ELSEVIER
Available online at www.sciencedirect.com
SCIENCE DIRECT®
Discrete Applied Mathematics 131 (2003) 151–167
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

MAD trees and distance-hereditary graphs

E. Dahlhaus[a], P. Dankelmann[b], W. Goddard[c,*], H.C. Swart[b]

[a] *Institute of Computer Graphics, Vienna University of Technology, Austria*
[b] *School of Mathematical and Statistical Sciences, University of Natal, Durban, South Africa*
[c] *Department of Computer Science, University of Natal, 4041, Durban, South Africa*

Received 26 July 2000; received in revised form 7 March 2001; accepted 5 May 2002

Abstract

For a graph G with weight function w on the vertices, the total distance of G is the sum over all unordered pairs of vertices x and y of $w(x)w(y)$ times the distance between x and y. A MAD tree of G is a spanning tree with minimum total distance. We develop a linear-time algorithm to find a MAD tree of a distance-hereditary graph; that is, those graphs where distances are preserved in every connected induced subgraph.

Keywords: Distance; Spanning tree; Minimum average distance; Distance-hereditary graphs

1. Introduction

The problem of finding a minimum-cost spanning tree is one of the classic algorithmic questions in graph theory. In several other instances the problem of finding the best spanning tree has been studied; for example, spanning trees with minimum diameter, minimum radius, minimum number of leaves or minimum average distance. We consider here the latter problem in a restricted class of graphs.

In this paper all our graphs will be simple. By a weighted graph, we mean a graph together with a function which assigns a positive integral weight to each vertex. Given a weighted graph G with weight function w, the *total distance* of G and w is

$$d(G,w) := \sum_{\{x,y\}\subset V(G)} w(x)w(y)d_G(x,y),$$

* Corresponding author. Tel.: +27-31-260-3018; fax: +27-31-261-6550.
E-mail address: goddard@nu.ac.za (W. Goddard).

0166-218X/03/$ - see front matter
PII: S0166-218X(02)00422-5

where $V(G)$ is the vertex set of G and $d_G(x, y)$ is the distance between x and y in G. If there is no ambiguity, we omit the argument w and write just $d(G)$. The average distance of G and w is the total distance divided by $\binom{N}{2}$, where N is the sum $\sum_{x \in V(G)} w(x)$ of all weights.

The rationale behind these definitions is from a facility location problem. If the weight indicates how many facilities are located at a particular vertex, then the average distance is the expected distance between two randomly chosen facilities. Also, in attempting an algorithm for unweighted graphs where parts of the graph are contracted, this idea of weighted graphs and total distance arises naturally.

The *distance* of a vertex v is

$$\sigma_G(v) := \sum_{x \in V(G)} w(x) d_G(x, v).$$

The distance of a vertex gives the total distance from all facilities to that vertex. If there is no ambiguity we will write just $\sigma(v)$.

A *MAD tree* of a graph is defined as a spanning tree with minimum average distance or, equivalently, with minimum total distance. In general, finding a MAD tree is NP-hard [15]. Entringer et al. [11] showed that there is a spanning tree whose average distance is less than twice the average distance of the original, and that such a tree can be found in polynomial time. Wu et al. [16] developed a polynomial-time approximation scheme to approximate a MAD tree. A discussion of further results on MAD trees is given in [10]. In [8], it is shown that a MAD tree of an outerplanar graph can be found in polynomial time.

In this paper, we develop an efficient algorithm to find a MAD tree of a (weighted) distance-hereditary graph. A graph is called *distance-hereditary* if it preserves distances in every connected induced subgraph. Distance-hereditary graphs were introduced by Howorka [14] and are studied in [1,9].

First, we discuss some structural properties of MAD trees. Then, we present a polynomial-time algorithm for MAD trees in this class of graphs. Finally, we use the fact that distance-hereditary graphs are exactly those graphs that can be split decomposed into stars and cliques to obtain a linear-time algorithm to determine a MAD tree.

2. The structure of MAD trees

A breadth-first-search (BFS) tree is a spanning tree which is distance-preserving from a vertex. The second author showed that MAD trees are not in general BFS trees [7]. Nevertheless, we show that MAD trees have some structure.

For convenience, we will work mainly with the total rather than the average distance. If X and Y are sets, we define $d(X, Y)$ as the weighted sum of all the distances between vertices in X and vertices in Y. That is: $\sum_{x \in X, y \in Y} w(x)w(y)d_G(x, y)$. If necessary, we will indicate by a subscript the graph we are working in.

We will also use the following notation. The edge set of graph G is denoted by $E(G)$. If $S \subseteq V(G)$ then $G[S]$ denotes the subgraph induced by the set S. Further, if w is a weight function then $w(S)$ denotes the sum of weights of S.

Recall that a *median* vertex of a graph is one with minimum distance. Barefoot et al. [2] showed the following property of a median vertex of a tree. Although they stated it only for unweighted trees, their proof can be trivially extended to weighted trees.

Lemma 1 (Barefoot et al. [2]). *Let T be a weighted tree, v a median vertex of T, and let u, w be vertices such that the path from u to v in T contains w. Then $\sigma(u) \geqslant \sigma(w) \geqslant \sigma(v)$.*

Lemma 2. *Let T be a MAD tree of graph G and weight w.*

(a) *Let uv be an edge of T and let T_u and T_v denote the components of $T - uv$ containing u and v, respectively. Then $\sigma_{T_v}(v) \leqslant \sigma_{T_v}(y)$ for all vertices y in T_v such that u and y are adjacent in G.*
(b) *If T' is a subtree of T, and w' is the weight function that assigns to each vertex v of T' the total weight of the vertices in the component of $T - E(T')$ containing v, then T' is a MAD-tree of $G[V(T')]$ and w'.*

Proof. (a) Consider any spanning tree T' of G which contains T_u and T_v as subtrees. Then one edge of T' joins T_u and T_v. Say the edge is $e = xy$ with $x \in V(T_u)$ and $y \in V(T_v)$.

Then the total distance of T' can be written as the sum of three pieces depending on whether the two vertices are both in T_u, both in T_v or neither. Thus

$$d(T') = d(T_u) + d(T_v) + d_{T'}(V(T_u), V(T_v)).$$

The third term is the only one that depends on the choice of e. Each path from a vertex in T_u to a vertex in T_v can be split up into three parts: the portion in T_u to x, the portion in T_v to y, and the edge e. For each facility in T_u, there are $w(T_v)$ paths to T_v which need to be summed, and there are $w(T_u)w(T_v)$ paths that use e. Thus, the third term can be written as

$$d_{T'}(V(T_u), V(T_v)) = w(T_v)\sigma_{T_u}(x) + w(T_u)w(T_v) + w(T_u)\sigma_{T_v}(y).$$

Thus, we have an expression for $d(T')$ the only part of which that depends on y is $\sigma_{T_v}(y)$. Since T is a MAD tree, for fixed x the y must be the vertex that minimises $\sigma_{T_v}(y)$. That is, $\sigma_{T_v}(v) \leqslant \sigma_{T_v}(y)$, as required.

(b) Let $W = V(T')$. Consider any spanning tree U of the graph $G[W]$ and weight w'. This extends to a spanning tree T_U of G by the addition of the edges $E(T) - E(T')$. For each vertex $v \in W$, let T_v denote the component of $T - E(T')$ containing v.

The total distance of T_U can be split up into distances between vertices in the same component of $T - E(T')$ and distances between vertices in different components of $T - E(T')$. The former is equal to $\sum_{v \in W} d(T_v, w)$.

The latter can be divided into the portion of the paths inside U and the portions outside. The total of the paths inside U is $d(U, w')$. And, for each facility in T_v there are $w(V(T - T_v))$ paths that leave T_v. Thus the total for paths leaving T_v is $w(V(T - T_v))\sigma_{T_v}(v)$.

Thus, the total distances of T_U and U are related by

$$d(T_U, w) = d(U, w') + \sum_{v \in W} d(T_v, w) + w(V(T - T_v))\sigma_{T_v}(v).$$

It follows that the spanning tree U with minimum total distance gives the tree T_U with minimum total distance and vice versa. In particular, if one could improve on T' then one could improve on T, a contradiction. □

If P is a path in a graph G, then a *chord* of P is an edge of G joining nonconsecutive vertices of the path. Two chords e and f are said to *nest* if the subpath of P joining the vertices of e is contained in the subpath of P joining the vertices of f, or vice versa. The next lemma shows that paths in MAD trees have nested chords.

Lemma 3. *Let T be a MAD tree of a weighted graph G and let P be a path in T. Then any pair of chords of P nest.*

Proof. Let $P = v_0, v_1, \ldots, v_n$. Suppose that chords e and f of P do not nest. Without loss of generality, we may assume that $e = v_a v_b$ and $f = v_c v_d$ with $a + 1 < b, c < d - 1$. Let P' denote the v_a–v_d subpath of P.

For each i with $a \leqslant i \leqslant d$, define w_i to be the total weight of the vertices in the component of $T - E(P')$ containing v_i. Define the graph H as the subgraph $G[V(P')]$ with the weight function in which each vertex v_i has weight w_i. By Lemma 2(b), since T is a MAD tree of G, the path P' is a MAD tree of H.

By Lemma 2(a) applied to the tree P' and edge $v_a v_{a+1}$, we have

$$\sigma_{P'-v_a}(v_{a+1}) \leqslant \sigma_{P'-v_a}(v_b).$$

Consider the path $Q = P' - \{v_a, v_d\}$. Since the distance between v_{a+1} and v_d in P is greater than the distance between v_b and v_d, we have from the above inequality

$$\sigma_Q(v_{a+1}) < \sigma_Q(v_b).$$

Now let v_k be a median vertex of Q. Then, by Lemma 1 in conjunction with the above inequality, we have $k < b$. Analogously, we obtain $k > c$.

Without loss of generality we can assume that $\sigma_Q(v_b) \leqslant \sigma_Q(v_c)$. Hence, by the above inequality,

$$\sigma_Q(v_{a+1}) < \sigma_Q(v_c),$$

which contradicts Lemma 1 applied to v_{a+1}, v_c, and v_k. □

Theorem 1. *Let T be a MAD tree of weighted graph G. Then there exists a root: a vertex v_0 such that for all vertices w the (unique) path from v_0 to w in T has no chords.*

Proof. For a vertex v, call an edge $e \in E(G) - E(T)$ a *bad* edge for v if it joins two vertices on a path starting at v (possibly one end is v). Take a vertex v_0 with the minimum number of bad edges, and suppose there are bad edges for v_0. By Lemma 3, the ends of the bad edges are confined to v_0 and one component of $T - v_0$.

Out of all the ends of bad edges, let v_1 be an end nearest to v_0 (possibly $v_1 = v_0$). If the other end of the bad edge is v_3, let v_2 be the first vertex on the v_1–v_3 path in T. We claim that any bad edge for v_2 is also a bad edge for v_0, since otherwise it would not nest with v_1v_3 (and so contradict Lemma 3). But v_1v_3 is not bad for v_2; hence there are fewer bad edges for v_2, a contradiction. □

The above theorem implies in particular that the root has full degree in a MAD tree. Also the path from the root to any vertex is induced in G. Since in a distance-hereditary graph, for any pair of vertices x and y every induced path from x to y has the same length, we obtain

Corollary 1. *In a weighted distance-hereditary graph, every MAD tree is a BFS tree for some vertex.*

This is not true for every graph [7].

3. Basic algorithm

The basic algorithm for finding a MAD tree in a distance-hereditary graph is as follows. We determine, for each vertex c, a BFS tree of G with root c that has minimum average distance (the MAD$_c$-tree). Then we simply select the MAD$_c$-tree of smallest average distance. So the problem we consider is how to find the MAD$_c$-tree.

For any fixed root c, we define the *distance levels* as

$$L_i := \{x|d(x,c) = i\}.$$

Two vertices in L_i are said to be *in the same class* if they are in the same connected component of the subgraph induced by $\bigcup_{j\geqslant i} L_j$. We let k denote the eccentricity of c, i.e., the maximum i for which L_i is not empty.

We make use of the following result due to Hammer and Maffray.

Lemma 4 (Hammer and Maffray [13]). (a) *Any two vertices of L_i in the same class have the same neighbours in L_{i-1}.* (b) *Let C_1 and C_2 be classes of L_i. Then the neighbourhoods of C_1 and C_2 in L_{i-1} are either disjoint or comparable with respect to the subset relation.*

Lemma 5. *Let G be a distance-hereditary graph and T a MAD spanning tree of G with root c of eccentricity k. Let C be a class of L_k such that the neighbourhood N of C in L_{k-1} is minimum. Then there exists a vertex $v \in N$ such that*

(i) *each vertex of C is an end-vertex of T and adjacent to v in T,*
(ii) *each vertex of N except v is an end-vertex of T,*
(iii) *the vertices of N have a common neighbour $w \in L_{k-2}$ in T,*
(iv) *v is a vertex of maximum weight among the vertices of N.*

Proof. (i) and (ii) Since T is a BFS tree with root c, the vertices of C are end-vertices of T. Let x be a vertex of C and let $v \in N$ be its unique neighbour in T. Suppose that some vertex $v' \in N$ has another neighbour $x' \in L_k$ in T. Then x' is in some class C' of L_k (possibly $C = C'$). By Lemma 4 and the choice of C we have $N \subset N_G(C')$. Hence the edges $xv, x'v', xv'$, and $x'v$ are in G. But then the x–x' path $P = x, v, \ldots, v', x'$ in T has two chords xv' and $x'v$ which do not nest. This contradiction to Lemma 3 proves (i) and (ii).

(iii) Let $v_1, v_2 \in N$ and let w_1, w_2 be their respective neighbours in L_{k-2} as given in T. Suppose $w_1 \neq w_2$. Let $P = v_1, w_1, \ldots, w_2, v_2$ be the v_1–v_2 path in T. Since v_1 and v_2 are in the same class, the edges $v_1w_1, v_2w_2, v_1w_2, v_2w_1$ are all in G. Hence P has two chords v_1w_2 and v_2w_1 which do not nest, a contradiction to Lemma 3.

(iv) Suppose there exists a vertex $v' \in N$ of larger weight. By Lemma 4, each vertex in L_k that is adjacent in G to v is also adjacent to v' in G. Then the tree T' obtained from T by making each vertex of L_k that is adjacent to v in T, adjacent to v' in T' is easily seen to have smaller total distance than T, a contradiction. □

This lemma proves that the following algorithm computes a MAD_c-tree T.

Algorithm 1.

1. *Let k be the eccentricity of c. Let C be a class of L_k such that the neighbourhood of C in L_{k-1}, denoted by N_C, is minimum.*
2. *Let v_C be a vertex in N_C of maximum weight. We apply the algorithm recursively to $G' := G - C$, where the weight of v_C is increased by the sum of the weights of the vertices in C. This yields a MAD_c-tree T' of G'.*
3. *We obtain T from T' by attaching the vertices of C to v_C.*

An iterative formulation of the algorithm that also computes the minimum total distance is given below. The method of computing the total distance follows [6]. Note that we speed up the process by simultaneously selecting all classes of L_k that have the same minimum neighbourhood in L_{k-1}.

Algorithm 2.

1. *We initialise the total distance d of the MAD_c-tree by* 0 *and T is set to be empty.*
2. *While G does not consist only of the vertex c, we do the following:*
   (a) *Let k be the eccentricity of c. Let $\tilde{N}$ be the neighbourhood of a vertex $v \in L_k$ in L_{k-1} of minimum size.*
   (b) *Let $C := \{v \in L_k | N(v) \cap L_{k-1} = \tilde{N}\}$.*
   (c) *We select a vertex $v_C \in \tilde{N}$ of maximum weight $w(v_C)$ and add the edges vv_C with $v \in C$ to T.*
   (d) *We add to d:*
      ∘ $(\sum_{v \in C} w(v))^2 - \sum_{v \in C} w(v)^2$ (*the total distance of the vertices in C*)
      ∘ $(\sum_{v \in C} w(v))(\sum_{v \in V \setminus C} w(v))$ (*for each path from C* to *$V \setminus C$,one edge*)

(e) *We add* $\sum_{v \in C} w(v)$ *to* $w(v_C)$ *and delete* C *from* G.

3. *We output* T *as* MAD_c*-tree and* d *as total distance of* T.

We can determine a MAD_c-tree in $\mathrm{O}(n+m)$ time, where n is the number of vertices and m is the number of edges. The overall time complexity to determine a MAD tree is therefore $\mathrm{O}(nm)$.

4. Conversion into a linear-time algorithm

The basic idea to obtain a linear-time algorithm is to perform the above calculations efficiently for all roots c. We make use of the fact that distance-hereditary graphs are exactly the *completely separable* graphs [13], i.e., they are totally split decomposable [4] into stars and cliques.

4.1. Split decompositions and tree structures

A *split* of the graph $G=(V,E)$ is a partition of V into two subsets V_1 and V_2 with at least two elements each, such that all vertices in V_1 that have neighbours in V_2 have the same neighbours in V_2.

Split decomposition is the following recursive procedure.

- If G is not complete and has a split into subsets V_1 and V_2, then form the graph G_1 from G by contracting V_2 to a single vertex v_2, and form the graph G_2 from G by contracting V_1 to a single vertex v_1. Then apply split decomposition to the graphs G_1, G_2.
- If G is complete or does not have a split, then G is called *prime*.

We call the final graphs created by the split decomposition of G the *split components* of G.

Theorem 2 (See, for example, Hammer and Maffray [13]). *A graph is distance-hereditary if and only if all split components are stars or cliques.*

We now define a tree T_G which represents the split decomposition. Start by defining the graph G^{split} as follows. If G is prime then $G^{\mathrm{split}}=G$. Otherwise G^{split} is formed by taking the union of G_1^{split} and G_2^{split} and joining the vertices v_1 and v_2 by a *virtual edge*. Fig. 1 shows the first step in forming G^{split}.

The tree T_G is obtained from G^{split} by the following two steps. First, we make each vertex v of $V(G)$ into an own split component; i.e. we replace the vertex v by a copy v' and then re-introduce a leaf v and join v and v' by a virtual edge. Second, any split component that is a clique (including a K_2) is shrunk to a single vertex (called a *clique-node*). Thus, if E_v is the set of virtual edges of T_G, then each component of $T_G - E_v$ is either a vertex of G, a clique node, or a star with at least three vertices.

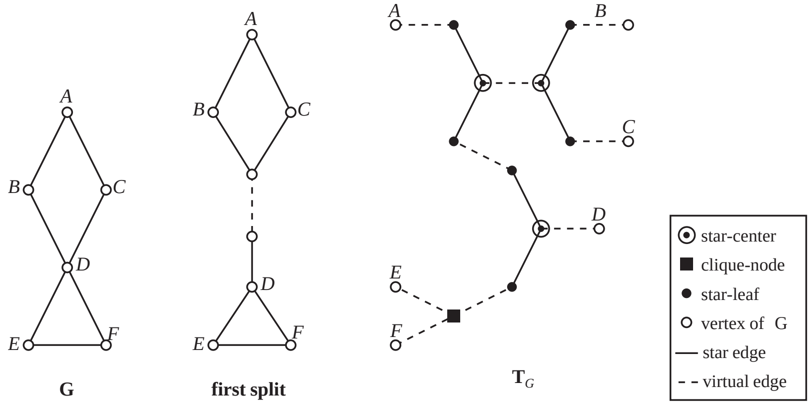

Fig. 1. A distance-hereditary graph and its tree.

Since G^{split} is connected, T_G is connected. It can easily be argued that in G^{split} the number of virtual edges is the number of split components minus one. Thus T_G is indeed a tree (see also [5]).

Now, we label the edges of the remaining split components as *star edges*. Thus the edges of T_G can be classified into two types: virtual and star. Every node of T_G is incident with exactly one virtual edge.

Further, we classify each inner node of T_G as one of three types: a *clique-node* if it resulted from the contraction of a clique split component, a *star-leaf* if it is incident with one star edge, and a *star-centre* if it is incident with more than one star edge. The leaves of T_G are the vertices of G.

An example is shown in Fig. 1. We show a distance-hereditary graph G, the first step in the construction of G^{split}, and the final tree structure T_G. Since split decomposition can be done in linear time, T_G can be determined in linear time [5].

It is clear that G is reconstructible from G^{split} and hence from T_G. The following result shows how to recognise the edges of G from examining the paths in T_G. We define a *separator* on a path in T_G as a consecutive pair of star edges.

Lemma 6. (a) *Two distinct vertices x and y of G are adjacent in G iff the unique path from x to y in T_G has no separator.*

(b) *In general, the distance between x and y in G is one more than the number of separators on the x–y path in T_G.*

Proof. (a) Consider a split during the formation of G^{split}. If adjacent vertices x and y are on different sides of the split, say $x \in V_1$ and $y \in V_2$, then the edge xy is replaced by the path xv_1v_2y. By repeated application it follows that two vertices of $V(G)$ are adjacent in G iff there is a path in G^{split} that alternates between nonvirtual and virtual edges, starting and ending with a nonvirtual edge.

Now, each vertex of $V(G)$ is incident only with nonvirtual edges in G^{split}. Let G^{spmore} denote the result after the first step of transforming G^{split} into T_G. This process simply adds a virtual edge to the start and end of any such path. That is, two vertices of G are adjacent iff there is a path in G^{spmore} that alternates between nonvirtual and virtual edges, starting and ending with a virtual edge.

Note that any path of G^{spmore} that uses vertices of a clique C can be shortened to one which uses only one edge of C (since C is a clique!). In the second step of making T_G, each clique split component of G^{spmore} is contracted to a clique-node. Clearly this cannot create a separator. But by the above comment, this process cannot destroy a separator.

(b) The proof in general is by induction on distance. Assume vertex y is at distance $i+1$ from x and w is a neighbour of y closer to x. By the induction hypothesis, the T_G-path from x to w has $i-1$ separators. Let z be the inner node of T_G, where the y–x and w–x paths first meet. Since the y–w path has no separator, the only way there can be a separator on the T_G-path from x to y that is not on the T_G-path from x to w is that the separator is centred at z. Therefore, there are at most i and therefore exactly i separators on the T_G-path from x to y, as required. □

4.2. Using separating star-centres

We now consider how the original MAD-tree algorithm runs using T_G instead of G. In Algorithm 2, the main problems are to (a) identify the vertices in the level L_k, (b) determine for each such vertex its neighbourhood in L_{k-1}, and (c) adjust T_G to T_{G-C}.

The following definitions provide the key. For a fixed root c, we define:

- *A star-centre x of T_G is a* separating star-centre *if the edge joining x to its parent is a star edge*,
- *for a separating star-centre x, the set $\mathbf{V_x}$ is those vertices w of G such that the T_G-path from x to w starts with the virtual edge on x and contains no separator*,
- *and the set $\mathbf{L_x}$ is those vertices w of G that are descendants of x in T_G such that the T_G-path from x to w starts with a star edge and contains no separator*.

Lemma 7. *For root c of eccentricity k, if $v \in L_k$ and x is the centre of the last separator on the T_G-path from c to v, then $N_G(v) \cap L_{k-1} = V_x$.*

Proof. Since v has maximum distance from c, if $w \in V(G)$ is a descendant of x in T_G and the T_G-path from x to w starts with a star edge, then the path contains no separator. Thus L_x is all $w \in V(G)$ that are descendants of x starting with a star edge.

Say the last separator on the T_G-path from c to v is $\{yx, xz\}$. Let $w \in N_G(v)$. Then $w \in L_x \cup V_x$, since by Lemma 6(a) the T_G-path from v to w cannot contain y (since then it would contain the separator $\{yx, xz\}$). If the lowest common ancestor of v and w in T_G is not x, then w is in L_k, because the T_G-path from c to w then contains all separators found on the T_G-path from c to v. So, $N_G(v) \cap L_{k-1} \subseteq V_x$.

On the other hand, let $w \in V_x$. Then, by Lemma 6(b), w is in L_{k-1}, because the T_G-path from c to w contains all separators found on the T_G-path from c to v except $\{yx, xz\}$. But by Lemma 6(a) and the definitions of L_x and V_x, all of L_x is adjacent to all of V_x. So $V_x \subseteq N_G(v) \cap L_{k-1}$, and the lemma is established. □

By the above lemma, the set $\tilde{N}$ found in step 2(a) of Algorithm 2 can be obtained by considering all separating star-centres x with $k-1$ separators on the T_G-path from x to c , and subject to this choosing the x with V_x of minimum size.

In fact, we can restrict our attention to separating star-centres x such that no descendant is a separating star-centre. For, if x' is a separating star-centre which is a descendant of x, it must lie in the subtree which starts with the virtual edge at x; but then $V_{x'} \subseteq V_x$, and since we are choosing the minimum V the star-centre x can be ignored. We define a *minimally separating star-centre* as a separating star-centre x such that no descendant is a separating star-centre.

It remains to determine the class C. In fact this is given by L_x.

Lemma 8. *For root c of eccentricity k, if x is a minimally separating star-centre such that there are $k-1$ separators on the T_G-path from c to x, then $L_x = \{v \in L_k | N_G(v) \cap L_{k-1} = V_x\}$.*

Proof. By the choice of x, $L_x \subseteq L_k$. So by Lemma 7, if $v \in L_x$ then $N_G(v) \cap L_{k-1} = V_x$.

Now suppose there is a vertex v such that $N_G(v) \cap L_{k-1} = V_x$ but $v \notin L_x$. Then v is the descendant of some separating star-centre x' such that x is a descendant of x'. Since $N_G(v) \cap L_{k-1} = V_{x'}$, it follows that $V_{x'} = V_x$. That means if we contract x, x' and the T_G-path joining x and x' to a single vertex, we still have a valid tree structure for G. It is easy to check that the T_G is in fact minimal, a contradiction. (This situation could arise if one were to split a split component that was already a star.) $\square$

Thus, the following version of the iterative MAD$_c$-tree algorithm is equivalent to Algorithm 2.

Algorithm 3.

1. *We initialise the total distance d of the MAD$_c$-tree by 0 and T is set to be empty.*
2. *While T_G has a separating star-centre*:
   (a) *We select a minimally separating star-centre x with the maximum number of star-pairs on the T_G-path from c to x and subject to this the x with V_x of minimum size.*
   (b) *We select a vertex v_x of V_x with maximum weight $w(v_x)$ and add the edges vv_x with $v \in L_x$ to T.*
   (c) *We add to d*:
      ∘ $(\sum_{v \in L_x} w(v))^2 - \sum_{v \in L_x} w(v)^2$ (*the total distance of the vertices in L_x*)
      ∘ $(\sum_{v \in L_x} w(v))(\sum_{v \in V \setminus L_x} w(v))$ (*for each path from C to $V \setminus L_x$, one edge*)
   (d) *We add $\sum_{v \in L_x} w(v)$ to $w(v_x)$ and delete from T_G the set L_x as well as the inner vertices of the path from each $v \in L_x$ to x.*
3. *The parent of all remaining vertices is c and we add to d*:
   ∘ $(\sum_{v \neq c} w(v))^2 - \sum_{v \neq c} w(v)^2$
   ∘ $(\sum_{v \neq c} w(v))w(c)$

4.3. Traversal calculations

The next step in the conversion to a linear-time algorithm is to introduce a sextet of functions which can be calculated in linear time by performing a postorder traversal.

We define the parameters first for separating star-centres.

For a fixed root c and separating star-centre x, define

$$Sum_x = \sum_{v \in L_x} w(v),$$

$$SQuare_x = \sum_{v \in L_x} w(v)^2,$$

$$M_x = \underset{v \in V_x}{Max}\ w(v),$$

$$S_x = \sum_{v \in V_x} w(v),$$

and

$$SQ_x = \sum_{v \in V_x} w(v)^2,$$

where the weights $w(v)$ are those that hold *after* processing x and all its descendants that are separating star-centres. The parameter, d_x is the value of d after processing x.

The key point is that when we process the separating star-centre x, we add to d in step 2(c) of Algorithm 3:

$$Sum_x^2 - SQuare_x + Sum_x(S - Sum_x),$$

where $S = \sum_{v \in V} w(v)$ is the sum of all weights in G.

Recall that we defined the set V_x for a separating star-centre. We extend the definition as follows: if x is a leaf then $V_x = \{x\}$; if x is an inner node but not a separating star-centre, then $V_x = \bigcup_{y \prec x} V_y$, where we use the notation $y \prec x$ to mean y is a child of x.

The following is easily verified.

Lemma 9. *If x is a separating star-centre such that $y_1, \ldots, y_l$ are the children of x joined by a star edge and y is the child of x joined by a virtual edge, then*

$$V_x = V_y \quad \text{and} \quad L_x = \bigcup_{i=1}^{l} V_{y_i}.$$

Thus, the definitions of M_x, S_x and SQ_x are immediately generalised to all nodes x. We generalise d_x by defining it as 0 for leaves, and as the sum of the values of the children for the remaining nodes. The following lemma shows how the values can be calculated by postorder traversal.

Lemma 10. *Let x be a node of T_G.*

(a) *If x is a leaf, then $d_x = 0$, $S_x = M_x = w(x)$, and $SQ_x = w(x)^2$.*

(b) *If x is an inner node, but neither c nor a separating star-centre, then S_x, SQ_x, and d_x are the sum of the values over the children of x. The value M_x is the maximum over the children.*

(c) *If x is a separating star-centre, such that $y_1, \ldots, y_l$ are the children of x joined by a star edge and y is the child of x joined by a virtual edge, then*

$$Sum_x = \sum_{i=1}^{l} S_{y_i},$$

$$SQuare_x = \sum_{i=1}^{l} SQ_{y_i},$$

$$S_x = S_y + \sum_{i=1}^{l} S_{y_i},$$

$$M_x = M_y + \sum_{i=1}^{l} S_{y_i},$$

$$SQ_x = SQ_y - M_y^2 + M_x^2,$$

$$d_x = d_y + \sum_{i=1}^{l} d_{y_i} + Sum_x^2 - SQuare_x + Sum_x(S - Sum_x).$$

(d) *If $x = c$ and y is the unique child of x in T_G, then $d_c = d_y + S_y^2 - SQ_y + w(c)S_y$.*

Proof. Part (a) is trivial. Part (b) is immediate from Lemma 9.

(c) The formulas for Sum_x and $SQuare_x$ also follow immediately from Lemma 9.

Now, suppose the algorithm processes x. The algorithm selects a vertex $v_x \in V_x$ of maximum weight. It adds the weights of all vertices of L_x—which is $\sum_{i=1}^{l} S_{y_i}$—to $w(v_x)$. Since $V_x = V_y$, the old value S_y is incremented by the increase in the weight of $w(v_x)$, and thus the formula for S_x holds.

Vertex v_x remains a vertex of maximum weight in V_x after the processing of x, and therefore $M_x = M_y + \sum_{i=1}^{l} S_{y_i}$. One also can observe that during processing x, v_x is the only vertex with changing weight. The old weight of v_x is M_y. Therefore $SQ_x = SQ_y - M_y^2 + M_x^2$.

Part (d) follows from step 3 of Algorithm 3. □

We can reformulate the algorithm as follows to save computation time. Throughout, if x is a separating star-centre, then $y_1, \ldots, y_l$ are the children of x joined by a star edge and y is the child of x joined by a virtual edge.

Algorithm 4.

1. *We compute S_x by postorder traversal*:
   - *For leaf x, $S_x = w(x)$.*
   - *For nonleaf x, $S_x = \sum_{y \prec x} S_y$*

2. *We compute* M_x *by postorder traversal*:
   - *For leaf* x, $M_x = w(x)$.
   - *If* x *is not a separating star-centre, then* $M_x = Max_{y \prec x} M_y$.
   - *If* x *is a separating star-centre, then* $M_x = M_y + S_x - S_y$
3. *We compute* SQ_x *by postorder traversal*:
   - *For leaf* x, $SQ_x = w(x)^2$.
   - *If* x *is not a separating star-centre, then* $SQ_x = \sum_{y \prec x} SQ_y$.
   - *If* x *is a separating star-centre, then* $SQ_x = SQ_y - M_y^2 + M_x^2$.
4. *We compute, for all separating star-centres* x, Sum_x *and* $SQuare_x$ *by postorder traversal.*
5. *We compute* d_x *by postorder traversal*:
   - *If* x *is a leaf, the* $d_x = 0$.
   - *If* x *is not a separating star-centre and different from* c, *then* $d_x = \sum_{y \prec x} d_y$.
   - *If* x *is a separating star-centre, then* $d_x = \sum_{y \prec x} d_y + Sum_x^2 - SQuare_x + Sum_x(S - Sum_x)$.
   - *If* $x=c$ *and* y *is the unique child of* x *in* T_G, *then* $d_c = d_y + S_y^2 - SQ_y + w(c)S_y$.
6. *Output* d_c.

4.4. Calculation of parameters for all parents

For a directed edge $f = (y,x)$, we define S_f, M_f, SQ_f, Sum_f, $SQuare_f$, and d_f as the S_x, M_x, SQ_x, Sum_x, $SQuare_x$, and d_x that we get if y is the parent of x, i.e., the root c belongs to the component of $T_G - xy$ containing y.

We will continue with a fixed root c. The above algorithm calculates in linear time the values for all edges (y,x) where y is the parent of x. The final phase is to calculate the values for all (y,x) where x is the parent of y. This is achieved in a preorder traversal starting at c.

We have to be careful when we say a node is a separating star-centre. We define (y,x) as a *separating star-centre* if x is a separating star-centre under the assumption that T_G is rooted at a node in the component of $T_G - xy$ containing y. That means (y,x) is a separating star-centre if and only if xy is a star edge of T_G and x a star-centre.

Now, consider a directed edge (y,x) of T_G such that x is the parent of y and assume that x is not the root so that x has a parent z. The idea we exploit is the following. Consider Fig. 2 where A is the component of $T_G - xz$ containing z, C is the component of $T_G - xy$ containing y, and B is the remainder of T_G. For any parameter ψ, the value $\psi_{(y,x)}$ considers the subtree containing $A \cup B$, the value $\psi_{(z,x)}$ the subtree containing A, the value $\psi_{(x,z)}$ the subtree containing $B \cup C$, and $\psi_{(x,y)}$ the subtree containing C. Thus, we have the symbolic equation:

$$(y,x) = (z,x) + (x,z) - (x,y).$$

The values $\psi_{(z,x)}$ and $\psi_{(x,y)}$ were calculated in the previous phase, and we have already calculated $\psi_{(x,z)}$ since we are performing a preorder traversal.

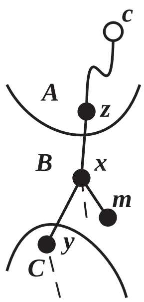

Fig. 2. A partition of T_G.

Lemma 11.

(a) $S_{(y,x)} = S_{(z,x)} + S_{(x,z)} - S_{(x,y)}$.
(b) *If neither* (y,x) *nor* (z,x) *is a separating star-centre, then* $SQ_{(y,x)} = SQ_{(z,x)} + SQ_{(x,z)} - SQ_{(x,y)}$ *and* $d_{(y,x)} = d_{(z,x)} + d_{(x,z)} - d_{(x,y)}$.
(c) *If* (y,x) *is a separating star-centre and* wx *is the virtual edge on* x (*possibly* $w=z$), *then* $SQ_{(y,x)} = SQ_{(x,w)} - M^2_{(x,w)} + M^2_{(y,x)}$ *and* $M_{(y,x)} = M_{(x,w)} + S_{(y,x)} - S_{(x,w)}$.
(d) *If* (y,x) *is a separating star-centre and* xz *is not the virtual edge on* x, *then* $Sum_{(y,x)} = Sum_{(z,x)} + S_{(x,z)} - S_{(x,y)}$, $SQuare_{(y,x)} = SQuare_{(z,x)} + SQ_{(z,x)} - SQ_{(x,y)}$, *and* $d_{(y,x)} = d_{(z,x)} + d_{(x,z)} - d_{(x,y)} + Sum^2_{(y,x)} - Sum^2_{(z,x)} - SQuare_{(y,x)} + SQuare_{(z,x)} + Sum_{(y,x)}(S - Sum_{(y,x)}) - Sum_{(z,x)}(S - Sum_{(z,x)})$.
(e) *If* (y,x) *is a separating star-centre and* xz *is the virtual edge on* x, *then* $Sum_{(y,x)} = S_x - S_y$, $SQuare_{(y,x)} = SQ_x - SQ_y$ *and* $d_{(y,x)} = d_{(z,x)} + d_{(x,z)} - d_{(x,y)} + Sum^2_{(y,x)} - SQuare_{(y,x)} + Sum_{(y,x)}(S - Sum_{(y,x)})$.

Proof. Note that when we change the root from c to some descendant c' of y, then y becomes the parent of x and z becomes a child of x.

(a) We can rewrite the equation $S_x = \sum_{y \prec x} S_y$ into $S_{(y,x)} = \sum_{w \sim x;\ w \neq y} S_{(x,w)}$, where we use $w \sim x$ to mean w is a neighbour of x in T_G. The equation for S follows.
(b) By the same argument as (a).
(c) We rewrite the equations $SQ_x = SQ_w - M^2_w + M_x$ and $M_x = M_w + S_x - S_w$ (y is replaced by w).
(d) Then (z,x) is a separating star-centre. Note that $Sum_{(z,x)} = Sum_x$ and $SQuare_{(z,x)} = SQuare_x$ are already defined. When we make y the parent of x, in Sum_x ($SQuare_x$), $S_{(x,z)}$ ($SQ_{(x,z)}$) is added and ($S_{(x,y)}$) ($SQ_{(x,y)}$) is subtracted, because z becomes a child of x that is joined by x by a nonvirtual edge.

    $d_x = \sum_{y \prec x} d_y + Sum^2_x - SQuare_x + Sum_x(S - Sum_x)$ transforms into the equation $d_{(y,x)} = \sum_{w \sim x;\ w \neq y} d_{(x,w)} + Sum^2_{(y,x)} - SQuare_{(y,x)} + Sum_{(y,x)}(S - Sum_{(y,x)})$. If we compare $d_{(y,x)}$ and $d_{(z,x)}$, we get the desired formula.
(e) So z becomes the child of x joined by a virtual edge and y becomes the parent. $Sum_{(y,x)}$ ($SQuare_{(y,x)}$) is the sum over all $S_{(x,y')}$ ($SQ_{(x,y')}$) with $y' \neq y$ and $y' \neq z$. Therefore $Sum_{(y,x)} = S_x - S_y$ and $SQuare_{(y,x)} = SQ_x - SQ_y$.

$d_{(y,x)}=\sum_{w\sim x;\ w\neq y} d_{(x,w)}+S^2_{(y,x)}-SQ_{(y,x)}+S_{(y,x)}(S-S_{(y,x)})$ and $d_{(z,x)}=\sum_{w\sim x;\ w\neq z} d_{(x,w)}$. The equation for $d_{(y,x)}$ follows. □

Therefore all of $S_{(y,x)}$, $SQ_{(y,x)}$, and $d_{(y,x)}$ can be expressed by closed formulas if (z,x) is not a separating star-centre or if (y,x) is a separating star-centre (and therefore computed by one time-unit). Since M_x can be determined by a closed formula if x is a separating star-centre, $M_{(y,x)}$ can be determined by a closed formula if (y,x) is a separating star-centre. It remains (i) to get $M_{(y,x)}$ for the case that (y,x) is not a separating star-centre and (ii) to cover the case that (z,x) is a separating star-centre and (y,x) is not (i.e. yx is the virtual edge on x).

These cases are easily handled, and the resultant formulas are incorporated in the algorithm below. (These formulas are the reason for the introduction of the parameters M', SQ' and d' calculated in steps 2 and 3.)

Algorithm 5.

1. *We compute S_x, M_x, SQ_x, and d_x, for all nodes x of T_G; and Sum_x and $SQuare_x$, for all star-centres x.*
2. *For each node x that is not a separating star-centre, let m_x be a child of x, such that $M_x = M_{m_x}$ and $M'_x = Max_{w\neq m_x \text{ child of } x} M_w$.*
3. *For a node x that is a separating star-centre, let m_x be the child of x joined by a virtual edge and let $M'_x = Max_{w\neq m_x \text{ child of } x} M_x$, $SQ'_x = \sum_{w\neq m_x \text{ child of } x} SQ_x$, and $d'_x = \sum_{w\neq m_x \text{ child of } x} d_x$.*
4. *For each x of T_G with parent y, let $S_{(y,x)} = S_x$, $M_{(y,x)} = M_x$, $SQ_{(y,x)} = SQ_x$, and $d_{(y,x)} = d_x$; and for each star-centre x with parent y, let $Sum_{(y,x)} = Sum_x$ and $SQuare_{(y,x)} = SQuare_x$.*
5. *Let $y_1,\dots,y_p$ be a preorder enumeration of T_G (it starts with the root and each initial segment induces a tree). For $i = 2,\dots,p$, let x_i be the parent of y_i, and for $i \geqslant 3$ let z_i be the parent of x_i.*
6. *$d_{(y_2,x_2)} = 0$, $S_{(y_2,x_2)} = M_{(y_2,x_2)} = w(x_2)$, and $SQ_{(y_2,x_2)} = w(x_2)^2$ ($x_2 = y_1$ is the root).*
7. *Calculate S using preorder traversal:*

$$S_{(y_i,x_i)} = S_{(z_i,x_i)} + S_{(x_i,z_i)} - S_{(x_i,y_i)}.$$

8. *Calculate M using preorder traversal:*
   - *If (y_i,x_i) is a separating star-centre, then $M_{(y_i,x_i)} = M_{(x_i,w)} + S_{(y_i,x_i)} - S_{(x_i,w)}$ where x_iw is the virtual edge on x_i.*
   - *If (y_i,x_i) is not a separating star-centre but (z_i,x_i) is, then $M_{(y_i,x_i)} = Max(M'_x, M_{(x_i,z_i)})$.*
   - *If neither (y_i,x_i) nor (z_i,x_i) is a separating star-centres, then: If $y_i = m_{x_i}$ then $M_{(y_i,x_i)} = Max(M'_x, M_{(x_i,z_i)})$. Otherwise $M_{(y_i,x_i)} = Max(M_x, M_{(x_i,z_i)})$.*
9. *Calculate SQ using preorder traversal:*
   - *If (y_i,x_i) is a separating star-centre and w is the neighbour of x_i joined by a virtual edge, then $SQ_{(y_i,x_i)} = SQ_{(x_i,w)} - M^2_{(x_i,w)} + M^2_{(y_i,x_i)}$*

- *If* (y_i, x_i) *is not a separating star-centre but* (z_i, x_i) *is, then* $SQ_{(y_i,x_i)} = SQ'_{x_i} + SQ_{(x_i,z_i)}$.
- *If neither* (y_i, x_i) *nor* (z_i, x_i) *is a separating star-centre, then* $SQ_{(y_i,x_i)} = SQ_{(z_i,x_i)} + SQ_{(x_i,z_i)} - SQ_{(x_i,y_i)}$.

10. *Calculate Sum, SQuare and d using preorder traversal*:
    - *If* (y_i, x_i) *is a separating star-centre and* z_i *is not the neighbour of* x_i *joined by a virtual edge, then* $Sum_{(y_i,x_i)} = Sum_{(z_i,x_i)} + S_{(x_i,z_i)} - S_{(x_i,y_i)}$, $SQuare_{(y_i,x_i)} = SQuare_{(z_i,x_i)} + SQ_{(x_i,z_i)} - SQ_{(x_i,y_i)}$, $d_{(y_i,x_i)} = d_{(z_i,x_i)} + d_{(x_i,z_i)} - d_{(x_i,y_i)} + Sum^2_{(y_i,x_i)} - Sum^2_{(z_i,x_i)} - SQuare_{(y_i,x_i)} + SQuare_{(z_i,x_i)} + Sum_{(y_i,x_i)}(S - Sum_{(y_i,x_i)}) - Sum_{(z_i,x_i)}(S - Sum_{(z_i,x_i)})$.
    - *If* (y_i, x_i) *is a separating star-centre and* z_i *is the neighbour of* x_i *joined by a virtual edge, then* $Sum_{(y_i,x_i)} = S_{x_i} - S_{y_i}$, $SQuare_{(y_i,x_i)} = SQ_{x_i} - SQ_{y_i}$, $d_{(y_i,x_i)} = d_{(z_i,x_i)} + d_{(x_i,z_i)} - d_{(x_i,y_i)} + Sum^2_{(y_i,x_i)} - SQuare_{(y_i,x_i)} + Sum_{(y_i,x_i)}(S - Sum_{(y_i,x_i)})$
    - *If* (y_i, x_i) *is not a separating star-centre and* (z_i, x_i) *is, then* $d_{(y_i,x_i)} = d'_{x_i} + d_{(x_i,z_i)}$.
    - *If neither* (y_i, x_i) *nor* (z_i, x_i) *is separating star-centre, then* $d_{(y_i,x_i)} = d_{(z_i,x_i)} + d_{(x_i,z_i)} - d_{(x_i,y_i)}$.
11. *For all vertices* v *of* G, *let* v' *be the node of* T_G *adjacent with* v *in* T_G *and let* $d_v = S^2_{(v,v')} - SQ_{(v,v')} + w(v)S_{(v,v')}$.
12. *Select a vertex* c *of* G, *such that* d_c *is minimum.*
13. *Determine a* MAD_c*-tree of* G.

It is easily seen that all steps with the exception of the last one run in $O(n)$ time. As we have seen in the previous section, the last step runs in $O(n+m)$ time. We can run the last step also in $O(n)$ time:

Algorithm 6.

1. *We root* T_G *at* c.
2. *For each* x *with parent* y, *we determine a descendant vertex* $m'_x \in V_x$, *such that* $w(m'_x) = M_x$.
3. *For each vertex* v *of* G *different from* c, *we determine the first ancestor* x *of* v *that is a separating star-centre (with respect to the rooting from* c*). The parent of* v *in the MAD-tree* T *is* m_x. *If* v *has no ancestor that is a separating star-centre then the parent of* v *is* c.

The final result of this section is the following.

Theorem 3. *Let* G *be a connected weighted distance-hereditary graph. If the split decomposition of* G *is known, then a MAD-tree of* G *can be determined in* $O(n)$ *time.*

Corollary 2. *Let* G *be a connected weighted distance-hereditary graph. Then a MAD-tree of* G *can be determined in* $O(n)$ *time.*

5. Conclusion

Since the MAD-tree problem is NP-complete for general graphs, the question for which graph classes polynomial algorithms exist arises naturally. Possible candidates are strongly chordal graphs and graphs of bounded clique width (the latter graph class contains the distance-hereditary graphs as well as the outerplanar graphs). The general approach of Courcelle, et al. [3] is not applicable for the case of MAD trees (the natural formulation does not match the definition scheme as stated in [3]). An interesting generalization of the MAD-tree problem is stated in [12]: Given a connected graph G and a subset of the vertex set, the set of sources, find a tree that minimises the distances from any source to any other vertex.

Acknowledgements

Financial support by the University of Natal and the National Research Foundation is gratefully acknowledged.

References

[1] H.-J. Bandelt, H.M. Mulder, Distance-hereditary graphs, J. Combin. Theory Ser. B 41 (1986) 182–208.
[2] C.A. Barefoot, R.C. Entringer, L.A. Székely, Extremal values for ratios of distances in trees, Discrete Appl. Math. 80 (1997) 37–56.
[3] B. Courcelle, J.A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Systems 33 (2000) 125–130.
[4] W.H. Cunningham, Decomposition of directed graphs, SIAM J. Algebraic Discrete Methods 3 (1982) 214–228.
[5] E. Dahlhaus, Efficient parallel and linear time sequential split decomposition, FST-TCS 94, Madras.
[6] P. Dankelmann, Computing the average distance of an interval graph, Inform Process. Lett. 48 (1993) 311–314.
[7] P. Dankelmann, A note on MAD spanning trees, J. Combin. Math. Combin. Comput. 32 (2000) 93–95.
[8] P. Dankelmann, P. Slater, Average distance in outerplanar graphs, preprint.
[9] A. D'Atri, M. Moscarini, Distance-hereditary graphs, Steiner trees, and connected domination, SIAM J. Comput. 17 (1988) 521–538.
[10] R.C. Entringer, Distance in graphs: trees, J. Combin. Math. Combin. Comput. 24 (1997) 65–84.
[11] R.C. Entringer, D.J. Kleitman, L.A. Székely, A note on spanning trees with minimum average distance, Bull. Inst. Combin. Appl. 17 (1996) 71–78.
[12] A.M. Farley, P. Fragopoulou, D. Krumme, A. Proskurowski, D. Richards, Multi-source spanning tree problems, in: C. Gavoile, J.-C. Bermond, A. Raspaud (Eds.), Proceedings of SIROCCO'99 Lucanan-Océan, (Proceedings in Informatics, Vol. 5), Carleton Scientific, pp. 126–136.
[13] P. Hammer, F. Maffray, Completely separable graphs, Discrete Appl. Math. 27 (1990) 85–99.
[14] E. Howorka, A characterization of distance-hereditary graphs, Quart. J. Math. Oxford 28 (1977) 417–420.
[15] D.S. Johnson, J.K. Lenstra, A.H.G. Rinnooy-Kan, The complexity of the network design problem, Networks 8 (1978) 279–285.
[16] B.Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, C.Y. Tang, A polynomial-time approximation scheme for minimum routing cost spanning trees, SIAM J. Comput. 29 (2000) 761–778.