Extended circle graphs I

CHRISTOPH HUNDACK HERMANN STAMM-WILBRANDT
Institut für Informatik III
Universität Bonn

Abstract

A graph C is called an extended circle graph if it is the intersection graph of a finite set of hyperchords of a circle. A hyperchord is defined by the interior of a polygon which is given by a finite, ordered set H of points on a circle. (We assume the selected points on the circle to be numbered consecutively from 1 to n.) The class of extended circle graphs is a generalization of a number of well–known graph classes, eg circle graphs and trapezoid graphs and it is known under various names. We summarize a number of results concerning class inclusions, related graph models, and algorithms for extended circle graphs.

We introduce an efficient description for these graphs which is linear in the input size $I = \sum_{H \in V(C)} |H|$, although C may contain $\Theta(I^2)$ edges. This permits algorithms with running time sublinear in $|V(C)| + |E(C)|$; as an example we present a bipartation algorithm for extended circle graphs.

1 Introduction

A graph C is called an extended circle graph if it is the intersection graph of a finite set of hyperchords of a circle. A hyperchord is defined by the interior of a polygon which is given by a finite, ordered set H of points on a circle. (We assume the selected points on the circle to be numbered consecutively from 1 to n.)

Section 2 contains some basic definitions. Furthermore we introduce an efficient description for extended circle graphs called a standard representation.

The class of extended circle graphs is a generalization of a number of well–known graph classes, eg circle graphs and trapezoid graphs and it is known under various names. In Section 3 we summarize a number of results concerning class inclusions, related graph models, and algorithms for extended circle graphs.

In Section 4 we present the algorithm for generating a standard representation if given an extended circle graph by a list of hyperchords; the algorithm requires running time linear in the input size $I = \sum_{H \in V(C)} |H|$.

This permits algorithms for extended circle graphs with running time sublinear in $|V(C)| + |E(C)|$. In Section 5 we give as an example a bipartation algorithm for extended circle graphs with running time linear in I, although C may contain $\Theta(I^2)$ edges.

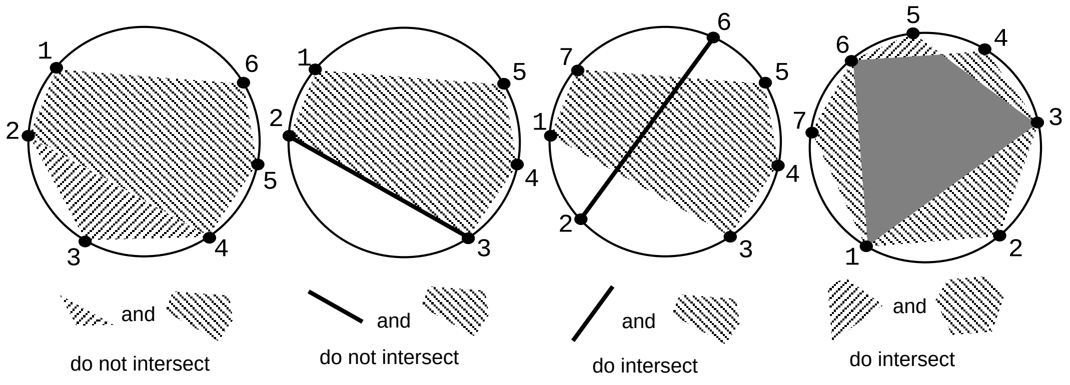
2 Basic definitions and data structures

2.1 Notations

Let G be a graph, $V(G)$ denoting the set of vertices of G and $E(G)$ the set of edges of G. $|V(G)|$ and $|E(G)|$ are called the order and size of G respectively. Let $\bigcirc^n$ be a circle with n specified points $1, \ldots, n$ given in counterclockwise order. Two or more points $p_1, \ldots, p_k \in \{1, \ldots, n\}$ of $\bigcirc^n$ with $p_1 < p_2 < \ldots < p_k$ define a simple closed k–gon P in $\bigcirc^n$. The *interior of a k–gon P* is denoted by $I(P) = I(\{p_1, \ldots, p_k\})$; in case $k == 2$ the interior is all of the straight–line segment $\overline{p_1 p_2}$ leaving out p_1 and p_2. (For technical reasons we define for a point p its interior by $I(\{p\}) = \{p\}$.)

Definition 1 *Let $\{p_1, \ldots, p_k\}$ be an ordered set of points on $\bigcirc^n$. The* hyperchord *$H = \{p_1, \ldots, p_k\}$ is defined by the interior of the corresponding k–gon. Two hyperchords H, H' with $I(H) == I(H')$ are called* intersecting, *if $|H| \geq 3$. Two hyperchords H, H' with $I(H) \neq I(H')$ are called* intersecting, *if $I(H) \cap I(H') \neq \emptyset$.*

Note that different hyperchords may possess common endpoints.



Definition 2 *Let CH be a set of hyperchords wrt $\bigcirc^n$. The graph C wrt $\bigcirc^n$ and CH defined by $V(C) = CH$ and $E(C) = \{\{H, H'\} \mid H, H' \in CH \wedge H\, and\, H'\, intersect\}$ is called an* extended circle graph.

For each hyperchord $H = \{p_1, \ldots, p_k\}$ the endpoints p_i of H with $1 < i < k$ are called *intermediate endpoints* of H; furthermore $first(H) = p_1$, $last(H) = p_k$. The *length* of a hyperchord $H = \{p_1, \ldots, p_k\}$ is given by $length(H) = last(H) - first(H)$. The *size* of a set of hyperchords CH is defined by $size(CH) = \sum_{H \in CH} |H|$; we consider $size(CH)$ as the input size for algorithms dealing with extended circle graphs.
We assume that an extended circle graph C is given by a list_of_list_of_integer L. Each sublist representing a hyperchord is sorted ascendingly in the range $1, \ldots, n$ with n being the number of points on the corresponding circle. Furthermore we assume that $n = O(|L|)$ and thus do not allow too many "unused points".

2.2 Standard representation

We consider a *number* to be an element of $\{-i, \pm i, +i | i \text{ is (positive) integer}\}$. The numbers $-i$, $\pm i$ and $+i$ are called *signed versions* of an integer $i \geq 1$; the *sign* of i is $-$, $\pm$ and $+$ respectively.

Definition 3 *Let S be a list_of_list_of_number and let $\{i\}$ represent any of $\{-i\}, \{\pm i\}, \{+i\}$. If for $i \neq j$ one of the patterns*

$$\{i\}, \ldots, \{j\}, \ldots, \{i\}, \ldots, \{j\} \quad or \quad \{j\}, \ldots, \{i\}, \ldots, \{j\}, \ldots, \{i\}$$

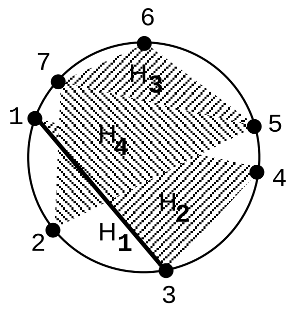
occurs in S this is called a crossing configuration *of i and j in S.*

Definition 4 *Let C be an extended circle graph wrt $\bigcirc^n$ and $CH = \{H_1, \ldots, H_r\}$. A list_of_list_of_number S is called a* standard representation *of C if*

1. *for each hyperchord $H_j \in CH$ and each point $p \in H_j$ we have*
   (a) *if $p == first(H_j)$ this is represented by the list $\{-j\}$ in S;*
   (b) *if p is an intermediate endpoint of H_j this is represented by the list $\{\pm j\}$ in S;*
   (c) *if $p == last(H_j)$ this is represented by the list $\{+j\}$ in S;*
2. *for each hyperchord $H_j \in CH$ we have*
   (a) *the number of the first list $\{j\}$ in S has sign $-$;*
   (b) *the number of the last list $\{j\}$ in S has sign $+$;*
3. *for all $H_i, H_j \in V(C)$ there exists a crossing configuration of i and j in S, if and only if, $\{H_i, H_j\} \in E(C)$.*

A standard representation S of the extended circle graph C with $V(C) = \{\{1,3\}, \{1,3,4\}, \{5,6,7\}, \{2,5,7\}\}$ in the example is given by

$$S = \{\{-2\}, \{-1\}, \{-4\}, \{+1\}, \{\pm 2\}, \{+2\}, \{\pm 4\}, \{-3\}, \{\pm 3\}, \{+3\}, \{+4\}\}$$



Theorem 1 *Every extended circle graph possesses a standard representation.*

Proposition 2 *Let C be an extended circle graph wrt $\bigcirc^n$ and the set of hyperchords $CH = \{H_1, \ldots, H_r\}$. Let $H_i \neq H_j$ be two hyperchords of C and $a, c \in H_i$, $b, d \in H_j$ with $a < b < c < d$. Then $I(H_i) \cap I(H_j) \neq \emptyset$.*

Proof: If $|H_i| == 2$ choose $e = c$, otherwise choose any $e \in H_i \setminus \{a, c\}$. If $|H_j| == 2$ choose $f = d$, otherwise choose any $f \in H_j \setminus \{b, d\}$. Let $p = \overline{ac} \cap \overline{bd}$. $\overline{ac}$ intersects $\overline{bf}$ or $\overline{df}$ in a point called g; $\overline{bd}$ intersects $\overline{ae}$ or $\overline{ce}$ in a point called h. Then $\emptyset \neq I(\{p, g, h\}) \subseteq I(H_i) \cap I(H_j)$. $\square$

Proposition 3 *Let C be an extended circle graph wrt $\bigcirc^n$ and the set of hyperchords $CH = \{H_1, \ldots, H_r\}$. Let $H_i \neq H_j$ be two hyperchords of C with $a = first(H_i) == first(H_j)$, $b = last(H_i) == last(H_j)$ and $|H_i|, |H_j| \geq 3$. Then $I(H_i) \cap I(H_j) \neq \emptyset$.*

Proof: Choose $p \in H_i \setminus \{a, b\}$ and $q \in H_j \setminus \{a, b\}$. Define $e = p$ if $p == q$ or $e = \overline{aq} \cap \overline{bp}$ if $p < q$ or $e = \overline{ap} \cap \overline{bq}$ otherwise. Since $a < p, q < b$ we have $e \notin \overline{ab}$ and therefore $\emptyset \neq I(\{a, e, b\}) \subseteq I(H_i) \cap I(H_j)$. □

Proof (of Theorem 1): Let C be an extended circle graph wrt $\bigcirc^n$ and the set of hyperchords $CH = \{H_1, \ldots, H_r\}$. Define for each $p \in \{1, \ldots, n\}$ the lists

1. $High[p] = \{\{+i_1\}, \{+i_2\}, \ldots, \{+i_k\}\}, k \in \{1, \ldots, r\}$, with
   (a) $p == last(H_{i_j}), 1 \leq j \leq k$;
   (b) $High[p]$ is ordered by
   $$(length(H_{i_j}), -|H_{i_j}|, i_j) \leq_{lex} (length(H_{i_{j'}}), -|H_{i_{j'}}|, i_{j'})$$
   with $1 \leq j < j' \leq k$;
2. $Medium[p] = \{\{\pm i_1\}, \{\pm i_2\}, \ldots, \{\pm i_k\}\}, k \in \{1, \ldots, r\}$, with
   (a) $p \in H_{i_j} \setminus \{first(H_{i_j}), last(H_{i_j})\}, 1 \leq j \leq k$;
   (b) the entries of $Medium[p]$ are in arbitrary order;
3. $Low[p] = \{\{-i_1\}, \{-i_2\}, \ldots, \{-i_k\}\}, k \in \{1, \ldots, r\}$, with
   (a) $p == first(H_{i_j}), 1 \leq j \leq k$;
   (b) $Low[p]$ is ordered by
   $$(length(H_{i_j}), -|H_{i_j}|, i_j) \geq_{lex} (length(H_{i_{j'}}), -|H_{i_{j'}}|, i_{j'})$$
   with $1 \leq j < j' \leq k$.

Concatenating the lists results in

$$S = High[1] \circ Medium[1] \circ Low[1] \circ High[2] \circ \ldots \circ Medium[n] \circ Low[n].$$

Note that by definition

$$High[1] == Medium[1] == Medium[n] == Low[n] == \{\}.$$

S fulfils the requirements $1(a)-(c)$ of Definition 4. Due to the definition of $Low[p]$ via $first()$ and $High[i]$ via $last()$ and their order (for all $p \in \{1,\ldots,n\}$) $2(a)-(b)$ of Definition 4 are also satisfied.

It remains to be shown that the equivalence condition 3 of Definition 4 is also fulfilled. Assume first that there exists a crossing configuration

$$\ldots,\{i\},\ldots,\{j\},\ldots,\{i\},\ldots,\{j\},\ldots$$

in S. We have to show $\{H_i,H_j\} \in E(C)$ which is equivalent to $I(H_i) \cap I(H_j) \neq \emptyset$. We only consider the cases

$$\ldots,\{-i\},\ldots,\{j\},\ldots,\{i\},\ldots,\{+j\},\ldots$$

since the first list $\{i\}$ may be replaced by the leftmost list $\{i\}$ in S, ie $\{-i\}$, and a similar argument holds for $\{+j\}$. We abbreviate the above configuration by $-i,j,i,+j$. We write $a\|b$ to indicate that lists $\{a\}$ and $\{b\}$ belong to different endpoints of $\bigcirc^n$ and $\underline{a,b}$ to indicate that they belong to the same endpoint. By definition of S we have

$$-a\|\pm b, \quad -c\|+d, \quad \pm e\|+f \quad \forall a,b,c,d,e,f.$$

We replace list $\{j\}$ by the leftmost list to the right of $\{-i\}$ and $\{i\}$ by the rightmost list to the left of $\{+j\}$. Thus there remain four possible configurations to examine. In each case we prove that $I(H_i) \cap I(H_j) \neq \emptyset$.

$\boxed{\text{case } -i,-j\|+i,+j\text{:}}$ The assumption $\underline{-i,-j}\|+i,+j$ contradicts $1(b)$ and $3(b)$ as well as the assumption $-i,-j\|\underline{+i,+j}$. Therefore we have $-i\|j\|i\|+j$ in this case and by Proposition 2 $I(H_i) \cap I(H_j) \neq \emptyset$.

$\boxed{\text{case } -i,-j\|\pm i\|+j\text{:}}$ The $+i$ has to exist in S to the right of $+j$ and so we have $-i,-j\|\pm i\|+j,+i$ (*). The case $-i\|-j\|\pm i\|+j,+i$ leads to the first case. The same applies to the case $-i,-j\|\pm i\|+j\|+i$. We are left with $\underline{-i,-j}\|\pm i\|\underline{+j,+i}$. If there exists no $\pm j$ in S this contradicts $1(b)$ and $3(b)$, else by Proposition 3 $I(H_i) \cap I(H_j) \neq \emptyset$.

$\boxed{\text{case: } -i\|\pm j\|+i,+j\text{:}}$ Since there is a $-j$ in S it has to occur before $-i$ and thus we have $-j,-i\|\pm j\|+i,+j$. By renaming i and j we are again at the previous case (*).

$\boxed{\text{case } -i\| \pm j, \pm i\| + j\text{:}}$ S contains $-j$ which precedes $-i$ (and $+i$ which succeeds $+j$). Therefore we have $-j, -i\| \pm j, \pm i\| + j, +i$ and by dropping the two medium entries and renaming i and j we end up in the first case.

Now we assume that there exists no edge between two hyperchords H_i, H_j, ie $I(H_i) \cap I(H_j) = \emptyset$. We have to show that there exists no crossing configuration between i and j in S.

Wlog let $(first(H_i), -last(H_i), |H_i|) \geq_{lex} (first(H_j), -last(H_j), |H_j|)$ and $a = first(H_i)$, $b = last(H_i)$ respectively. For all $p \in H_i \setminus \{a, b\}$ we have $a < p < b$.
(We neglect $(first(H_i), -last(H_i), |H_i|) =_{lex} (first(H_j), -last(H_j), |H_j|)$: In case $|H_i| == |H_j| \geq 3$ by Proposition 3 we have $I(H_i) \cap I(H_j) \neq \emptyset$ contradicting the assumption. In case $|H_i| == |H_j| == 2$ the 2–gons H_i and H_j are identical; $Low[a]$ is sorted increasingly by $(length(H_k), -|H_k|, k)$, $High[b]$ is sorted decreasingly by $(length(H_k), -|H_k|, k)$; therefore there exists no crossing configuration.)

It remains to be shown that for all $q \in H_j$ either $q \leq a$ or $b \leq q$. Assume that there exists a $q' \in H_j : a < q' < b$. Then we know that there also exists either $q'' < a$ or $q'' > b$ with $q'' \in H_j$ due to the selection of H_i and H_j. $\overline{ab}$ and $\overline{q'q''}$ intersect and by Proposition 2 we have $I(H_i) \cap I(H_j) \neq \emptyset$ contradicting the assumption. $\square$

2.3 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 denoted by $|L|$. If $L = \epsilon$ 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 $\{L[it_1], \ldots, L[it_k]\}$ and $L = \epsilon$ by $\{\}$. 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:

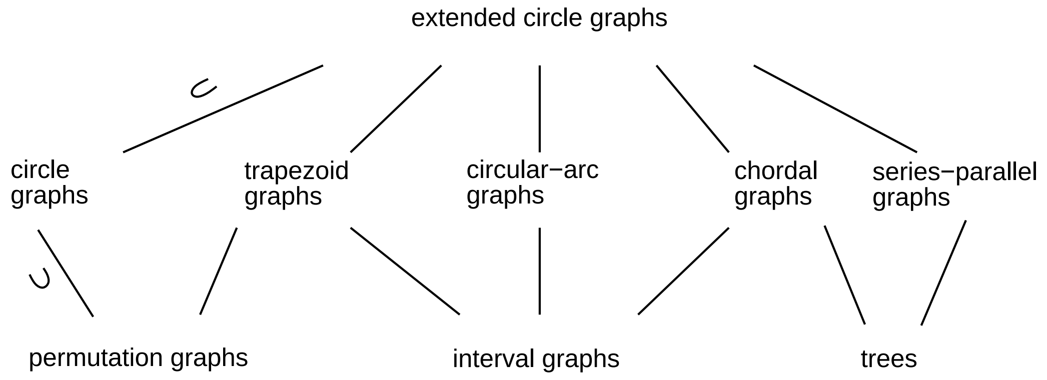
- get the content $L[it]$ of an item *it* of L;
- get the first/last item of L;
- get the successor/predecessor of a given item in L;
- determine $|L|$;
- append/delete elements or items to/from L;
- append list L' to L.

We assume the implementation of datatype *graph* to be standard by using incidence lists and providing eg constant time insertion of new edges. The implementation of the datatype *number* may be realized easily by using 2–tupels, one part representing the sign $(-, \pm, +)$ and the other one the unsigned version of the number.

3 Related results

The most general way to introduce extended circle graphs is via intersection graphs. Each graph can be regarded as an intersection graph wrt a specific model. For extended circle graphs the corresponding model is given by the intersection of the interior of polygons as described in Definition 2.

The class of extended circle graphs is a generalization of a number of well–known graph classes. It contains the class of circle graphs, trapezoid graphs, series–parallel graphs, and circular–arc graphs. M. Koebe [5] observed that it also includes the class of chordal graphs.



Graph models similar to extended circle graphs have been introduced independently several times under different names for a variety of purposes.

Definition 5 [2] *Let G be a two–connected graph and C be a cycle of G. G is called pseudo–hamiltonian wrt C if all components of $G \setminus V(C)$ consist of single vertices.*

G.J. Fisher, O. Wing [2] used *pseudo–hamiltonian graphs* for a planarity testing algorithm with running time $O(n^3)$. For this purpose they developed a bipartation algorithm for pseudo–hamiltonian graphs with running time $O(n^3)$. The bipartation problem for pseudo–hamiltonian graphs is equivalent to the one for extended circle graphs as two components H, H' of a pseudo–hamiltonian graph have to be placed on different sides, if and only if, the interiours of the convex hull of H, H' determined by the corresponding set of vertices intersect. (Each component including its edges and vertices of attachment may be viewed as a hyperchord.)

W.T. Tutte introduced H–fragments (also called H–components or bridges) and found a planarity criterion based on that (compare [8]). Similar concepts concerning cycles with bridges are included in the book by S. Even [1]. H.J. Voss defined overlap graphs in [10], which are (with additional restrictions) similar to extended circle graphs.

M. Koebe [5],[7] has introduced the model of *spider graphs* which is equivalent to the one of extended circle graphs.
As circle graphs are one of the graph classes examined best he has shown extensions of algorithms for circle graphs to spider graphs, for instance for the colouring problem and the computation of a maximal independent set. [6]

includes an equivalence criterion for spider graphs which admits a polynomial time recognition algorithm.

The result of this paper has been proved already for circle graphs in [3]; in that context the bipartation algorithm is used for efficient planarity testing of hamiltonian graphs.
The following table lists a number of known algorithms for extended circle graphs. We assume the extended circle graph C to be given by the list of hyperchords L (compare Section 2).

algorithm/problem	running time	reference
recognition	polynomial	[6]
Maximum Independent Set	$O(size(L)^2)$	[5]
k-Colouring, $k > 3$	NP-complete	[9]
3-Colouring (bounded degree)	polynomial	[5]
bipartation	$O(size(L))$	this paper
determination of connected components	$O(size(L))$	[4]

4 Generating a standard representation

In this section we show the fundamental algorithm for extended circle graphs, ie the generation of a standard representation of an extended circle graph given by a list of hyperchords.

We generate a list which contains for each hyperchord entries for all its endpoints. Each entry representing such an endpoint is a 4-tupel consisting of the number of the endpoint, the length of the hyperchord, the size of the hyperchord, and the number of the hyperchord.

Via four stable bucket we sort this list lexicographically by the four entries. After that for each endpoint i all incident hyperchords appear consecutively in the following order: First the hyperchords whose maximal endpoint is i sorted ascendingly by length, second the hyperchords for which i is an intermediate endpoint, and third the hyperchords whose minimal endpoint is i sorted descendingly by length. These entries are replaced by $\{+h\}, \{\pm h\}, \{-h\}$ respectively (h denoting the number of the corresponding hyperchord) yielding a standard representation S of C equivalent to the one used in the proof of Theorem 1.

```
Generate_Standard_Representation(list_of_list_of_integer $L$)
    {
    integer $i, h, l, p, s, n$; list_of_list_of_number $S$; list_of_integer $H$;
    $h = 0$; $S = \{\}$; $n = \max_{H \in L}\{\, last(H)\,\}$;
    forall $H \in L$ do
       {
1      $h = h + 1$;
2      append $(last(H), length(H), -|H|, h)$ to $S$;
3      append $(first(H), -length(H), |H|, -h)$ to $S$;
       forall intermediate endpoints $i$ of $H$ do
4         append $(i, 0, 0, h)$ to $S$;
       }
5   sort $S$ increasingly using stable bucket sort in the range
      $[-h, \ldots, h]$ by the tupels fourth entries;
6   sort $S$ increasingly using stable bucket sort in the range
      $[-n, \ldots, n]$ by the tupels third entries;
7   sort $S$ increasingly using stable bucket sort in the range
      $[-(n-1), \ldots, n-1]$ by the tupels second entries;
8   sort $S$ increasingly using stable bucket sort in the range
      $[1, \ldots, n]$ by the tupels first entries;
9   replace each tupel $(i, l, p, s)$ of $S$ by the list $\begin{cases} \{-h\} & \text{if} \quad l < 0 \\ \{\pm h\} & \text{if} \quad l == 0 \\ \{+h\} & \text{if} \quad l > 0 \end{cases}$
      with $h = |s|$ resulting in $S$ being a list_of_list_of_number;
    return $S$;
    }
```

Lemma 4 Function Generate_Standard_Representation(L) returns a standard representation S of the extended circle graph C given by the list of hyperchords L.

Proof: In a standard representation S of C the list $\{-i\}$ occurs before list $\{+i\}$ in S for all hyperchords H_i of C. If there are lists $\{\pm i\}$ they occur between $\{-i\}$ and $\{+i\}$ in S (Definition 4). This order is generated by step (2–4) and the fourth stable bucket sort (8). The definition of the entries 2,3,4 of the 4–tupels in step (2–4) together with the three stable bucket sorts in step (5–7) ensures the correct order of all lists containing endpoint p for all $p \in \{1, \dots, n\}$ (compare Definition 4). Thus by the stability of the bucket sorts in step (5–8) we receive the standard representation used in proof of Theorem 1. $\square$

5 Extended Circle Graph Bipartation

The algorithm decides whether an extended circle graph C given by a list of hyperchords L is bipartite. In order to determine a feasible bipartation of C we have to know which hyperchords of C intersect and therefore must be in different partition classes. Obviously we cannot examine all intersections as there might be $\Theta(size(L)^2)$ of them. Thus we have to find a number of intersections linear in $size(L)$, the input size, which inherit all partition information on the hyperchords as well as possible non-bipartiteness.

First we generate a standard representation S of C as shown in the previous section. Now the main procedure of the algorithm is to traverse the standard representation and to detect dependencies between the hyperchords. Each dependency found leads to a modification of S, ie deleting numbers and concatenating sublists. A sublist containing more than one element is used to represent the fact that its hyperchords have to be in the same partition class. During the course of this procedure the conflict graph $Conflict$ is built up. In this graph the vertices represent the hyperchords, the edges correspond to detected conflicts, ie hyperchords which have to be in different classes of a bipartation. Non-bipartiteness of C is either tracked down during the main procedure, ie if $Conflict$ cannot be created, or if $Conflict$ is not bipartite. (Bipartiteness can be checked in time linear in $O(|V(Conflict)|) + O(|E(Conflict)|)$.)

The running time of the algorithm is proportional to the number of edges in $Conflict$, which is bounded by $size(L)$.

Generate_Conflict_Graph tries to generate a conflict graph $Conflict$ from the standard representation S of the extended circle graph C given by L. It returns either $true$ and $Conflict$ or $false$.

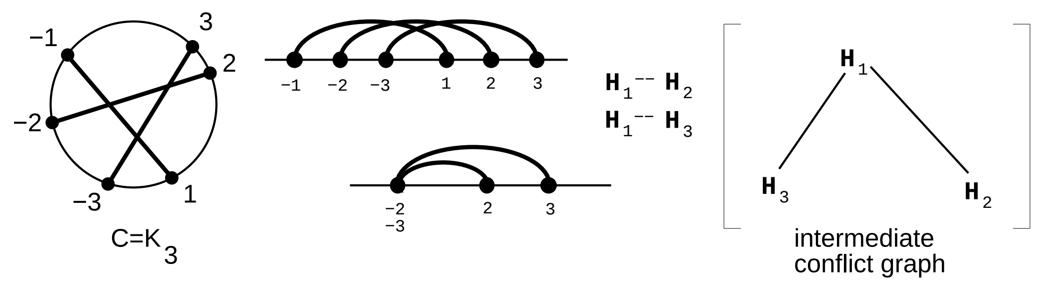
```
Generate_Conflict_Graph(list_of_list_of_number $S$, list_of_list_of_integer $L$)
    {
    item $actual$,$search$,$aux$; graph $Conflict$;
    initialize $Conflict$ by $E(Conflict) = \emptyset$ and
      $V(Conflict) = \{ H_i \mid i \in \{1, \ldots, |L|\} \}$;
    let $actual$ denote the first item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

Lemma 5 If function Generate_Conflict_Graph(S,L) returns false, then the extended circle graph C given by list of hyperchords L with standard representation S is not bipartite.

Proof: In a standard representation S of C given by L $-i$ is always contained in the first list containing a signed version of i; this applies to all hyperchords H_i of C. This property remains valid during the course of Generate_Conflict_Graph. While searching for a number with sign $-$ $(-i)$ corresponding to one with sign $\pm$ or $+$ (i) in step (3), all last entries of lists in between, ie the detected conflicts (Definition 4), are noted. This is done by adding an edge between H_i and the corresponding hyperchord to the graph *Conflict*. All lists between the list containing $-i$ and $\{+i\}$ or $\{\pm i\}$ are concatenated resulting in one single list maintaining their relative order in S. For any feasible bipartation of C the hyperchords H_j corresponding to the elements $-j$ of this list have to be members of the same partition class (not containing H_i). No other conflicts between these hyperchords in C are possible without violating the bipartiteness of C.

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

The example illustrates the stopping criterion of step (4). Non-bipartiteness of C is detected since -2 is "hidden" in $\{-2,-3\}$. The list $\{\{-2,-3\},\{+2\},\{+3\}\}$ is obtained from $\{\{-1\},\{-2\},\{-3\},\{+1\},\{+2\},\{+3\}\}$, the standard representation of C, after the first step of the algorithm.



Extended_Circle_Graph_Is_Bipartite uses Generate_Standard_Representation and Generate_Conflict_Graph in order to decide whether the extended circle graph C given by the list of hyperchords L is bipartite.

```
Extended_Circle_Graph_Is_Bipartite(list_of_list_of_integer $L$)
    {
    list_of_list_of_number $S$; bool $ok$; graph $Conflict$;
    $S$ = Generate_Standard_Representation($L$);
    $(ok, Conflict)$ = Generate_Conflict_Graph($S, L$);
    if ($ok == false$) return $false$;
    if ($Conflict$ is not bipartite) return $false$;
    return $true$;
    }
```

Now we prove the correctness of Extended_Circle_Graph_Is_Bipartite.

Lemma 6 If Generate_Conflict_Graph(S, L) returns true and the generated graph $Conflict$ is not bipartite then the extended circle graph C given by L is not bipartite.

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

Lemma 7 If Extended_Circle_Graph_Is_Bipartite(L) returns true, then the extended circle graph C given by L is bipartite.

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

The first time we deal with a pair $-i, \pm i$ or $-i, +i$ in Generate_Conflict_Graph an edge in $Conflict$ is inserted between the vertex H_i and the ones representing the single elements with negative sign of lists between $-i$ and $\pm i$ or $+i$ in S. (The latter are thereby pairwise connected by paths of length 2.) Then these lists of numbers are concatenated resulting in one list. Each new removal of $\pm j$ or $+j$ (together with $-j$) leads to conflict edges between the vertex H_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 paths of odd length between the vertex H_j and every vertex representing an element of a list between $-j$ and $\pm j$, $+j$ respectively. Therefore all conflicts (ie edges) between hyperchords in C are detected and noted either by edges or by paths of odd length in $Conflict$, in contradiction to the assumption. $\square$

Corollary 8 Extended_Circle_Graph_Is_Bipartite(L) returns true, if and only if, the extended circle graph C given by list of hyperchords L is bipartite.

Proof: This is a simple consequence of Lemmas 5, 6 and 7. $\square$

Now we show that Generate_Conflict_Graph(S, L) runs in time linear in $size(L)$. The order of $Conflict$ is equal to the order of C. Within while-loop (2) every number 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 $\{\pm i\}$ or $\{+i\}$ together with $-i$ each number i with sign $\pm$ or $+$ contributes to the edges of $Conflict$ at most once. What remains to be shown is that the number of edges of $Conflict$ is linear in $size(L) = |S|$ with S being the standard representation of C.

Lemma 9 $|E(Conflict)| \leq size(L)$.

Proof: Let $k = |S| - |L|$ denote the number of non-negative entries in S, ie the number of runs of the main loop (1). Denote by c_i the number of sublists in S and by e_i the number of edges created during the ith step of the main loop (1) of Generate_Conflict_Graph. Initially we have $c_0 = |S|$ and after the completion of Generate_Conflict_Graph $c_k = 0$. The number of edges inserted in $Conflict$ at the ith step is at most the number of sublists between the active pair of numbers (3). After step (3) these sublists are concatenated to one list. The single element list containing the non–negative element of the active pair of numbers is removed in step (5). Therefore $e_i \leq c_{i-1} - c_i$ for $1 \leq i \leq k$. Now we bound $e = |E(Conflict)|$ from above by

$$e = \sum_{i=1}^{k} e_i \leq \sum_{i=1}^{k} (c_{i-1} - c_i) = c_0 - c_k = |S| = size(L). \quad \square$$

Corollary 10 Extended_Circle_Graph_Is_Bipartite(L) runs in time linear in $size(L)$. $\square$

References

[1] Even, S.: *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979;

[2] 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;

[3] Hundack, C., Stamm-Wilbrandt, H.: Planar embedding of hamiltonian graphs via efficient bipartation of circle graphs, Report IAI-TR-94-2, Institut für Informatik III, Universität Bonn, 1994;

[4] Hundack, C., Stamm-Wilbrandt, H.: Extended circle graphs II, in preparation;

[5] Koebe, M.: Colouring of spider graphs, in: Bodendiek, R., Henn, R. (eds.) *Topics in Combinatorics and Graph Theory*, Physica, Heidelberg, 1990, 435–441;

[6] Koebe, M.: Spider graphs – a new class of intersection graphs, unpublished manuscript;

[7] Koebe, M.: On a new class of intersection graphs, in: Nešetřil, J., Fiedler, M. (eds.) *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity*, Elsevier, Amsterdam, 1992, 141–143;

[8] Tutte, W.T.: Bridges and hamiltonian circuits in planar graphs, *Aequationes Mathematicae* **15**, 1987, 1–33;

[9] Unger, W.: On the k–colouring of circle–graphs, in: Cori, R., Wirsing, M. (eds.) *STACS 88*, LNCS 294, Springer, Heidelberg, 1988, 61-72;

[10] Voss, H.J.: *Cycles and bridges in graphs*, Kluwer Academic Publishers, Dordrecht, 1991.