Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®
Discrete Applied Mathematics 138 (2004) 371–379
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs

Derek G. Corneil

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

Received 17 September 2002; received in revised form 22 July 2003; accepted 24 July 2003

Abstract

We present a simple linear time algorithm for unit interval graph recognition. This algorithm uses 3 LBFS sweeps and then a very simple test to determine if the given graph is a unit interval graph. It is argued that this algorithm is the most easily implementable unit interval graph recognition algorithm known.

Keywords: Unit interval graphs; Proper interval graphs; Graph recognition algorithm; Lexicographic breadth first search

1. Introduction

One of the most elegant algorithms in graph theory is the chordal (a graph with no induced cycle of size greater than three) graph recognition algorithm by Rose et al. [20]. This paper introduced the notion of Lexicographic Breadth First Search (LBFS), a modification of Breadth First Search (BFS) where ties are broken by favouring vertices with the earliest visited neighbours. As in Korte and Möhring [11], we say that x and y *disagree* on z if exactly one of x, y is adjacent to z. LBFS produces an ordering $(v_1, v_2, \ldots, v_n)$ of V such that if there are $1 \leqslant i < j < k \leqslant n$ such that v_j, v_k disagree on v_i, then the leftmost vertex on which they disagree is adjacent to v_j. The algorithmic details are presented in Section 2. Rose et al. showed that a graph is

E-mail address: dgc@cs.toronto.edu (D.G. Corneil).

0166-218X/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2003.07.001

chordal if and only if the ordering of V produced by any LBFS is a *Perfect Elimination Ordering*, namely an ordering $v_1, v_2, \ldots, v_n$ of V such that for all i, $1 < i \leqslant n$, $(N(v_i) \cap \{v_j \mid j \leqslant i\})$ is a clique. Recently considerable success has been achieved in developing easily implementable linear time multi-sweep LBFS algorithms to recognize various families of graphs such as interval graphs [4], cographs [1] and bipartite permutation graphs [2]. Our paper continues this approach by presenting a simple 3-sweep LBFS algorithm to recognize unit interval graphs.

A *unit interval graph* (*uig*) is a graph whose vertices can be put in one-to-one correspondence with unit length intervals of the real line such that two vertices are adjacent if and only if the corresponding intervals intersect. Roberts [18,19] has shown that *uigs* are equivalent to *proper interval graphs* (no interval may properly contain another interval) and *indifference graphs* (there exists a positive number δ and an assignment of reals $f(x)$ to the vertices in V such that $xy \in E$ whenever $|f(x) - f(y)| \leqslant \delta$). These equivalences as well as others are captured in the following theorem.

Theorem 1 (Roberts [18,19]). *The following are equivalent*:

1. $G(V,E)$ *is a unit interval graph.*
2. $G(V,E)$ *is a proper interval graph.*
3. $G(V,E)$ *is an indifference graph.*
4. $G(V,E)$ *is an interval graph with no induced* $K_{1,3}$.
5. *There is an ordering of* V *such that for all* $v \in V$, $N[v]$ (*the closed neighbourhood of* v) *is consecutive* ("*the neighbourhood condition*").
6. *There is an ordering of* V *such that vertices contained in the same maximal clique are consecutive* ("*the clique condition*").

Subsequently, Looges and Olariu [14] presented another equivalent ordering condition.

Theorem 2 (Looges and Olariu [14]). $G(V,E)$ *is a uig if and only if there is an ordering of* V *such that for all* $x < y < z$, $xz \in E \Rightarrow xy, yz \in E$ ("*the* 3*-vertex condition*").

Note that the "3-vertex condition", the "neighbourhood condition" and the "clique condition" are equivalent in the sense that any ordering of V that satisfies one of the conditions also satisfies the other two. Thus, for convenience, we will often switch amongst them. For example, the proof of correctness of our algorithm will use the "3-vertex condition" whereas the implementation will use the "neighbourhood condition".

The first linear time recognition algorithm of *uigs* seems to be that of Looges and Olariu [14]. This algorithm first recognizes whether the given graph is an interval graph and then proceeds to determine if it is a unit interval graph. Since it is based on interval graph recognition, this algorithm is not simple. Deng et al. [5] presented the first direct *uig* recognition algorithm, direct in the sense that they did not first recognize

whether the graph is an interval graph. This algorithm is based on local tournaments and is more complicated than two subsequent algorithms. The first, by Corneil et al. [3], uses BFS to find a "left-anchor" and then starts another BFS from this vertex. The vertices in each BFS layer are then ordered by nondecreasing order of the number of adjacencies to the following BFS layer minus the number of adjacencies to the previous BFS layer. Bucket sort is used to provide this ordering. It is then determined if the overall ordering of V satisfies the "neighbourhood condition". The second algorithm, by de Figueiredo et al. [6] does a single LBFS and then uses this order to construct an ordering of V that satisfies the "clique condition".

Hell et al. [10] have studied the problem of recognizing and representing a dynamically changing *uig*. Now, the input is a series of modifications (either addition or deletion of vertices or edges) to be performed on a graph and the goal is to maintain a representation of the graph resulting from each modification (if it is a *uig*) or to detect that it is no longer a *uig*. Their algorithm requires $O(d + \log n)$ steps per modification, where d denotes the number of edges involved in the modification. (Thus if the modification involves a vertex, d is the degree of the vertex; if the modification involves an edge, $d = 1$.)

Our algorithm uses three LBFS sweeps and then, as in [3], tests to see if this order satisfies the "neighbourhood condition". The first sweep is an arbitrary LBFS to find a "left-anchor". The following two LBFS sweeps, with a specific tie-breaking rule, provide an ordering of V that satisfies the "neighbourhood condition" if and only if G is a unit interval graph. As will be seen later, given a particular implementation of LBFS, one immediately has an implementation of the tie-breaking LBFS. Thus, the new algorithm is considerably simpler than the two previous algorithms by avoiding the need to sort the vertices of each BFS layer by degree differences (as in [3]) and by avoiding the construction of the ordering of V satisfying the "clique condition" (as in [6]).

Before presenting the algorithm we introduce some notation and definitions. For a vertex $v \in V, N(v)$ (the open neighbourhood) denotes the set of vertices adjacent to v. $N[v]$ (the closed neighbourhood) denotes $N(v) \cup \{v\}$. Throughout the paper, for S a subset of vertices of a graph $G(V,E)$, S will denote both the subset as well as $G[S]$, the graph induced on S; the meaning will be clear from the context.

2. The algorithm

We first describe the two versions of LBFS that will be used in the *uig* recognition algorithm. The first allows arbitrary tie-breaking and the second uses the ordering from a previous LBFS to break ties.

Let $G(V,E)$ be a connected graph and let u be a vertex of G. We now reproduce the details of the "generic" LBFS [20]. We warn the reader that our LBFS ordering of the vertices of the graph may seem "backwards" compared to the ordering produced by other LBFS descriptions.

Procedure LBFS(G, u);
{Input: a connected graph $G(V,E)$ and a distinguished vertex u of G;
Output: a numbering σ_u of the vertices of G}
begin
label(u) $\leftarrow |V|$;
for each vertex v in $V - \{u\}$ **do**
label(v) $\leftarrow \Lambda$ (the empty list);
for $i \leftarrow |V|$ **downto** 1 **do begin**
pick an unnumbered vertex v with lexicographically the largest label; (★)
$\sigma_u(v) \leftarrow |V| + 1 - i$; {assign to v number $|V| + 1 - i$}
for each unnumbered vertex w in $N(v)$ **do**
append i to label(w)
end
end; {LBFS}

In an LBFS σ with two arbitrary vertices x and y, if vertex x is visited before y, i.e. $x <_\sigma y$ we say that x *occurs* before y in σ or that x is *visited before* y or that x is *to the left of* y. As mentioned above, this generic LBFS algorithm allows arbitrary choice of a vertex in step (★). We call the set of tied vertices encountered in step (★) a *slice* and denote it by S. Unless explicitly stated otherwise we also let V itself be considered as a slice, i.e. all vertices of G are considered tied before the first vertex is chosen.

We now describe a variant of LBFS that is used in the multisweep LBFS recognition algorithm for interval graphs [4]. In particular, it uses the ordering produced by a previous LBFS to break ties in step (★). This variant has been independently investigated by Simon [21] and Ma [15].

Procedure LBFS+ (G, τ):
For this LBFS procedure, one previous LBFS ordering τ is needed. In the LBFS procedure at step (★), let S be the set of vertices with the lexicographically largest label. Now v is chosen to be the vertex in S that appears *last* in τ.

We now show how these two versions of LBFS are used to recognize unit interval graphs.

The *uig* recognition algorithm
{Input: a connected graph $G(V,E)$;
Output: A statement declaring whether or not G is a *uig*}

1. Do an arbitrary LBFS σ.
2. LBFS+ (G, σ) yielding sweep σ^+.
3. LBFS+ (G, σ^+) yielding sweep σ^{++}.
4. If σ^{++} satisfies the "neighbourhood condition", then conclude that G is a unit interval graph; else, conclude that G is not a unit interval graph.

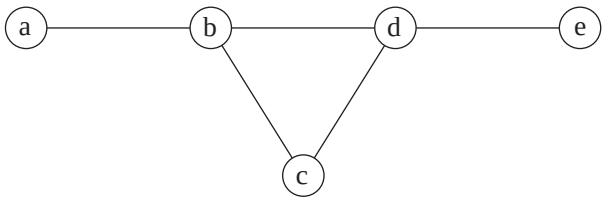

Fig. 1. The bull.

To illustrate various aspects of this algorithm consider the "bull" presented in Fig. 1 and the following three LBFS sweeps performed by the algorithm:

σ: c [b d] a e,

σ^+: e d [b c] a,

σ^{++}: a b [c d] e.

Note that the parentheses indicate the nontrivial slices in each sweep. When applied to σ, LBFS+ first chooses e, the last vertex in σ. Once the slice $S = \{b, c\}$ is encountered, b is chosen first since it is the last S vertex in σ. It is easy to see that the σ^{++} ordering of V satisfies the "neighbourhood condition" and thus we conclude that G is a *uig*. Note that this example shows that the third sweep is required. The ordering in σ^+ does not satisfy the "neighbourhood condition" since $ab \in E$ but $ac \notin E$ thereby showing that $N[a]$ is not consecutive.

As shown in [3], for G a *uig*, it is easy to use the ordering of the vertices in σ^{++} to produce a representation of G by unit intervals in which every endpoint is rational, with its denominator not exceeding n.

3. Correctness and linear time implementation of the algorithm

First, we state an important theorem about the role of LBFS on chordal graphs.

Theorem 3 (Rose et al. [20], The P_3 rule). *Let τ be an LBFS of a chordal graph G and let u, v, w be vertices of G with v adjacent to u and w and such that $u <_\tau v$ and $w <_\tau v$. Then vertices u and w must be adjacent.*

To simplify our notation we let $<_+$ denote $<_{\sigma^+}$ and $<_{++}$ denote $<_{\sigma^{++}}$. In preparation for the proof of correctness of the algorithm, we state a number of facts about *uigs*; all of these are easy to prove and most are well known. The first holds for interval graphs and is proved in [4]. The proof for unit interval graphs is easier and follows from the manner in which LBFS chooses vertices as well as Facts 5 and 8 mentioned below.

Fact 4 (Corneil et al. [4]). *Suppose σ^+ starts with vertex u and ends with vertex v. Then σ^{++} starts with v and ends with u.*

Since LBFS is a specific form of BFS, we will refer to the BFS layers of both σ^+ and σ^{++}. In particular, we let L_i^+ (respectively, L_i^{++}) denote the ith BFS layer of σ^+

(respectively, σ^{++}) where $L_0^+ = \{u\}$ and $L_0^{++} = \{v\}$. Suppose $dist(u,v) = k$; we let P denote the set of vertices that are on a shortest u,v-path.

Fact 5 (Corneil et al. [3]). *Each of* L_i^+, L_i^{++}, $0 \leqslant i \leqslant k$ *is a clique.*

Fact 6.

$$x \in L_i^{++},\ 0 \leqslant i \leqslant k \Rightarrow x \in \begin{cases} L_{k-i}^+ & \text{if } x \in P, \\ L_{k-i+1}^+ & \text{if } x \notin P. \end{cases}$$

Proof. This follows from a straightforward induction argument. □

The next fact captures the "neighbourhood nesting" of vertices in a BFS layer in σ^{++}; a similar result holds for σ^+ but is not needed in our proof of correctness.

Fact 7 (Corneil et al. [3]). $\forall x, y \in L_i^{++}$, $0 < i < k$,
1. $(L_{i-1}^{++} \cap N(x)) \subseteq (L_{i-1}^{++} \cap N(y))$ *or* $(L_{i-1}^{++} \cap N(y)) \subseteq (L_{i-1}^{++} \cap N(x))$,
2. $(L_{i+1}^{++} \cap N(x)) \subseteq (L_{i+1}^{++} \cap N(y))$ *or* $(L_{i+1}^{++} \cap N(y)) \subseteq (L_{i+1}^{++} \cap N(x))$,
3. $(L_{i-1}^{++} \cap N(x)) \subset (L_{i-1}^{++} \cap N(y)) \Rightarrow (L_{i+1}^{++} \cap N(x)) \supseteq (L_{i+1}^{++} \cap N(y))$,
4. $(L_{i+1}^{++} \cap N(x)) \subset (L_{i+1}^{++} \cap N(y)) \Rightarrow (L_{i-1}^{++} \cap N(x)) \supseteq (L_{i-1}^{++} \cap N(y))$.

Fact 8. *If* $x, y \in S \subset V$, *a nontrivial slice of* σ^{++}, *then* $x <_{++} y$ *if and only if* $y <_{+} x$.

Proof. Clearly, any nontrivial slice of σ^{++} must be a subset of L_i^{++}, $(0 < i \leqslant k)$ and thus by Fact 5 is a clique. Since S is a clique, once a vertex is chosen by σ^{++} all remaining vertices in S get its label and remain lexicographically tied. Thus in σ^{++}, the ordering of the vertices of S is the reversal of their ordering in σ^+. □

Theorem 9. *If* G *is a uig, then* σ^{++} *satisfies the* "3-*vertex condition*".

Proof. Suppose not. In particular, suppose $x <_{++} y <_{++} z$, where $xz \in E$ and at least one of $xy, yz \notin E$. Without loss of generality assume z is the rightmost such vertex in σ^{++} with respect to x and y. Suppose $x \in L_i^{++}$. Since σ^{++} is a BFS, if $z \in L_i^{++}$ then since L_i^{++} is a clique (Fact 5), $\{x, y, z\}$ forms a triangle. Thus $z \in L_{i+1}^{++}$. If $y \in L_{i+1}^{++}$, then $yz \in E$ and $xy \notin E$ contradicting the P_3 Rule. Thus $y \in L_i^{++}$, $xy \in E$ and $yz \notin E$. Since $(L_{i+1}^{++} \cap N(y)) \subset (L_{i+1}^{++} \cap N(x))$, by Fact 7 $(L_{i-1}^{++} \cap N(y)) \supseteq (L_{i-1}^{++} \cap N(x))$. But since $x <_{++} y$, LBFS forces $(L_{i-1}^{++} \cap N(x)) \supseteq (L_{i-1}^{++} \cap N(y))$ implying $(L_{i-1}^{++} \cap N(x)) = (L_{i-1}^{++} \cap N(y))$. Thus there is a slice $S \subseteq L_i^{++}$ containing x and y.

Now look at σ^+. By Fact 8, since $x <_{++} y$, $y <_{+} x$. Now $x <_{+} z$, otherwise we have contradicted the P_3 Rule in σ^+. Since σ^+ is a BFS of G, for x to be before z in σ^+, by Fact 6, $x \in L_i^{++} \cap P$ and $z \in L_{i+1}^{++} \setminus P$ (i.e. both x and z are in L_{k-i}^+). Since $x \in P$, there exists $z' \in L_{i+1}^{++} \cap P$ such that $xz' \in E$. We see that $yz' \notin E$ since otherwise $y \in L_i^{++} \cap P$ which by Fact 6 implies $y \in L_{k-i}^+$; but since L_{k-i}^+ is a clique, this implies that $yz \in E$.

We finish the argument by showing that $z <_{++} z'$, thereby contradicting our choice of z being rightmost with respect to x and y. Since $z' \in P$ and $z \notin P$, $(N(z) \cap L_{i+2}^{++}) \subset (N(z') \cap L_{i+2}^{++})$. Thus by Fact 7, $(N(z) \cap L_i^{++}) \supseteq (N(z') \cap L_i^{++})$. If $(N(z) \cap L_i^{++}) \supset (N(z') \cap L_i^{++})$, then LBFS forces $z <_{++} z'$. Now assume $(N(z) \cap L_i^{++}) = (N(z') \cap L_i^{++})$ which implies there is a slice in L_{i+1}^{++} containing z and z'. By Fact 6 since $z' \in (L_{i+1}^{++} \cap P)$ and $z \in (L_{i+1}^{++} \setminus P)$, we know that $z' \in L_{k-i-1}^{+}$ and $z \in L_{k-i}^{+}$. Thus $z' <_{+} z$ and by Fact 8, $z <_{++} z'$ as required. □

We now turn to the implementation of the algorithm. There are many linear time implementations of LBFS described in the literature, for example see [7,8,20]. In our discussion, we will follow the implementation presented in [8], namely one that follows the paradigm of "partitioning". In this scheme, we start with all vertices in the same cell (i.e. slice) and choose an arbitrary vertex, often the first vertex in the cell. As we will see during our discussion of the implementation of LBFS+, it will be advantageous for us to assume that the vertices already have some order. When a vertex is chosen, i.e. is chosen as the *pivot*, it is placed in its own cell and invokes a partitioning of all cells that follow it in the ordering. Under this partitioning of a cell, vertices that are adjacent to the pivot form a new cell that precedes the cell containing the vertices not adjacent to the pivot. After this partitioning is complete, a new pivot is chosen from the cell immediately following the old pivot and the process of refinement continues. As pointed out by Lanlignel [13], one of the advantages of using this paradigm is that we immediately have an implementation of LBFS+. Once our initial LBFS has terminated, we merely reverse the ordering of the vertices produced by the first LBFS and run the algorithm again. Every time a slice is encountered, the last vertex from the previous LBFS is automatically the vertex at the front of the list.

To implement the last step of the algorithm, namely the test of the "neighbourhood condition", as in [3], to each vertex x we assign two integers, $lmn(x)$ and $rmn(x)$ storing the indices (in σ^{++}) of the leftmost (respectively, rightmost) element of $N[x]$. Note that by using the closed neighbourhood of x, $lmn(x)$ (respectively, $rmn(x)$) is set to the index of x if x has no neighbours to the left (respectively, to the right) in σ^{++}. It is easy to modify the LBFS implementation to store these values. The final step consists of determining for each x whether $deg(x) = rmn(x) - lmn(x)$. Thus we have:

Theorem 10. *The algorithm can easily be implemented to run in linear time.*

4. Concluding remarks

After the submission of this paper, there has been a number of further developments concerning the recognition of *uigs*. In particular, a new recognition algorithm based on "bicompatible elimination orderings" has recently appeared [17]. This algorithm, although linear time, seems to be more difficult to implement that the one presented in this paper.

Recently, the concept of a "certifying algorithm" [12] has received considerable attention. When applied to recognizing *uigs*, such an algorithm not only produces a

certificate that the algorithm is correct if the input graph is accepted as being a *uig* (i.e. it satisfies the "3-vertex condition" and thus has a *uig* representation) but also produces a certificate that the algorithm is correct if the input graph is rejected. Such a certificate of rejection is, in the case of *uigs*, an observed induced subgraph of the input graph that cannot appear in any *uig*. The first such algorithm is by Meister [16]. His *uig* recognition algorithm is similar to ours in that it involves 3 LBFS sweeps; however, his algorithm uses a variant of LBFS called "min-LexBFS". The test on the ordering produced by the third sweep is the same as ours (i.e. the one in [3]). Min-LexBFS can be implemented to run in linear time but does not use an "off the shelf" partitioning implementation, as does ours. Meister's algorithm does, however, produce a certificate if the input graph is not a *uig*. In particular, such a certificate is either an induced cycle of size greater than 3, a claw (i.e. $K_{1,3}$) or an "asteroidal triple" (an independent triple of vertices such that between any two there is a path that avoids the neighbourhood of the third). It seems as though Meister's certifying step could be applied to the present algorithm. A second certifying algorithm by Hell and Huang [9] augments our algorithm and uses Wegner's characterization of *uigs* [22], namely that a graph is a *uig* iff it does not contain an induced cycle of size greater than 3, a claw, or a "3-sun" or its complement, the "net" which consists of a triangle each of whose vertices is adjacent to a unique vertex of degree 1. Their algorithm produces such a certificate if G is not a *uig*. Interestingly, they also show that our algorithm can be augmented to provide a linear time certifying recognition algorithm for proper interval bigraphs. (A bipartite graph with bipartition (X, Y) is an *interval bigraph* if each vertex v is assigned an interval I_v and $x \in X, y \in Y$ are adjacent iff $I_x \cap I_y \neq \emptyset$; an interval bigraph is *proper* if no interval contains another.)

Acknowledgements

The author acknowledges the Natural Sciences and Engineering Research Council of Canada and CITO for financial assistance.

References

[1] A. Bretscher, D.G. Corneil, M. Habib, C. Paul, A simple linear time LexBFS cograph recognition algorithm—extended abstract, Proceedings of the 29th WG Workshop, Elspeet, The Netherlands, 2003, to appear.

[2] J.-M. Chang, C.-W. Ho, M.-T. Ko, LexBFS-ordering in asteroidal triple-free graphs, in: A. Aggarwal, C. Pandu Rangan (Eds.), Proceedings of the ISAAC 99, 10th Symposium on Algorithms and Computation, Lectures Notes in Computer Science, Vol. 1741, Springer, Berlin, 1999, pp. 163–172.

[3] D.G. Corneil, H. Kim, S. Natarajan, S. Olariu, A.P. Sprague, Simple linear time recognition of unit interval graphs, Inform. Process. Lett. 55 (1995) 99–104.

[4] D.G. Corneil, S. Olariu, L. Stewart, The LBFS structure and recognition of interval graphs, in preparation; extended abstract appeared as The ultimate interval graph recognition algorithm?—extended abstract, in: Proceedings of SODA 98, Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, 1998, pp. 175–180.

[5] X. Deng, P. Hell, J. Huang, Linear time representation algorithms for proper circular arc graphs and proper interval graphs, SIAM J. Comput. 25 (1996) 390–403.

[6] C.M.H. de Figueiredo, J. Meidanis, C.P. de Mello, A linear-time algorithm for proper interval graph recognition, Inform. Process. Lett. 56 (1995) 179–184.
[7] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[8] M. Habib, R. McConnell, C. Paul, L. Viennot, Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, Theoret. Comput. Sci. 234 (2000) 59–84.
[9] P. Hell, J. Huang, Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs, manuscript, 2003.
[10] P. Hell, R. Shamir, R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs, SIAM J. Comput. 31 (2001) 289–305.
[11] N. Korte, H. Möhring, An incremental linear-time algorithm for recognizing interval graphs, SIAM J. Comput. 18 (1989) 68–81.
[12] D. Kratsch, R.M. McConnell, K. Mehlhorn, J.P. Spinrad, Certifying algorithms for recognizing interval graphs and permutation graphs, in: Proceedings of the SODA 03, 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 2003, pp. 158–167.
[13] J.-M. Lanlignel, Private communications, 1999.
[14] P.J. Looges, S. Olariu, Optimal greedy algorithms for indifference graphs, Comput. Math. Appl. 25 (1993) 15–25.
[15] T. Ma, unpublished manuscript.
[16] D. Meister, Recognizing and computing minimal triangulations efficiently, Technical Report 302, Fakultät für Mathematik und Informatik, Universität Würzburg, 2002.
[17] B.S. Panda, S.K. Das, A linear time recognition algorithm for proper interval graphs, Inform. Process. Lett. 87 (2003) 153–161.
[18] F.S. Roberts, Indifference graphs, in: F. Harary (Ed.), Proof Techniques in Graph Theory, Academic Press, New York, 1969, pp. 139–146.
[19] F.S. Roberts, On the compatibility between a graph and a simple order, J. Combin. Theory Ser. B 11 (1971) 28–38.
[20] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (1976) 266–283.
[21] K. Simon, A new simple linear algorithm to recognize interval graphs, in: H. Bieri, H. Noltemeyer (Eds.), Proceedings of the CG 91, International Workshop on Computational Geometry, Lectures Notes in Computer Science, Vol. 553, Springer, Berlin, 1992, pp. 289–308.
[22] G. Wegner, Eigenschaften der nerven homologische-einfactor familien in R^n, Ph.D. Thesis, Universität Gottigen, Germany, 1967.