) return ($false, Conflict$);
       set $actual$ to successor of $actual$ in $L$;
       remove list $\{i\}$ from $L$ and (last) element $-i$ from $L[search]$;
       if ($L[search]$ is the empty list) remove $search$ from $L$;
       }
     return ($true, Conflict$);
```

Circle_Graph_Is_Bipartite uses this function in order to decide whether circle graph C given by standard representation L is bipartite.

```
Circle_Graph_Is_Bipartite(list_of_list_of_integer $L$)
    bool $ok$; graph $Conflict$; list_of_chord $P$;
    $(ok, Conflict)$ = Generate_Conflict_Graph($L$);
    if $(ok == false)$ return $(false, \{\})$;
    if ($Conflict$ is not bipartite) return $(false, \{\})$;
    denote by $P \subseteq V(Conflict)$ one partition class
      of a bipartation of $Conflict$;
    return $(true, P)$;
```

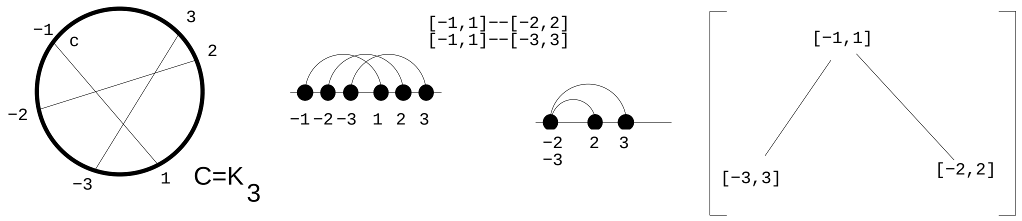
Now we prove the correctness of Circle_Graph_Is_Bipartite.

Lemma 3 If function Generate_Conflict_Graph(L) returns false, then the circle graph C with standard representation L is not bipartite.

Proof: In a standard representation L of C $-i$ is always contained in a list preceding $\{i\}$ for all $i \in \{1, \ldots, |L|/2\}$. This property remains valid during the course of Generate_Conflict_Graph. While searching for the negative number $-i$ corresponding to a positive one i (3), all negative integers, ie all detected conflicts (Lemma 2), are noted by adding an edge to the graph $Conflict$. All lists between the list containing $-i$ and $\{i\}$ are concatenated resulting in one single list. For any feasible bipartation of C the chords $[-j, j]$ corresponding to the elements $-j$ of this list have to be members of the same partition class (not containing $[-i, i]$). No other conflicts between these chords in C are possible without violating the bipartiteness of C.

The only reason for not finding $-i$ (4), is that it is "hidden" in one of the lists preceding $\{i\}$, ie $-i$ is not the last element of its list. Therefore in the list containing $-i$ each $-j$ succeeding $-i$ indicates a conflict between chord $[-j, j]$ and chord $[-i, i]$. This shows that C is not bipartite. $\square$

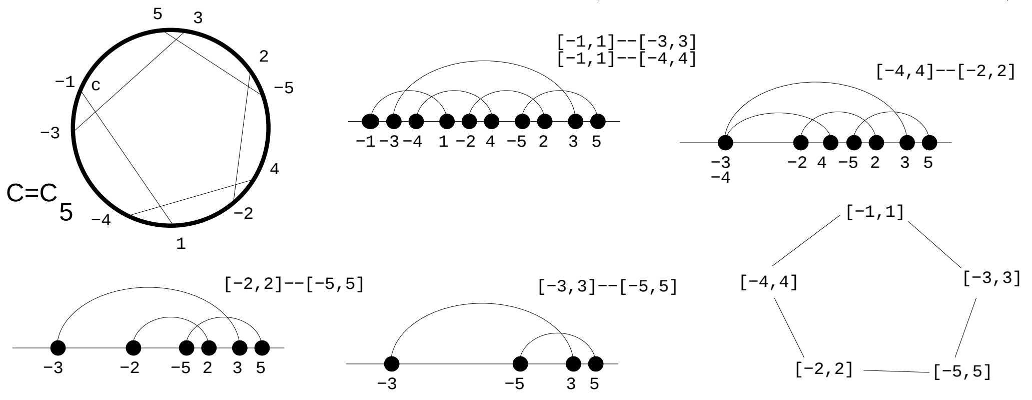
The example illustrates the problem mentioned in the proof above. Non-bipartiteness of C is detected since -2 is "hidden" in $\{-2, -3\}$.



Lemma 4 If the generated graph $Conflict$ is not bipartite then the circle graph C is not bipartite.

Proof: $Conflict$ is isomorphic to a subgraph of C. □

Now an example for a non-bipartite graph C (since $Conflict$ is not bipartite):

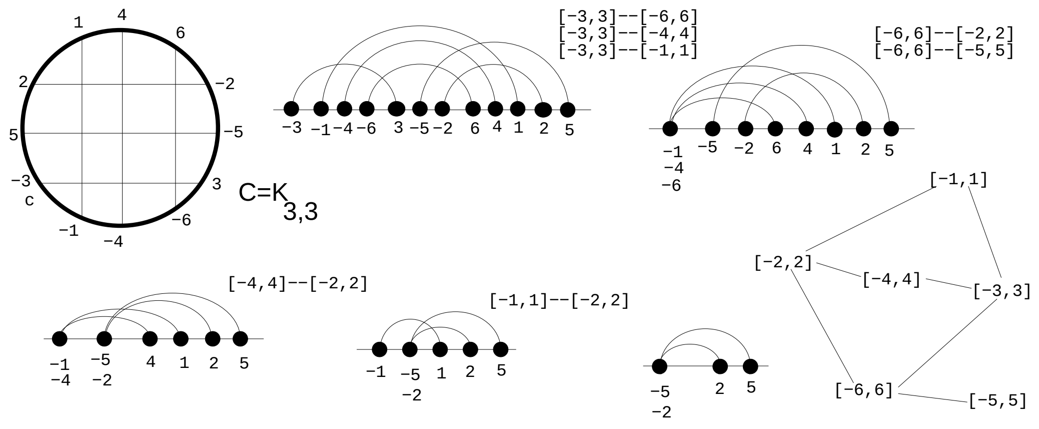


Lemma 5 If $\mathsf{Circle\_Graph\_Is\_Bipartite}(L)$ returns true, then circle graph C with standard representation L is bipartite.

Proof: Colour the chords of circle graph C according to the generated partition of the corresponding vertices of $Conflict$. Assume this two-colouring is not feasible. Then at least two crossing chords have been coloured the same, ie the corresponding vertices in $Conflict$ are not connected by an odd path.

The first time a pair $-i, i$ in Generate_Conflict_Graph is removed an edge in $Conflict$ is inserted between the vertex $[-i, i]$ and the ones representing the negative integers between $-i$ and i. (Therefore the corresponding vertices are pairwise connected by paths of length 2.) Then these integers are joined in one list. Each new removal of a pair $-j$ and j leads to conflict edges between the vertex $[-j, j]$ and all representatives of the last elements of lists in between. These representatives have already been connected by paths of even length to all representatives of preceding list elements. This results in odd paths between the vertex $[-j, j]$ and every vertex representing an element of a list between $-j$ and j. Therefore all conflicts (ie edges) between chords in C are detected and noted either by edges or by odd paths in $Conflict$, contrary to the assumption. □

A $k \times k$ *grid graph* illustrates that the number of edges in $Conflict$ is "linear" although the number of conflicts is "quadratic". The 3×3 grid graph of the example shows a complete run of the algorithm giving a bipartation $\{[-2,2],[-3,3],[-5,5]\} \cup \{[-1,1],[-4,4],[-6,6]\}$.



Corollary 6 Circle_Graph_Is_Bipartite(L) returns true, if and only if, the circle graph C with standard representation L is bipartite. □

Now we show that Generate_Conflict_Graph(L) runs in time linear in the size of L. The order of $Conflict$ is equal to the order of C. Within while-loop (2) every integer is visited once. The total running time of while-loop (3) is linear in the number of edges inserted in $Conflict$ as all operations in while-loop (3) require only constant time and due to the removal of $\{i\}$ and $-i$ no edge is added twice. What remains to be shown is that the number of edges of $Conflict$ is linear in the size of L (and $|V(Conflict)|$).

Lemma 7 $|E(Conflict)| < 2|V(Conflict)|$.

Proof: Let $n = |V(Conflict)|$ be the number of chords of the circle graph. Denote by c_i the number of connected components of $Conflict$ and by e_i the number of edges created during the i-th step of the main loop (1) of Generate_Conflict_Graph. Initially we have $c_0 = n$ and after the completion of Generate_Conflict_Graph $c_n \geq 1$. In each step at most one of the edges created newly connects vertices of the same component. All other edges decrease the number of connected components, each by one. Therefore $e_i \leq c_{i-1} - c_i + 1$ for $1 \leq i \leq n$. Now we bound $e = |E(Conflict)|$ from above by

$$e = \sum_{i=1}^{n} e_i \leq \sum_{i=1}^{n} (c_{i-1} - c_i + 1) = n + c_0 - c_n \leq n + n - 1 < 2n. \quad \Box$$

Corollary 8 Circle_Graph_Is_Bipartite(L) runs in time linear in $|L|$. □

5 Planarity testing/embedding of hamiltonian graphs

5.1 Introduction

For the rest of this section we assume that the hamiltonian input graph G with $V(G) = \{1, 2, ..., n\}$ is given via incidence lists. Its vertices are numbered ascendingly from 1 to n according to their appearances in the given hamiltonian cycle HC.

We also assume that $|E(G)| = O(|V(G)|)$, since this is a necessary condition for the planarity of a graph (Lemma 1). This is checked at the very beginning of function Planar_Hamilton_Embedding to be introduced later.

We consider HC as "circle" $\overset{\circ}{C}$ and the edges of $E(G) - HC$ as set of "chords" CH. Let C be the circle graph wrt to $\overset{\circ}{C}$ and CH. We call C the circle graph wrt to G and HC, too. Denote for $\{u, v\} \in E(G) - HC$ by $d_u^v = (v - u) \bmod |V(G)|$ the *distance* from u to v in G.

The next lemma shows the equivalence of the planarity of G and the bipartiteness of C.

Lemma 9 The graph G with hamiltonian cycle HC is planar, if and only if, the circle graph C wrt G and HC is bipartite.

Proof: **(if)** Consider an embedding of HC being numbered counterclockwise. Since C is bipartite its chords can be partitioned into two crossing-free classes. Denote the corresponding sets of edges by A and B. Now sort all incidence lists of G resulting in

$$INC[v] = \{\{v, 1 + (v \bmod |V(G)|)\}, \{v, u_1\}, \ldots, \{v, u_k\},$$
$$\{v, 1 + ((v-2) \bmod |V(G)|)\}, \{v, w_1\}, \ldots, \{v, w_l\}\}$$

with $\forall i \in \{1, \ldots, k\} : \{v, u_i\} \in A, d_v^{u_i} < d_v^{u_{i+1}}$ and $\forall j \in \{1, \ldots, l\} : \{v, w_j\} \in B, d_v^{w_j} > d_v^{w_{j+1}}$. This results in an embedding of G thus proved to be planar.

(only if) Consider the embedding of G given. The chords corresponding to the edges of $E(G) - HC$ inside HC do not cross. The same applies to the chords corresponding to the edges of $E(G) - HC$ outside HC. Thus both sets of chords are independent sets of C whose union is CH. Therefore C is bipartite. □

5.2 Algorithm

First we describe the procedure Preprocessing, which takes G as input. It returns a standard representation L of the circle graph C wrt G and HC and the mapping M of chords of C to their corresponding edges from $\overline{HC} = E(G) - HC$.

```
Preprocessing(graph $G$ )

     list $L$; edge_array $N$; list_of_edge $\overline{HC}$; array_of_edge $M$;
     $\overline{HC} = \{ e = \{u,v\} \in E(G) \mid ((v-u) \bmod |V(G)|) \notin \{1, |V(G)|-1\} \}$
     number in $N$ the edges of $\overline{HC}$ by $\{1,\ldots,|E(G)|-|V(G)|\}$;
     forall $e \in \overline{HC}$:  $M[N[e]] = e$;
 1   $L = \{(d_u^v, u, -N[e]), (d_v^u, v, N[e]) \mid e = \{u,v\} \in \overline{HC}, u < v\}$
 2   sort $L$ decreasingly using stable bucket_sort in the range
       $[2,\ldots,|V(G)|-2]$ by the tupels first entries;
 3   sort $L$ increasingly using stable bucket_sort in the range
       $[1,\ldots,|V(G)|]$ by the tupels second entries;
     replace each tupel $(d,u,i)$ of $L$ by the the list $\{i\}$ resulting
       in $L$ being a list_of_list_of_integer;
     return $(L, M)$;
```

Lemma 10 The list L returned by function Preprocessing(G) is a standard representation of the circle graph C wrt G and HC. Preprocessing(G) runs in time $O(|V(G)|)$.

Proof: Each number $i \in \{1,\ldots,|E(G)|-|V(G)|\}$ occurs exactly once as $\{-i\}$ and once as $\{i\}$ in L (1), and $|L| = 2(|E(G)|-|V(G)|)$. $\{-i\}$ occurs before $\{i\}$ in L because of (3) and $u < v$ in (1). Consider HC as a circle numbered counterclockwise. The first stable bucket_sort (2) realizes implicitly the order of chords with common endpoint according to their distance. The second stable bucket_sort (3) moves chords with common endpoint to successive positions in L preserving the order of (2) for single vertices. Therefore the generated list L is a standard representation of the circle graph C wrt G and HC (c corresponding to vertex 1).

Since $|L| < 2|E(G)|$ and the mapping range of the two bucket sorts is also linear in $|V(G)|$, Preprocessing(G) runs in time $O(|V(G)|)$ due to the remarks on data structures in Section 2 and the assumption made at the beginning of this section. $\square$

Assume that the circle graph C wrt G and HC is bipartite. We describe the procedure Postprocessing, which modifies G to be an embedding.

```
Postprocessing(graph $G$ , list_of_list_of_integer $L$,
               array_of_edge $M$, list_of_chord $P$)

    array_of_bool $inside$;
    forall $i \in \{1, \ldots, |L|/2\}$:  $inside[i] = true$;
    forall elements $[-i, i]$ of $P$:  $inside[i] = false$;
1   forall $\{j\}$ in $L$
      {
      $e = M[\ |j|\ ] = \{u, w\}$ with $u < w$;
      if $(j < 0)$  $v = u$;  else  $v = w$;
      if $(inside[\,|j|\,] == true)$
        $G\_move\_edge(e, v, \{v, 1 + (v \bmod |V(G)|)\}, after)$
      else
        $G\_move\_edge(e, v, \{v, 1 + (v \bmod |V(G)|)\}, before)$;
      }
```

Lemma 11 Function Postprocessing modifies G to be an embedding. It requires $O(|V(G)|)$ time.

Proof: Within L the edges having one endpoint v in common are sorted decreasingly according to d_v^w. This order is preserved within the list of edges to be moved before the hamiltonian cycle edge $\{v, 1 + (v \bmod |V(G)|)\}$. It is reversed within the list of edges moved after the hamiltonian cycle edge $\{v, 1 + (v \bmod |V(G)|)\}$. The second hamiltonian cycle edge $\{v, 1 + ((v - 2) \bmod |V(G)|)\}$ is not moved, ie its position is between the edge of largest distance placed after $\{v, 1 + (v \bmod |V(G)|)\}$ and the edge of largest distance placed before $\{v, 1 + (v \bmod |V(G)|)\}$. Therefore the order generated within the incidence lists $INC[v]$ is the same as the one in the proof of Lemma 9 (if-part).
The running time of (1) is linear in $|L|/2$, ie in $|V(G)|$: All operations within loop (1) can be done in constant time as stated in Section 2. $\square$

Now we present the main procedure, Planar_Hamilton_Embedding, putting the developed things together. It returns true if G is planar and embeds G in this case. Otherwise it returns false.

```
Planar_Hamilton_Embedding(graph $G$)
    list_of_list_of_integer $L$; array_of_edge $M$; list_of_integer $P$; bool $bip$;
0   if ($|E(G)| > 3|V(G)| - 6$) return $false$;
1   $(L, M)$ = Preprocessing($G$);
2   $(bip, P)$ = Circle_Graph_Is_Bipartite($L$);
3   if ($bip == false$)
      return $false$;
    else
      {
4     Postprocessing($G, L, M, P$);
      return $true$;
      }
```

Corollary 12 Planar_Hamilton_Embedding(G) returns true, if and only if, G is planar. If G is planar then it is modified to be an embedding.

Proof: Lemma 1 shows that aborting with false in (0) is correct. Lemma 10 ensures that Preprocessing(G) returns a standard representation L of the circle graph C wrt G and HC. By Lemma 9 C is bipartite, if and only if, G is planar. Thus aborting with false in (3) is correct. By Lemma 11 G is modified to be an embedding if being planar (4). This completes the proof. □

Lemma 13 Planar_Hamilton_Embedding(G) runs in time and space $O(|V(G)|)$.

Proof: No part of Planar_Hamilton_Embedding or the other functions works on uninitialized data. Therefore showing running time $O(|V(G)|)$ implies the use of only $O(|V(G)|)$ space.

The first test (0) can be done in time $O(|V(G)|)$ by visiting edges one by one and aborting if having seen $3|V(G)| - 5$ different edges. (1), (2) and (4) have $O(|V(G)|)$ running time by Lemma 10, Corollary 8, and Lemma 11. □

References

[1] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.
Algorithms for drawing graphs: an annotated bibliography
November 1993, available via anonymous ftp from wilma.cs.brown.edu, file /pub/gdbiblio.tex.Z

[2] Even, S.
Graph Algorithms
Computer Science Press, Rockville, MD, 1979

[3] Fischer, G.J.,Wing, O.
Computer recognition and extraction of planar graphs from the incidence matrix
IEEE Transactions on circuit theory, **ct-13**(2) 1966, 154-163

[4] Hopcroft, J.E., Tarjan, R.E.
Efficient planarity testing
Journal of the Association for Computing Machinery, **21**(4) 1974, 549-568

[5] Hundack, C., Stamm-Wilbrandt, H.
Efficient bipartation of circle hypergraphs
Report IAI-TR-94-xx, Institut für Informatik III, Universität Bonn, 1994

[6] Mehlhorn, K.
Graph algorithms and NP-completeness, *Data structures and algorithms vol.2*
Springer-Verlag, Berlin, 1984

[7] Stamm-Wilbrandt, H.
A simple linear time algorithm for embedding maximal planar graphs
Report IAI-TR-93-10, Institut für Informatik III, Universität Bonn, 1993