9-5-2016

Recognition Algorithm for Probe Interval 2-Trees

Breeann Flesch
Western Oregon University, fleschb@mail.wou.edu

Matthew Nabity
Western Oregon University, nabitym@mail.wou.edu

British Journal of Mathematics & Computer Science
18(4): 1-11, 2016, Article no.BJMCS.28344
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Recognition Algorithm for Probe Interval 2-Trees

Breeann Flesch[1*] and Matthew Nabity[1]

[1] *Western Oregon University, Monmouth, OR, USA.*

Authors' contributions

This collaborative work was done by both authors. Both authors designed and implemented the algorithm and analyzed the complexity. Finally, both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/28344
Editor(s):
(1) Dariusz Jacek Jakbczak, Chair of Computer Science and Management in this Department, Technical University of Koszalin, Poland.
Reviewers:
(1) Radosaw Jedynak, Kazimierz Pulaski University of Technology and Humanities, Radom, Poland.
(2) Lexter R. Natividad, Central Luzon State University, Philippines.
(3) Ferda Ernawan, Universiti Malaysia Pahang, Malaysia.
Complete Peer review History: http://www.sciencedomain.org/review-history/16060

Received: ***15th July 2016***
Accepted: ***27th August 2016***
Published: ***5th September 2016***

Original Research Article

Abstract

Recognition of probe interval graphs has been studied extensively. Recognition algorithms of probe interval graphs can be broken down into two types of problems: partitioned and non-partitioned. A partitioned recognition algorithm includes the probe and nonprobe partition of the vertices as part of the input, where a non-partitioned algorithm does not include the partition. Partitioned probe interval graphs can be recognized in linear-time in the edges, whereas non-partitioned probe interval graphs can be recognized in polynomial-time. Here we present a non-partitioned recognition algorithm for 2-trees, an extension of trees, that are probe interval graphs. We show that this algorithm runs in $\mathcal{O}(m)$ time, where m is the number of edges of a 2-tree. Currently there is no algorithm that performs as well for this problem.

Keywords: Probe interval graph; recognition algorithm; 2-tree; linear-time algorithm.

2010 Mathematics Subject Classification: 05C85, 68R10.

**Corresponding author: E-mail: fleschb@wou.edu*

1 Introduction

Let G be a simple, undirected, finite graph with vertex set $V(G)$ and edge set $E(G)$. The number of vertices of G is referred to as n and the number of edges by m. A graph G is a *probe interval graph* if there is a partition of $V(G)$ into P and N and a collection $\{I_v : v \in V(G)\}$ of closed intervals of $\mathbb{R}$ in a one-to-one correspondence with $V(G)$ such that $uv \in E(G)$ if and only if $I_u \cap I_v \neq \emptyset$ and at least one of u or v belongs to P. The set P is referred to as the probes, and the set N the nonprobes. The collection of intervals along with the partition into probes and nonprobes will be referred to in this paper as a *representation.* Probe interval graphs were introduced in conjunction with the human genome project, in order to aid with a task called physical mapping [1, 2, 3].

Recognition of probe interval graphs has been studied extensively. Recognizing probe interval graphs can be broken down into two types of problems: partitioned and non-partitioned. A partitioned recognition algorithm includes the probe and nonprobe partition as part of the input, where a non-partitioned algorithm does not. McConnell and Nussbaum present a linear-time recognition algorithm for the partitioned probe interval graph problem in [4]. Chang et al. proved the existence of a polynomial-time recognition algorithm for the non-partitioned probe interval graph problem in [5], but stated that a more efficient algorithm remains an open problem.

Because of the difficulty of the non-partitioned problem, many people have turned their attention to the recognition of probe interval graphs from specific families of graphs. Some examples of probe graph classes with non-partitioned recognition algorithms are chordal graphs [6], probe distance-hereditary graphs [7], probe cographs [8], and probe comparability graphs [9]. In this paper, we will add to this list, giving an efficient non-partitioned recognition algorithm for probe interval 2-trees. Our algorithm runs in $\mathcal{O}(m)$ time, where m is the number of edges of a 2-tree, and currently there is no algorithm that performs as well for this problem. In addition, we implemented our algorithm and tested it on a variety of challenging 2-trees.

2 Foundations

A 2-*tree* is recursively defined as follows.

- K_2 is a 2-tree;
- Suppose G is a 2-tree; create G' by adding a vertex to G adjacent to both vertices of some K_2 of G.

Pržulj and Corneil investigated a forbidden induced subgraph characterization for 2-tree probe interval graphs in [10], finding that there are at least 62 forbidden subgraphs. This large obstruction set was added to in [11], where Brown, et al. found one more infinite family of forbidden subgraphs. In that paper, there was a complete characterization of 2-tree probe interval graphs based on a structure called a sparse spiny interval 2-lobster (ssi2-lobster). We use this structure as our basis for the recognition algorithm.

Theorem 2.1. *[11] Let G be a $2-$tree. Then G is a probe interval graph if and only if it is an ssi2-lobster.*

To understand the structure of an ssi2-lobster, we recall the generalized idea of a path from Beineke and Pippert in [12]. As we walk through the details of the structure of the ssi2-lobster, we will simultaneously give the corresponding piece of the algorithm.

Definition 2.1. A 2-path of G is an alternating sequence of distinct 2-cliques and 3-cliques of G, $(e_0,\ t_1, e_1,\ t_2, e_2\ , ..., t_p, e_p)$, starting and ending with a 2-clique and such that t_i contains exactly two of the distinct 2-cliques: e_{i-1} and e_i $(1 \leq i \leq p)$. The *length* of the 2-path is the number p of 3-cliques. The letters e and t are used to remind us of edges and triangles (K_3s).

A vertex v of a 2-tree G is a 2-*leaf* if the degree of v is two. Let G be a 2-tree and define $G^{'}$ to be $G - \hat{L}$, where $\hat{L}$ is the set of 2-leaves of G; iteratively, $G^{''} = G^{'} - \hat{L}^{'}$. It is necessary, but not sufficient, that $G^{''}$ is a 2-path for G to be a probe interval graph. Thus we first remove the 2-leaves via Algorithm 2.1 twice.

Let G be the graph depicted in Fig. 1. In G the vertices $v_1, v_7, v_{11}, v_{15}, v_{17}$ and v_{18} are all 2-leaves, so $V(G^{'}) = \{v_2, v_3, v_4, v_5, v_6, v_8, v_9, v_{10}, v_{12}, v_{13}, v_{14}, v_{16}\}$. Now $G^{'}$ has the 2-leaves v_2, v_6, v_{14} and v_{16}, so $V(G^{''}) = \{v_3, v_4, v_5, v_8, v_9, v_{10}, v_{12}, v_{13}\}$.

In Algorithm 2.1, we define $M^{'}$ to be the adjacency matrix for $G^{'}$ and $M^{''}$ to be the adjacency matrix for $G^{''}$. In our implementation, for computational savings, we also store the degrees of the vertices for $G, G^{'}$ and $G^{''}$ and label the array as $d, d^{'}$ and $d^{''}$, respectively. The 2-leaves that are removed will later need to be classified as probes or nonprobes, so we save them and their neighbors in an array titled $\hat{L}_1$ for the first sweep and $\hat{L}_2$ for the second sweep.

Although we conceptually speak of removing the vertices from the graph, the implementation of our algorithm keeps the vertices in the adjacency matrix and zeros out the row and column, giving it degree zero. Maintaining the original data structure helps keep the indexing correct and saves on computations.

After removing the 2-leaves twice, we check to make sure the resulting graph, $G^{''}$, is a 2-path, which is only to check that the resulting matrix has exactly two 2-leaves. If it has more than two 2-leaves, then the graph is not a probe interval graph, and the algorithm ends. If it has exactly two 2-leaves, then we need to define the edges and triangles of the 2-path, which is done in Algorithm 2.2.

Algorithm 2.1: Remove 2-Leaves

Input: $n \times n$ adjacency matrix M and $n \times 1$ list of the degree of each vertex (if known)
Output: $n \times n$ adjacency matrix M', $n \times 1$ degree of each vertex d', and $\hat{l} \times 3$ matrix $\hat{L}$ of 2-leaves of M with neighbors

```
l = 0;
if Degrees are unknown (d = 0) then
    for i = 1 : n do
        d(i,1) = sum ith row of M;
        if d(i,1) = 2 then
            Store index i and neighbors in L;
            Count number of 2-leaves: l = l + 1;
else
    for i = 1 : n do
        if d(i,1) = 2 then
            Store index i and neighbors in L;
            Count number of 2-leaves: l = l + 1;
Copy the original adjacency matrix and list of degree: M' = M, d' = d;
for i = 1 : l do
    Zero out corresponding rows and columns of M';
    Adjust degrees in d' based on pruned vertices;
```

Algorithm 2.2: Construct 2-Path

Input: $n \times n$ adjacency matrix M'', $n \times 1$ list of the degree of each vertex d''
Output: $p \times 3$ matrix T of triangles and $(p+1) \times 2$ matrix E of edges

Locate 2-leaves and remaining active vertices in M'';
d_{count} = the number of two leaves;
$active$ = the number of active vertices in M'';
if $d_{count} > 2$ **then**
    Exit, not a Probe Interval Graph;
$sweep = 1$;
while $sweep < active - 1$ **do**
    if $sweep = 1$ **then**
        $i = locate(1, 1)$, $T(sweep, 1) = i$;
        Find neighbors of vertex i and store in $T(sweep, 2)$ and $T(sweep, 3)$;
        Construct E using the two appropriate vertices from current triangle T;
        Select new vertex i from currrent neighborhood in T;
        $sweep = sweep + 1$;
    else
        Current location in ith vertex
        Construct E using the two appropriate vertices from current triangle T;
        Select new vertex i from currrent neighborhood in T;
        $sweep = sweep + 1$;
        if $sweep = active - 2$ **then**
            Return T and E as the 2-path has been identified;

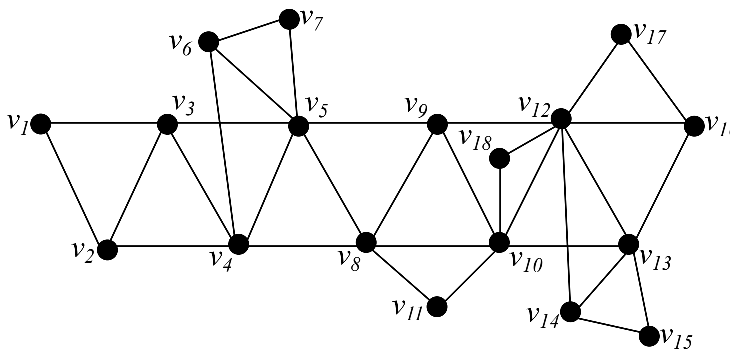

Fig. 1. An example of a probe interval 2-tree

3 Classification of Vertices

If G'' is a 2-path we then need to start classifying certain vertices to determine whether it is an ssi2-lobster and to later help with the partition into probes and nonprobes. Suppose G is a 2-tree such that G'' is the 2-path $(e_0, t_1, e_1, t_2, \ldots, t_p, e_p)$, such that e_0 and e_p are defined in the following way. Let a_0 be a 2-leaf of G' such that $N_{G'}(a_0) \subset t_1$ and a_p be a 2-leaf of G' such that $N_{G'}(a_p) \subset t_p$. Define $e_0 = N_{G'}(a_0)$ and $e_p = N_{G'}(a_p)$. This will be our intended meaning for e_0 and e_p for the rest of the paper. Note that there may be an ambiguity in which edge of G'' is to be e_0 or e_p, but this choice may always be made arbitrarily as it does not affect any results.

Consider again our example G in Fig. 1. Notice here that $G^{''}$ is a 2-path, with $e_0 = \{v_3, v_4\}$, $t_1 = \{v_3, v_4, v_5\}$, $e_1 = \{v_4, v_5\}$ $t_2 = \{v_4, v_5, v_8\}$, $e_2 = \{v_5, v_8\}$, $t_3 = \{v_5, v_8, v_9\}$, $e_3 = \{v_8, v_9\}$, $t_4 = \{v_8, v_9, v_{10}\}$, $e_4 = \{v_9, v_{10}\}$, $t_5 = \{v_9, v_{10}, v_{12}\}$, $e_5 = \{v_{10}, v_{12}\}$, $t_6 = \{v_{10}, v_{12}, v_{12}\}$ and $e_6 = \{v_{12}, v_{13}\}$.

After the completion of Algorithm 2.2, we need check the vertices $\hat{L}_1$ and $\hat{L}_2$ to see if their neighborhood in G and $G^{'}$, respectively, are equal to some e_i in the 2-path created in Algorithm 2.2. This is the first step in Algorithm 3.1. If a vertex has a neighborhood not equal to an e_i, then it is put in L_1 or L_2. We now formally define L_1 and L_2:.

- $L_1 = \{v \in \hat{L}_1 : N_G(v) \neq e_i \ (0 \leq i \leq p)\}$;
- $L_2 = \{v \in \hat{L}_1 : N_{G'}(v) \neq e_i \ (0 \leq i \leq p)\}$.

Algorithm 3.1: Categorize 2-Leaves

Input: $\hat{l}_1 \times 3$ matrix $\hat{L}_1$ of 2-leaves from M, $\hat{l}_2 \times 3$ matrix $\tilde{L}_2$ of 2-leaves from M', and $n \times 1$ list d'' of the degree of each vertex in M''

Output: $l_1 \times 3$ matrix L_1, $l_1^1 \times 3$ matrix L_1^1, $l_1^2 \times 3$ matrix L_1^2, $l_2 \times 3$ matrix L_2

Set $L_1 = \hat{L}_1$ and $L_2 = \hat{L}_2$ with $l_1 = \hat{l}_1$ and $l_2 = \hat{l}_2$;
Remove from L_1 any 2-leaves in L_1 with neighborhoods in E and adjust l_1 accordingly;
Remove from L_2 any 2-leaves in L_2 with neighborhoods in E and adjust l_2 accordingly;
if L_2 *is nonempty* **then**
    Exit, not a Probe Interval Graph;
Set $L_1^1 = L_1$ and $l_1^1 = l_1$;
Remove from L_1^1 any vertices whose neighborhood is nonzero in d'' and adjust l_1^1 accordingly;
Set $L_1^2 = L_1$ and $l_1^2 = l_1$;
Remove from L_1^2 any vertices also in L_1^1 and adjust l_1^2 accordingly;

In the example from Fig. 1, $\hat{L}_1 = \{v_1, v_7, v_{11}, v_{15}, v_{17}, v_{18}\}$ and $L_1 = \{v_1, v_7, v_{11}, v_{15}, v_{17}\}$. The vertex v_{18} is not in L_1, since its neighborhood is $e_5 = \{v_{10}, v_{12}\}$. The rest of the vertices, though, have a neighborhood which is not equal to any e_i, so they are in L_1. Also, $\hat{L}_2 = \{v_2, v_6, v_{14}, v_{16}\}$ and $L_2 = \emptyset$, since all of the vertices from $\hat{L_2}$ have a neighborhood equal to an e_i in $G^{'}$. If L_2 is not empty, then G is not a spiny interior 2-lobster nor a probe interval graph, and the algorithms ends.

Definition 3.1. A 2-tree G is a 2-*lobster* if $G^{''}$ is a 2-path. A *spiny interior* 2-*lobster* is a 2-lobster G with $L_2 = \emptyset$.

Spiny interior 2-lobsters are the defining characteristic of another variation of interval graphs, called interval 3-graphs. However, for G to be a probe interval graph, the spiny interior 2-lobster must also be sparse. Thus we have some further checking of conditions if L_2 is empty. Algorithm 3.1 then checks each vertex in L_1 to see if its neighborhood in G is a subset of the 2-path found in Algorithm 2.1, creating L_1^1 and L_1^2, which are formally as

- $L_1^1 = \{v \in L_1 : N_G(v) \subseteq V(G^{''})\}$;
- $L_1^2 = \{v \in L_1 : N_G(v) \not\subseteq V(G^{''})\}$.

In Fig. 1, $L_1 = \{v_1, v_7, v_{11}, v_{15}, v_{17}\}$, so these vertices are divided into $L_1^1 = \{v_{11}\}$ and $L_1^2 = \{v_1, v_7, v_{15}, v_{17}\}$ by Algorithm 3.1. The vertex v_{11} has a neighborhood in G of v_8 and v_9, both of which are vertices in $G^{''}$. On the other hand, consider vertex v_7, whose neighborhood in G is v_6 and v_5. The vertex v_6 is not in $G^{''}$, so v_7 is put into L_1^2.

Some subset of the vertices of L_1^1 and L_1^2 will end up being part of the set of nonprobes, but which subset is determined by relationships with the vertices of $G^{''}$. Algorithm 3.2 creates the sets W^1, W^2, W^3 and $W^{3'}$ which help determine those relationships. We now define the following:

- $W^1 = \{v \in V(G^{''}) : N_G(x) = t_i - v$ for some $x \in L_1^1$, $(1 \leq i \leq p)\}$;
- $W^2 = \{v \in V(G^{''}) : N_G(y) = N_{G''}(z) + z - v = e_i + z - v$ for some $y \in L_1^2$ and $z \in V(G)$, $(1 \leq i \leq p-1)\}$;
- $W^3 = \{v \in V(G^{''}) : N_G(y) = N_{G''}(z) + z - v = e_i + z - v$ for some $y \in L_1^2$ and $z \in V(G)$, $(i \in \{0, p\})\}$.

Algorithm 3.2: Categorize 2-Path

Input: $l_1^1 \times 3$ matrix L_1^1, $l_1^2 \times 3$ matrix L_1^2, $\tilde{l}_2 \times 3$ matrix $\tilde{L}_2$, $p \times 3$ matrix T, and $(p+1) \times 2$ matrix E

Output: vectors W^1, W^2, W^3 and $W^{3'}$

```
for i = 1 : l_1^1 do
    Locate the vertex in T that has neighbors L_1^1(i,2) and L_1^1(i,3) and then put this vertex
    in W^1;
for i = 1 : l_1^2 do
    Locate the neighbor of L_1^2(i,1) that is in L̂_2;
    Set xyz equal to the located row in L̂_2;
    if yz ∈ E(1,:) or yz ∈ E(p+1,:) then
        From yz, choose the vertex which is not a neighbor of L_1^2(i,1) and add it to W^3;
    else
        From yz, choose the vertex which is not a neighbor of L_1^2(i,1) and add it to W^2;
If a vertex occurs more than once in W^3 add it to W^3';
```

For each vertex $x \in L_1^1$, you check which t_i in $G^{''}$ contains the neighborhood of x. There will be exactly one vertex v in that t_i that is not adjacent to x, and that v goes in W^1.

In G from Fig. 1, we consider v_{11}, which is in L_1^1. Notice that v_{11} is adjacent to v_8 and v_{10}, which are in t_4 defined in Algorithm 2.2. The only vertex that is part of t_4 that is not adjacent to v_{11} is v_9, so $v_9 \in W^1$, and since there is only one vertex in L_1^1, $W^1 = \{v_9\}$. The idea is that v_{11} and v_9 will both need to be nonprobes with intersecting intervals. Thus you want these two vertices to be non-adjacent, since two nonprobes cannot have an edge between them. The algorithm continues to find these types of pairs of vertices.

Similarly we look at every vertex $x \in L_1^2$. In this case, however, every vertex $x \in L_1^2$ will be adjacent to exactly one vertex, z from $\hat{L}_2$. Furthermore, that neighborhood of z in $G^{'}$ and the neighborhood of x in G will differ by exactly one vertex, v, which goes in W^3 if $N_{G'}(z)$ is equal to either e_0 or e_p or W^2 otherwise. In the graph from Fig. 1, consider $v_7 \in L_1^2$. It is adjacent to $v_6 \in \hat{L}_2$ and $N_{G'}(v_6) = e_1$ and the vertex that v_7 is not adjacent to from e_1 is v_4, so $v_4 \in W^2$. Consider further $v_{17} \in L_1^2$, which for the reasons stated above places $v_{13} \in W^3$. After checking each vertex in L_1^2, we get $W^2 = \{v_4\}$ and $W^3 = \{v_4, v_{12}, v_{13}\}$.

Notice that W^3 will never be empty because Algorithm 2.1 eliminates two vertices on either end of the longest 2-path in G. Thus some tricky things can happen on the end of 2-path in $G^{''}$. This forces the creation of another set of vertices $W^{3'}$, which is a subset of the vertices of W^3 and defined as follows:

- $W^{3'}(G) = \{v \in W^3 : N_G(y) = N_{G''}(z) + z - v = e_i + z - v$ and $N_G(s) = N_{G''}(r) + r - v = e_i + r - v$ for $y, s \in L_1^2$ and distinct $z, r \in V(G)$, $(i = 0$ or $i = p)\}$.

In G from Fig. 1, $W^{3'}$ is empty because for each vertex v in W^3 there are not two distinct vertices from L_1^2 that place v in W^3.

4 Partition and Complexity

We are now ready to identify the class of 2-trees which are probe interval graphs: the sparse spiny interior 2-lobsters.

Definition 4.1. Let G be a spiny interior 2-lobster with $G^{''}$ the 2-path $(e_0,\ t_1,\ e_1,\ t_2,\ \ldots,\ t_p,e_p)$. The following two conditions hold if and only if G is a *sparse spiny interior* 2*-lobster (ssi*2*-lobster)*:

1. No $t_i, 1 \leq i \leq p$, has two vertices in $W^1 \cup W^2 \cup W^{3'}$.
2. No $t_i, i \in \{1, p\}$, has three vertices x, y, and z such that $x, y \in W^3$ and if $e_0 = xy$ or $e_p = xy$ then $z \in W^1 \cup W^2 \cup W^{3'}$.

Algorithm 4.1 checks the condition above to verify that the graph G is an ssi2-lobster, which implies that it is a probe interval graph. This check shows that there is a partition of vertices into probes and nonprobes, but does not necessarily give the partition, so the algorithm needs to take it a step further. Although it is true that all of the vertices of $W^1 \cup W^2 \cup W^{3'}$ must be nonprobes as well as the corresponding vertices of L_1^1 and L_1^2 that put them there, not all of W^3 must be nonprobes. Consider our example G from Fig. 1, $W^1 \cup W^2 \cup W^{3'} = \{v_9, v_4\}$ and $W^3 = \{v_4, v_{12}, v_{13}\}$. Since v_{12} and v_{13} are adjacent to one another, they can't both be nonprobes. Thus the algorithm checks to see one of these vertices is adjacent to another in $W^1 \cup W^2 \cup W^{3'}$. If it is adjacent to a vertex in $W^1 \cup W^2 \cup W^{3'}$, then it, and the corresponding vertex from L_1^2, will not be nonprobes. If there isn't a vertex adjacent to $W^1 \cup W^2 \cup W^{3'}$, then an arbitrary choice can be made. In our example, v_{12} is adjacent to v_9, so v_{12} and v_{15} (the vertex that put v_{12} in W^3) are both probes. Algorithm 4.1 checks these last details and outputs the nonprobes and probes, if it is a probe interval graph. In our example, the nonprobes are the set $N = \{v_1, v_4, v_7, v_9, v_{11}, v_{13}, v_{17}\}$ and the set of probes $P = V(G) - N$.

Now Algorithm 4.2 puts all of the steps together representing our entire implementation. It inputs an adjacency matrix for a 2-tree, and if the 2-tree is not a probe interval graphs, it exits. If the 2-tree is a probe interval graph, then it outputs the partition of probes and nonprobes.

In addition to the aforementioned pseudo-code, we also implemented the algorithm in the Matlab environment and tested it on a variety of challenging 2-trees. Our implementation consistently returned the desired partition within the estimated number of operations. We now prove the complexity of our algorithm for all 2-trees.

Theorem 4.1. *Probe interval* 2*-trees can be recognized in* $\mathcal{O}(n^2)$ *time.*

Proof. Algorithm 4.2 determines whether the inputted 2-tree is a probe interval graph by determining whether it is an ssi2-lobster and then outputs the partition of vertices into probes and nonprobes. Thus we now determine the complexity of Algorithm 4.2.

We begin with an $n \times n$ adjacency matrix M for a graph G with n vertices. The first call to Algorithm 2.1 requires the computation of the degree of each vertex. This is accomplished by a row sum for each row in the matrix M. Along with computing the row sum, the location of the neighbors is noted requiring a total of $2n$ operations. When a 2-leaf is located, the desired information is stored. If d_{count} is the number of 2-leaves identified, there are $2d_{count}$ operations

Algorithm 4.1: Categorize Probes and Nonprobes

Input: vectors W^1, W^2, W^3, $W^{3'}$, $(p+1) \times 2$ matrix of edges E, $l_1^2 \times 3$ matrix L_1^2, $l_1^1 \times 3$ matrix L_1^1, and $n \times n$ adjacency matrix M

Output: Partition consiting of a vector of Probe vertices P and a vector of nonprobe vertices N

Set $W = W^3 - \left(W^1 \cup W^2 \cup W^{3'}\right)$;

if $E(1,:)$ *or* $E(p+1,:)$ *has exactly one vertex in* W **then**

  Locate that vertex in W^3 and let i equal the index in W^3;

  Remove the ith vertex from W^3;

  Remove the ith row from L_1^2;

if $E(1,:)$ *or* $E(p+1,:)$ *has both vertices in* W **then**

  Locate that vertex in W^3 that is adjacent to $W^1 \cup W^2 \cup W^{3'}$ and let i equal the index in W^3;

  Remove the ith vertex from W^3;

  Remove the ith row from L_1^2;

Set $N = W^1 \cup W^2 \cup W^3 \cup L_1^2(:,1) \cup L_1^1(:,1)$;

if N *has two adjacent vertices* **then**

  Exit; not a Probe Interval Graph;

Set $P = V(G) - N$;

Algorithm 4.2: Probe Interval Graph Recognition

Input: $n \times n$ adjacency matrix M

Output: Partition consiting of a vector of Probe vertices P and a vector of nonprobe vertices N

Prune 2-leaves:

$[d, M', d', \hat{L}_1] = Remove2\text{–}Leaves(M, d)$;

Prune 2-leaves again:

$[dd, M'', d'', \hat{L}_2] = Remove2\text{–}Leaves(M', d')$;

Construct 2-path:

$[T, E] = Construct2\text{–}Path(M'', d'', d')$;

Categorize 2-leaves:

$[L_1, L_1^1, L_1^2, L_2] = Categorize2\text{–}Leaves(\hat{L}_1, \hat{L}_2, d'')$;

Categorize 2-path:

$[W^1, W^2, W^3, W^{3'}] = Categorize2\text{–}Path(L_1^1, L_1^2, \hat{L}_2, T, E)$;

Categorize probe and nonprobe vertices:

$[P, N] = CategorizeProbes\text{–}Nonprobes(W^1, W^2, W^3, W^{3'}, E, L_1^2, M)$;

where $2 \leq d_{count} \leq n$. Finally, pruning the 2-leaves and adjusting the vector of degrees requires an additional $5d_{count}$ operations bringing the total to $n(2n + 2d_{count}) + 5d_{count}$. Using the bound on d_{count} we have that the total cost is $4n^2 + 5n$ or $\mathcal{O}(n^2)$ for this first call to Algorithm 2.1.

The next step of Algorithm 4.2 is again a call to Algorithm 2.1. Some complexity savings are made as the degree of each vertex is known. Identifying the location of the 2-leaves, the neighboring vertices and pruning requires a total of

$$n(2n + 3) + 5d_{count},$$

where d_{count} is again the number of 2-leaves. In this case, we have $2 \leq d_{count} \leq \frac{n}{2}$ and our total cost is $2n^2 + \frac{11}{2}n$ or $\mathcal{O}(n^2)$ for the second call to Algorithm 2.1.

Next we construct the 2-path using Algorithm 2.2. Algorithm 2.2 begins by identifying the active vertices which requires n operations. Another $2n$ are required to check if a 2-path exits. To construct the 2-path, we begin at one of the 2-leaves and work towards the other 2-leaf. This process requires $2n + 3n(active - 2)$ operations where *active* is the number of vertices in M''. Since we pruned 2-leaves in the repeated calls to Algorithm 2.1, we have that $active \leq n - 6$ so the total cost is $3n^2 - 19n$ or $\mathcal{O}(n^2)$ to construct the 2-path.

The next step of Algorithm 4.2 uses Algorithm 3.1 to categorize the 2-leaves. The parameters of interest for this algorithm are the following:

- $\hat{l}_1$ = the number of 2-leaves in M
- $\hat{l}_2$ = the number of 2-leaves in M'
- p = the length of the 2-path T.

We have that $\hat{l}_1 + \hat{l}_2 + p + 2 = n$. First, the neighborhoods in $\hat{L}_1$ are checked against the edges in E requiring $2\hat{l}_1(p+1)$ operations. Removing identified vertices from $\hat{L}_1$ to construct L_1 requires $r_{count}\hat{l}_1$ operations where r_{count} is the number of vertices to be removed and $r_{count} < \hat{l}_1$. The total cost for constructing L_1 from $\hat{L}_1$ is at most $2\hat{l}_1(p+1) + (\hat{l}_1)^2$. Constructing L_1^1 from L_1 requires $l_1 + r_{count}l_1$ operations where $l_1 \leq \hat{l}_1$ and $r_{count} \leq l_1$. This results in at most $\hat{l}_1 + (\hat{1}_1)^2$ operations to construct L_1^1. Constructing L_1^2 requires $l_1^1 l_1 + r_{count}l_1$ operations where $l_1^1 \leq \hat{l}_1$ and $r_{count} \leq l_1$. Here we have at most $2(\hat{l}_1)^2$ to construct L_1^2. Lastly, the neighborhoods in $\hat{L}_2$ are checked against the edges in E requiring $2\hat{l}_2(p+1)$ operations. Removing vertices from $\hat{L}_2$ to construct L_2 requires an additional $r_{count}\hat{l}_2$ operations where $r_{count} \leq \hat{l}_2$. Thus L_2 requires at most $2\hat{l}_2(p+1) + (\hat{l}_2)^2$ operations. Putting this all together we have a total cost of approximately

$$2\hat{l}_1(p+1) + 4(\hat{l}_1)^2 + \hat{l}_1 + \hat{l}_2(p+1) + (\hat{l}_2)^2$$

or at most $5n^2$ or $\mathcal{O}(n^2)$ operations.

Next we categorize the 2-path using Algorithm 2.2. Constructing W^1 requires the vertices in L_1^1 to be checked against the triangles in T requiring $3pl_1^1$ operations where $l_1^1 \leq \hat{l}_1$. Constructing W^2 and W^3 requires $2l_2l_1^2$ operations where $l_2 \leq \hat{l}_2$ and $l_1^2 \leq \hat{l}_1$. Further constructing the set $W^{3'}$ requires $2\left|W^3\right|$ operations where $\left|W^3\right|$. Putting this all together we have a total cost of

$$3pl_1^1 + 2l_2l_1^2 + 2\left|W^3\right|$$

for at most $5n^2 + 2n$ or $\mathcal{O}(n^2)$ operations.

The last step in Algorithm 4.2 is to categorize the vertices as probes or nonprobes and construct the partition by a call to 4.1. Checking that e_0 or e_p has exactly one vertex in W or two vertices in W and removing the desired vertex from W^3 requires

$$2\left|W\right| + 2n + \left(\left|W^1 \cup W^2 \cup W^{3'}\right| + \left|W^3\right| + 3l_1^2\right)n + 4\left|W^3\right|$$

operations for an upper bound of at most $5n^2 + 8n$ operations or $\mathcal{O}(n^2)$.

All together, the steps of Algorithm 4.2 require at most $24n^2 + \frac{3}{2}n$ or $\mathcal{O}(n^2)$ operations. □

Corollary 4.2. *Probe interval 2-trees can be recognized in $\mathcal{O}(m)$ time.*

Proof. Since $m = \mathcal{O}(n^2)$, Algorithm 4.2 runs in $\mathcal{O}(m)$ time. □

It is worth noting that while the algorithm runs in $\mathcal{O}(n^2)$ time, that the bound of $24n^2 + \frac{3}{2}n$ operations could be lower. Look for example at the bound on the number of 2-leaves, or d_{count} in the algorithm. The bound for the number of 2-leaves of G is n and the bound of the number of 2-leaves of $G^{'}$ is $\frac{n}{2}$. Both of these bounds are tight individually, but a 2-tree cannot simultaneously attain both of these bounds. There are many places where this happens, so there is research to be done in finding a lower constant.

5 Conclusions

A graph is a probe interval graph if its vertices can be partitioned into probes and nonprobes with an interval associated to each vertex so that vertices are adjacent if and only if their corresponding intervals intersect and at least one of them is a probe. A 2-tree is recursively defined by adding a vertex to an existing 2-tree that is adjacent to both vertices of some K_2 in the 2-tree, and a K_2 is a 2-tree. We have introduced, implemented and tested an efficient non-partitioned recognition algorithm for 2-trees, an extension of trees, that are probe interval graphs. Our algorithm is based on a structure called a sparse spiny interval 2-lobster introduced in [11]. The complexity of our approach is $\mathcal{O}(n^2)$ or $\mathcal{O}(m)$.

Our result is comparable with the partitioned algorithm of McConnell and Nussbaum [4], since their result was linear in edges. Our result is an improvement in state-of-the-art algorithms for this problem, since our nonparitioned algorithm is faster than all known nonpartitioned algorithms and runs in the same time as a partitioned algorithm. While our result is specific to 2-trees, the techniques may be generalizable. Often recognition algorithms use obstruction sets as a basis, while our algorithm uses a structural characterization. Since the non-partitioned version of recognition algorithms is quite a bit more difficult, future research in structural characterizations may be needed.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Sheng L, Wang C, Zhang P. On the perfectness of tagged probe interval graphs. In discrete mathematical problems with medical applications, volume 55 of DIMACS Series of Discrete Mathematics and Theoretical Computer Science, Providence: American Mathematical Society. 2000;159-163.

[2] Zhang P. United states patent; 1997.
Available: http://www.cc.columbia.edu/cu/cie/ techlists/patents/5667970.htm

[3] Zhang P, Schon E, Cayanis E, Weiss J, Kistler S, Bourne P. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. CABIOS. 1994;10(3):309-317.

[4] McConnell R, Nussbaum Y. Linear-time recognition of probe interval graphs. SIAM Journal on Discrete Mathematics. 201529(4):2006-2046.

[5] Chang G, Kloks A, Liu J, Peng S. The PIGs Full Monty-a floor show of minimal separators. In STACS 2005, volume 3404 of Lecture Notes in Computer Science, Berlin: Springer. 2005;521-532.

[6] Golumbic M. Algorithmic graph theory and perfect graphs. New York: Academic Press, Harcourt Brace Jovanovich Publishers; 1980 .

[7] Chang M, Hung L, Rossmanith P. Recognition of probe distance-hereditary graphs. Discrete Applied Mathematics. 2013;161(3):336-348.

[8] Le V, de Ridder H. Characterisations and linear-time recognition of probe cographs. In Graph-theoretic Concepts in Computer Science, volume 4769 of Lecture Notes in Computer Science, Berlin: Springer. 2007;226-237.

[9] Chandler D, Chang M, Kloks T, Liu J, Peng S. Partitioned probe comparability graphs. Theoretical Computer Science. 2008;396(1-3):212-222.

[10] Pržulj N, Corneil D. 2-tree probe interval graphs have a large obstruction set. Discrete Applied Mathematics. 2005;150(1-3):216-231.

[11] Brown D, Flesch B, Lundgren J. A characterization of 2-tree probe interval graphs. Discussiones Mathematicae Graph Theory. 2014;34(3):509-527.

[12] Beineke L, Pippert R. Properties and characterizations of k-trees. Mathematika. 1971;18:141-151.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/16060