
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 10, no. 2, pp. 97–122 (2006)

Collective Tree Spanners and Routing in AT-free Related Graphs

Feodor F. Dragan Chenyu Yan

Department of Computer Science
Kent State University
Kent, Ohio, USA
{dragan,cyan}@cs.kent.edu

Derek G. Corneil

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
dgc@cs.toronto.edu

Abstract

In this paper we study collective additive tree spanners for families of graphs that either contain or are contained in AT-free graphs. We say that a graph $G = (V, E)$ *admits a system of μ collective additive tree r-spanners* if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices x, y of G a spanning tree $T \in \mathcal{T}(G)$ exists such that $d_T(x, y) \leq d_G(x, y) + r$. Among other results, we show that AT-free graphs have a system of two collective additive tree 2-spanners (whereas there are trapezoid graphs that do not admit any additive tree 2-spanner). Furthermore, based on this collection, we derive a compact and efficient routing scheme. Also, any DSP-graph (there exists a dominating shortest path) admits an additive tree 4-spanner, a system of two collective additive tree 3-spanners and a system of five collective additive tree 2-spanners.

Article Type	Communicated by	Submitted	Revised
regular paper	S. Khuller	January 2005	January 2006

Results of this paper were partially presented at the WG'04 conference [5].

1 Introduction

Given a graph $G = (V, E)$, a spanning subgraph H is called a *spanner* if H provides a "good" approximation of the distances in G. More formally, for $t \geq 1$, H is called a *multiplicative t–spanner* of G [2, 18, 17] if $d_H(u, v) \leq t \cdot d_G(u, v)$ for all $u, v \in V$. If $r \geq 0$ and $d_H(u, v) \leq d_G(u, v) + r$ for all $u, v \in V$, then H is called an *additive r–spanner* of G [12]. The parameters t and r are called, respectively, the *multiplicative* and the *additive stretch factors*. Clearly, every additive r-spanner of G is a multiplicative $(r + 1)$-spanner of G (but not vice versa). In this paper, we continue the approach taken in [6] of studying *collective tree spanners*. We say that a graph $G = (V, E)$ *admits a system of μ collective additive tree r-spanners* if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices x, y of G a spanning tree $T \in \mathcal{T}(G)$ exists such that $d_T(x, y) \leq d_G(x, y) + r$ (a multiplicative variant of this notion can be defined analogously). Clearly, if G admits a system of μ collective additive tree r-spanners, then G admits an additive r-spanner with at most $\mu \times (n-1)$ edges (take the union of all those trees), and if $\mu = 1$ then G admits an additive tree r-spanner. Note also that any graph on n vertices admits a system of at most $n - 1$ collective additive tree 0-spanners (take $n - 1$ Breadth-First-Search–trees rooted at different vertices of G). In particular, we examine the problem of finding small systems of additive r-spanners for small values of r on classes of graphs that are related to the well known *asteroidal triple-free (AT-free) graphs*, notably the restricted families: permutation graph and trapezoid graphs, and the generalizations: DSP-graphs and graphs with bounded asteroidal number. (All graph class definitions appear in Subsection 1.2.)

Once one has determined a system of collective additive tree spanners, it is interesting to see if such a system can be used to design compact and efficient routing schemes for the given graph. Following [16], one can give the following formal definition. A family $\Re$ of graphs is said to have an $l(n)$-bit *routing labeling scheme* if there is a function L labeling the vertices of each n-vertex graph in $\Re$ with distinct labels of up to $l(n)$ bits, and there exists an efficient algorithm, called the *routing decision*, that given the label of a source vertex v and the label of the destination vertex (the header of the packet), decides in time polynomial in the length of the given labels and using only those two labels, whether this packet has already reached its destination, and if not, to which neighbor of v to forward the packet. The quality of a routing scheme is measured in terms of its *additive stretch*, called *deviation*, (or *multiplicative stretch*, called *delay*), namely, the maximum surplus (or ratio) between the length of a route, produced by the scheme for some pair of vertices, and their distance.

1.1 Our results

After introducing the notation and definitions used throughout the paper, we examine various families of graphs related to AT-free graphs from the perspective of determining whether they have a small constant number of collective additive tree c-spanners for small constant c. In Section 2 we show that AT-free graphs

have a system of two collective additive tree 2-spanners, permutation graphs have a single additive tree 2-spanner but there are trapezoid graphs that do not admit any additive tree 2-spanner (thereby disproving a conjecture of [19]). All of these tree spanners can be easily constructed in linear time. For families that strictly contain AT-free graphs we prove that any DSP-graph admits one additive tree 4-spanner, a system of two collective additive tree 3-spanners and a system of five collective additive tree 2-spanners. Furthermore, any graph G with asteroidal number $\mathsf{an}(G)$ admits a system of $\mathsf{an}(G)(\mathsf{an}(G)-1)/2$ collective additive tree 4-spanners and a system of $\mathsf{an}(G)(\mathsf{an}(G)-1)$ collective additive tree 3-spanners.

In Section 3, we show how the system of two collective additive tree 2-spanners for AT-free graphs can be used to derive a compact and efficient routing scheme. In particular we show that any AT-free graph with diameter D and maximum vertex degree Δ admits a $(3\log_2 D+6\log_2 \Delta+3)$-bit routing labeling scheme of derivation at most 2. Moreover, the scheme is computable in linear time, and the routing decision is made in constant time per vertex. The routing scheme for AT-free graphs can be adapted to permutation graphs, DSP-graphs and to graphs with bounded asteroidal number.

1.2 Basic notions and notation

All graphs occurring in this paper are connected, finite, and simple. In a graph $G=(V,E)$ the *length* of a path from a vertex v to a vertex u is the number of edges in the path. The *distance* $d_G(u,v)$ between the vertices u and v is the length of a shortest path connecting u and v. The *eccentricity* $ecc(v)$ of a vertex v of G is $\max_{u\in V} d_G(u,v)$. The diameter $diam(G)$ of G is $\max_{v\in V} ecc(v)$. The *ith neighborhood* of a vertex v of G is the set $N_i(v):=\{u\in V : d_G(v,u)=i\}$. For a vertex v of G, the sets $N(v):=N_1(v)$ and $N[v]:=N(v)\cup\{v\}$ are called the *open neighborhood* and the *closed neighborhood* of v, respectively. For a set $S\subseteq V$, by $N[S]:=\bigcup_{v\in S} N[v]$ we denote the *closed neighborhood* of S and by $N(S):=N[S]\setminus S$ the *open neighborhood* of S. A set $D\subseteq V$ is called a *dominating set* of a graph $G=(V,E)$ if $N[D]=V$.

An independent set of three vertices such that each pair is joined by a path that avoids the neighborhood of the third is called an *asteroidal triple*. A graph G is an *AT-free graph* if it does not contain any asteroidal triples [3]. In [11], the notion of asteroidal triple was generalized. An independent set $A\subseteq V$ of a graph $G=(V,E)$ is called an *asteroidal set* of G if for each $a\in A$ the vertices of $A\setminus\{a\}$ are contained in one connected component of $G-N[a]$, the graph obtained from G by removing vertices of $N[a]$. The maximum cardinality of an asteroidal set of G is denoted by $\mathsf{an}(G)$, and called the *asteroidal number* of G. The class of *graphs of bounded asteroidal number* extends naturally the class of AT-free graphs; AT-free graphs are exactly the graphs with asteroidal number at most two.

Let P be a shortest u,v-path of G, for some pair of vertices u,v. If every vertex z of G belongs to the neighborhood $N[P]$ of P, then we say that P is a *dominating shortest path* of G. A graph G is called a *Dominating-Shortest-*

Path–graph (or *DSP–graph*, for short), if it has a dominating shortest path. By the Dominating Pair Theorem given in [3], any AT-free graph is a DSP-graph.

The class of AT-free graphs contains many intersection families of graphs, among them the permutation graphs, the trapezoid graphs and the cocomparability graphs. These three families of graphs can be defined as follows [1, 9]. Consider two parallel lines (upper and lower) in the plane. Assume that each line contains n points, labeled 1 to n, and each two points with the same label define a segment with that label. The intersection graph of such a set of segments between two parallel lines is called a *permutation graph*. Assume now that each line contains n intervals, labeled 1 to n, and each two intervals with the same label define a trapezoid with that label (a trapezoid can degenerate to a triangle or to a segment). The intersection graph of such a set of trapezoids between two parallel lines is called a *trapezoid graph*. Clearly, every permutation graph is a trapezoid graph, but not vice versa. The class of cocomparability graphs (which contains all trapezoid graphs as a subclass) can be defined as the intersection graphs of continuous function diagrams, but for this paper it is more convenient to define them via the existence of a special vertex ordering. A graph G is a *cocomparability graph* if it admits a vertex ordering $\sigma = [v_1, v_2, \ldots, v_n]$, called a *cocomparability ordering*, such that for any $i < j < k$, if v_i is adjacent to v_k then v_j must be adjacent to at least one of v_i, v_k. According to [15], such an ordering of a cocomparability graph can be constructed in linear time. Note also that, given a permutation graph G, a *permutation model* (i.e., a set of segments between two parallel lines, defining G) can be found in linear time [15]. A *trapezoid model* for a trapezoid graph can be found in $O(n^2)$ time [13].

Lexicographic breadth-first search (LexBFS), started at a vertex u, orders the vertices of a graph by assigning numbers from n to 1 in the following way. The vertex u gets the number n. Then each next available number k is assigned to a vertex v (as yet unnumbered) which has lexically largest vector $(s_n, s_{n-1}, \ldots, s_{k+1})$, where $s_i = 1$ if v is adjacent to the vertex numbered i, and $s_i = 0$ otherwise. Vertex numbered 1 is called *last visited vertex*.

2 Collective additive tree spanners

2.1 AT-free graphs

It is known [19] that any AT-free graph admits one additive tree 3-spanner. In this subsection we show that any AT-free graph admits a system of two collective additive tree 2-spanners.

As a consequence of the Dominating Pair Theorem given in [3], any AT-free graph has a dominating shortest path that can be found in linear time by $2 \times LexBFS$ [4]. The $2 \times LexBFS$ method first starts a *lexicographic breadth-first search* (*LexBFS*) from an arbitrary vertex x of G and then starts a second LexBFS from the vertex x_0 last visited by the first LexBFS. Let x_l be the vertex of G last visited by the second LexBFS. As shown in [4], every shortest path $(x_0, x_1, \ldots, x_l)$, connecting x_0 and x_l, is a dominating shortest path of G. Next

we demonstrate how to use such a dominating shortest path in an AT-free graph to show that every AT-free graph admits a system of two collective additive tree 2-spanners. We will need the following result from [10].

Lemma 1 [10] *Let $P := (x_0, x_1, \ldots, x_l)$ be a dominating shortest path of an AT-free graph $G = (V, E)$ constructed by $2 \times LexBFS$. Then, for every $i = 1, 2, \ldots, l$, every vertex $z \in N_i(x_0)$ is adjacent to x_i or x_{i-1}.*

Using this lemma, we construct a first spanning tree $T_1 = (V, E_1)$ for an AT-free graph $G = (V, E)$ as follows: put into initially empty E_1 all edges of the path $P := (x_0, x_1, \ldots, x_l)$, and then for each vertex $z \in N_i(x_0)$, put edge zx_{i-1} into E_1, if z is adjacent to x_{i-1} in G, and put edge zx_i into E_1, otherwise. We call this spanning tree the *caterpillar-tree* of G (with *spine* P). According to [19], this caterpillar-tree gives already an additive tree 3-spanner for the AT-free graph G. To get a collective additive stretch factor 2 for G, we construct a second spanning tree $T_2 = (V, E_2)$ for G as follows. Set $L_i := N_i(x_0)$ for each $i = 1, 2, \ldots, l$.

set $E_2 := \{$all edges of the path $P := (x_0, x_1, \ldots, x_l)\}$;
set $dev(x_i) := 0$ for each vertex x_i of the path P;
for $i = 1$ to l do
  for each vertex $z \in L_i \setminus \{x_i\}$ do
    among all neighbors of z in L_{i-1} choose a neighbor w with minimum deviation $dev(w)$;
    add edge zw to E_2 and set $dev(z) := dev(w) + 1$;
  enddo
enddo.

We call spanning tree T_2 the *cactus-tree* of G (with *stem* P). It is evident, by construction, that the cactus-tree T_2 is a special kind of *breadth-first-search–tree* of G. The value $dev(z)$ (called the *deviation of z from stem P*) gives the distance in T_2 between vertex z and path P. In Figure 1 we show an AT-free graph G along with its caterpillar-tree T_1 and cactus-tree T_2.

Lemma 2 *Spanning trees $\{T_1, T_2\}$ are collective additive tree 2-spanners of AT-free graph G.*

Proof: Consider two arbitrary vertices $x \in L_i$ and $y \in L_j$ $(y \neq x)$ of G, where $j \leq i$. If $i = j$, i.e., both x and y lie in the same layer $L_i = L_j$, then the distance in T_1 between x and y is at most 3, since in the worst case one of them is adjacent to x_i in T_1 and the second to x_{i-1}. Thus, $d_{T_1}(x, y) \leq 3 \leq d_G(x, y)+2$ holds when $i = j$, and therefore, we may assume that $i > j$.

We know that $d_G(x, y) \geq i - j$. By the construction of the caterpillar-tree T_1, we have $d_{T_1}(y, x_j) \leq 2$ and $d_{T_1}(x, x_{i-1}) \leq 2$. Hence, $d_{T_1}(x, y) \leq d_{T_1}(x, x_{i-1}) + d_{T_1}(x_{i-1}, x_j) + d_{T_1}(y, x_j) \leq 2 + i - 1 - j + 2 \leq d_G(x, y) + 3$, and equality $d_{T_1}(x, y) = d_G(x, y) + 3$ holds if and only if $d_G(x, y) = i - j$, vertex x is adjacent to x_i in T_1 (and thus in G, vertex x is not adjacent to x_{i-1}) and

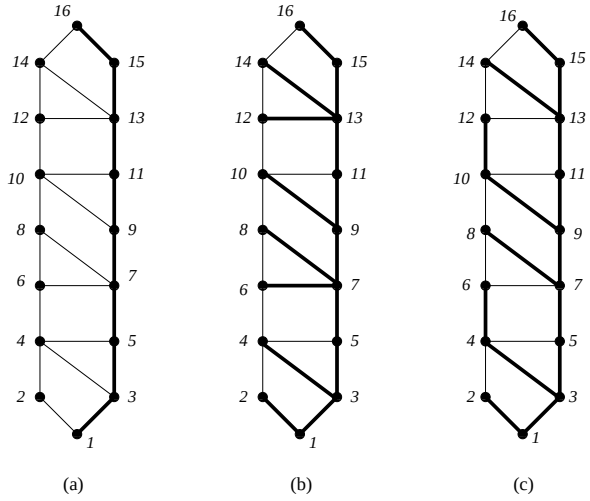

Figure 1: (a) An AT-free graph G with a dominating path P, (b) the caterpillar-tree T_1 of G and (c) the cactus-tree T_2 of G.

vertex y is adjacent to x_{j-1} in T_1 but does not coincide with x_j. We will show that in this case in the cactus-tree T_2, $d_{T_2}(x,y) \leq d_G(x,y)+2$.

Consider in G a shortest path $(y = y_0, y_1, \ldots, y_{i-j} = x)$ connecting vertices y and x. Clearly, $y_k \in L_{j+k}$ for each $k = 0, 1, \ldots, i-j-1$, and since y_k is a neighbor of y_{k+1} in layer L_{j+k}, by construction of T_2, we have $dev(y_0) = 1$ and $dev(y_{k+1}) \leq dev(y_k)+1 \leq k+2$. Hence, the deviation of vertex x is at most $i-j+1$. That is, there is a path in T_2 between x and a stem vertex x_s $(j-1 \leq s \leq i-2)$ of length $i-s$. The latter implies the existence in T_2 of a path of length $i-j+1$ between vertices x and x_{j-1}. Therefore, $d_{T_2}(x,y) \leq d_{T_2}(x,x_{j-1})+1 = i-j+1+1 = d_G(x,y)+2$. □

From this lemma we immediately conclude.

Theorem 1 *Any AT-free graph admits a system of two collective additive tree 2-spanners, constructable in linear time.*

In the next subsection, we will show that to get a collective additive stretch factor 2 for some AT-free graphs, one needs at least two spanning trees. Therefore, the result given in Theorem 1 is best possible. Furthermore, to achieve a collective additive stretch factor 1 or 0 for some AT-free graphs, we will show that one needs $\Omega(n)$ spanning trees.

2.2 Permutation graphs and trapezoid graphs

It is known [14] that any permutation graph admits a multiplicative tree 3-spanner. In this subsection, we show that any permutation graph admits an additive tree 2-spanner and any system of collective additive tree 1–spanners

must have $\Omega(n)$ spanning trees for some permutation graphs. Here also we disprove a conjecture given in [19], that any cocomparability graph admits an additive tree 2-spanner. We show that there exists even a trapezoid graph that does not admit any additive tree 2-spanner.

Let $G = (V, E)$ be a permutation graph given together with a permutation model. In what follows, "u.p." and "l.p." refer to a vertex's point on the upper and lower, respectively, line of the permutation model. Construct BFS-layers $(\{L_0, L_1, \cdots\})$ and the spine $\{x_1, x_2, \cdots\}$ of G as follows (the process continues until $L_i = \emptyset$).

set x_0 := the vertex whose u.p. is as far left as possible;
set $L_0 := \{x_0\}$;
set L_1 := {vertices whose l.p.s are to the left of the l.p. of x_0};
set x_1 := the vertex in L_1 with the u.p. as far right as possible;
set L_2 := {vertices whose u.p.s are between the u.p.s of x_0 and x_1} $\setminus L_1$;
set x_2 := the vertex in L_2 with the l.p. as far right as possible;
for $i = 3$ to n do
  if i is odd then
    set L_i := {vertices with l.p. between the l.p.s of x_{i-3} and x_{i-1}} $\setminus L_{i-1}$;
    set x_i := the vertex in L_i with the u.p. as far right as possible;
  else
    set L_i := {vertices with u.p. between the u.p.s of x_{i-3} and x_{i-1}} $\setminus L_{i-1}$;
    set x_i := the vertex in L_i with the l.p. as far right as possible;
enddo.

Lemma 3 *For all $i \geq 0$, if $y \in L_{i+1}$ then $x_i y \in E$.*

Proof: The lemma is clearly true for $i = 0$ or $i = 1$. Now assume $i \geq 2$ and i is even. (A similar proof holds for i odd.) By the construction of the L sets, it is clear that since $y \in L_{i+1}$, the l.p. of y is between the l.p.s of x_{i-2} and x_i and the u.p. of y is between the u.p.s of x_{i-1} and x_{i+1}. Furthermore, since the u.p. of x_i is to the left of the u.p. of x_{i-1} and thus to the left of the u.p. of y, we conclude that $x_i y \in E$. □

Note that as an immediate corollary of this lemma, the set $\{L_i : i = 1, 2, \cdots\}$ forms a BFS layering. This leads to the following theorem.

Theorem 2 *Every permutation graph admits an additive tree 2-spanner, constructable in linear time.*

Proof: Form the tree T by choosing all edges from x_i to L_{i+1}, for all appropriate i. Now consider any two vertices u and v, where $u \in L_i$ and $v \in L_j$, $i \leq j$. If $i = j$, then $d_G(u, v) = 1$ or 2 and $d_T(u, v) = 2$ (because of vertex x_{i-1} and Lemma 3). If $i < j$, then $d_G(u, v) \geq j - i$. If u is a spine vertex, then $d_T(u, v) = j - i$. Otherwise, by following the path $v, x_{j-1}, \cdots, x_{i-1}, u$, we see that $d_T(u, v) = j - i + 2$. Thus in all cases, $d_G(u, v) \leq d_T(u, v) + 2$, as required. □

We now show that there exists a trapezoid graph that does not admit any additive tree 2-spanner, thereby disproving a conjecture from [19] that any co-comparability graph admits an additive tree 2-spanner.

Consider the trapezoid graph G depicted in Figure 2, and assume that it has an additive tree 2-spanner T. We claim then, that all the cut edges (edges whereby the removal of their endpoints disconnects G) of G must belong to T. Indeed, if, for example, neither edge $(7,6)$ nor $(7,8)$ is an edge of T, then in T, vertices 7 and 6 must be connected by path $(7,5,4,6)$ and vertices 7 and 8 must be connected by path $(7,9,10,8)$. Now $d_T(8,6) = 6$ whereas $d_G(8,6) = 1$. If exactly one of $(7,6),(7,8)$ (without loss of generality $(7,6)$) is an edge of T, then at most three of the edges $(7,9),(9,10),(10,8),(8,6)$ may be in T. For at least one of the remaining edges, the distance in T between its endpoints is at least 4 contradicting T being an additive 2-spanner.

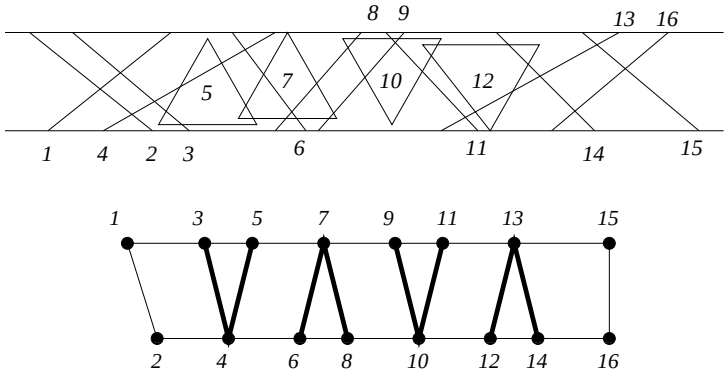

Figure 2: A trapezoid graph (with a trapezoid model) that does not admit an additive tree 2-spanner.

Thus, if G has an additive tree 2-spanner T, then all the cut edges of G must belong to T (see Figure 2). In T, paths $(9,10,11)$ and $(6,7,8)$ must be connected either by edge $(7,9)$ or by edge $(8,10)$. If $(7,9)$ is a tree edge, then we get $d_T(8,14) = 6 = d_G(8,14) + 3$ (independent of whether edge $(11,13)$ or edge $(10,12)$ is a tree edge). Otherwise, if $(8,10)$ is a tree edge, then we get $d_T(3,9) = 6 = d_G(3,9) + 3$. Contradictions with T being an additive tree 2-spanner of G prove the following result.

Observation 1 *There are trapezoid graphs that do not admit any additive tree 2-spanners.*

The next observation gives a lower bound on the number of spanning trees that guarantee a collective additive stretch factor 1 for bipartite permutation graphs (in fact, for all graph families containing complete bipartite graphs).

Observation 2 *There are bipartite permutation graphs on $2n$ vertices for which any system of collective additive tree 1–spanners needs to have at least $\Omega(n)$ spanning trees.*

Proof: Consider the complete bipartite graph $G = K_{n,n}$ on $2n$ vertices (which is clearly a permutation graph), and let $\mathcal{T}(G)$ be a system of μ collective additive tree 1-spanners of G. Then, for any two adjacent vertices x and y of G there must exist a spanning tree T in $\mathcal{T}(G)$ such that $d_T(x, y) \leq 2$. If $d_T(x, y) = 2$ then a common neighbor z of x and y in G would form a triangle with vertices x and y, which is impossible for $G = K_{n,n}$. Hence, $d_T(x, y) = 1$ must hold. Thus, every edge xy of G is an edge of some tree $T \in \mathcal{T}(G)$. Since there are n^2 graph edges to cover by spanning trees from $\mathcal{T}(G)$, we conclude $\mu \geq n^2/(2n-1) > n/2$. □

2.3 DSP-graphs

It follows from a result in [19] that any DSP-graph admits one additive tree 4-spanner. In this subsection we show that any DSP-graph admits a system of two collective additive tree 3-spanners and a system of five collective additive tree 2-spanners.

Let $G = (V, E)$ be a DSP-graph and let $P := (v = x_0, x_1, \ldots, x_l = u)$ be a dominating shortest path of G. We will build five spanning trees $\{T_1, T_2, T_3, T_4, T_5\}$ for G, all containing the edges of P, in such a way that for any two vertices $x, y \in V$, there will be a tree $T' \in \{T_1, T_2, T_3, T_4, T_5\}$ with $d_{T'}(x, y) \leq d_G(x, y) + 2$.

Our first three trees T_1, T_2, T_3 are very similar to the trees constructed for AT-free graphs. The tree $T_1 = (V, E_1)$ is constructed as follows. Add to initially empty set E_1 all edges of path P. Then, for each vertex $z \in V \setminus P$ choose an arbitrary neighbor w_z in P and add edge zw_z to E_1. The tree T_1 is an analog of the caterpillar-tree constructed for an AT-free graph. The second and third trees are analogs of the cactus-tree considered for an AT-free graph. The tree $T_2 = (V, E_2)$ is a special breadth-first-search-tree T_v with vertex v as the root, the tree $T_3 = (V, E_3)$ is a special breadth-first-search-tree T_u with vertex u as the root. We show here only how to construct the tree T_3. For construction of T_2 we can use the algorithm given in Subsection 2.1 with one additional line at the end: for each $z \in N_{l+1}(v)$, add edge zu to E_2 and set $dev(z) := 1$. T_3 is constructed similarly, we simply reverse the order of vertices of P and consider u instead of v and E_3 instead of E_2.

set $E_3 :=$ {all edges of the path $P := (v = x_0, x_1, \ldots, x_l = u)$};
set $dev(x_i) := 0$ for each vertex x_i of the path P;
for $i = 1$ to l do
  for each vertex $z \in N_i(u) \setminus \{x_{l-i}\}$ do
    among all neighbors of z in $N_{i-1}(u)$ choose a neighbor w with minimum deviation $dev(w)$;
    add edge zw to E_3 and set $dev(z) := dev(w) + 1$;
  enddo
enddo
for each $z \in N_{l+1}(u)$, add edge zv to E_3 and set $dev(z) := 1$.

Our tree $T_4 = (V, E_4)$ is a generalization of the tree T_2 and is constructed as follows.

set $E_4 := \{$all edges of the path $P := (v = x_0, x_1, \ldots, x_l = u)\}$;
set $dev(x_i) := 0$ for each vertex x_i of the path P;
for $i = 1$ to l do
  for each vertex $z \in N_i(v) \setminus \{x_i\}$ do case
    case (z is adjacent to x_{i-1} in G)
      add edge zx_{i-1} to E_4 and set $dev(z) := 1$;
    case (z is adjacent to x_i in G)
      add edge zx_i to E_4 and set $dev(z) := 1$;
    case (z is adjacent to a vertex $w \in N_i(v)$ that is adjacent to x_{i-1})
      choose such a w and add edge zw to E_4 and set $dev(z) := 2$;
    otherwise /* none of above */
      among all neighbors of z in $N_{i-1}(v)$ choose a neighbor w with
        minimum deviation $dev(w)$ (break ties arbitrarily);
      add edge zw to E_4 and set $dev(z) := dev(w) + 1$;
    endcase
  enddo
enddo
for each $z \in N_{l+1}(v)$, add edge zu to E_4 and set $dev(z) := 1$.

It is an easy exercise to show by induction that for any vertex $z \in N_i(v)$, the vertex of P closest to z in T_4 is either x_s or x_{s-1} with $s = i - dev(z) + 1$. Moreover, the length of the path of T_4 between z and P is $dev(z)$. Our last tree $T_5 = (V, E_5)$ is a version of the tree T_4, constructed downwards.

set $E_5 := \{$all edges of the path $P := (v = x_0, x_1, \ldots, x_l = u)\}$;
set $dev(x_i) := 0$ for each vertex x_i of the path P and $dev(z) := \infty$ for any
  $z \in V \setminus P$;
for $i = l - 1$ down to 1 do
  for each vertex $z \in N_i(v) \setminus \{x_i\}$ do case
    case (z is adjacent to x_{i+1} in G)
      add edge zx_{i+1} to E_5 and set $dev(z) := 1$;
    case (z is adjacent to x_i in G)
      add edge zx_i to E_5 and set $dev(z) := 1$;
    case (z is adjacent to a vertex $w \in N_i(v)$ that is adjacent to x_{i+1})
      choose such a w and add edge zw to E_5 and set $dev(z) := 2$;
    otherwise /* none of the above */
      if z has neighbors in $N_{i+1}(v)$ then
        among all neighbors of z in $N_{i+1}(v)$ choose a neighbor w with
          minimum deviation $dev(w)$ (break ties arbitrarily);
        if $dev(w) < \infty$ then add edge zw to E_5 and set $dev(z) := dev(w) + 1$;
    endcase
  enddo
enddo
for each vertex z with $dev(z)$ still ∞ do
  let $z \in N_i(v)$;
  if $i = l$ and z is adjacent to x_l then add edge zx_l to E_5 and set $dev(z) := 1$;

else add edge zx_{i-1} to E_5;
/* this edge exists in G since P is a dominating path
and z is adjacent in G neither to x_{i+1} nor x_i */
enddo.

Again, it is easy to see that for any vertex $z \in N_i(v)$ with finite deviation $dev(z)$, the vertex of P closest to z in T_5 is either x_s or x_{s+1} with $s = i + dev(z) - 1$. The length of the path of T_5 between z and P is $dev(z)$. In Figure 3 we show a DSP-graph G along with a shortest path P and spanning trees T_1, T_2, T_3, T_4, T_5.

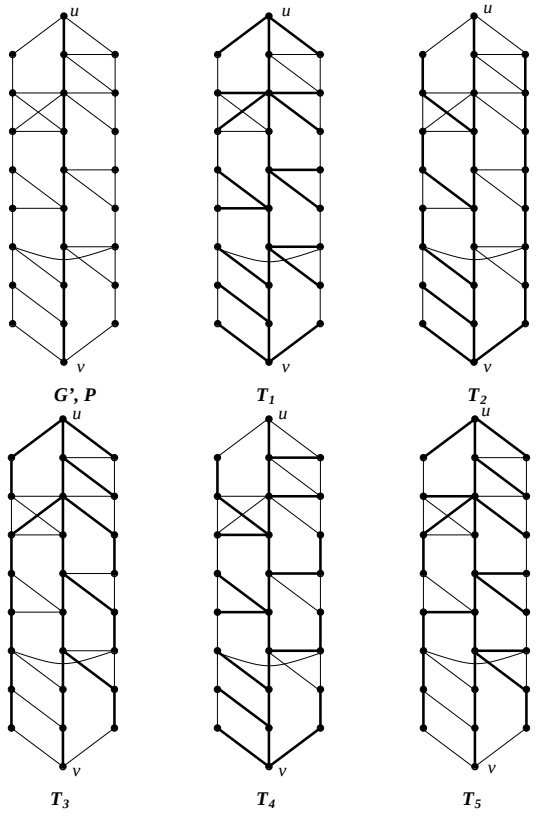

Figure 3: A DSP-graph G with a dominating path P and spanning trees T_1, T_2, T_3, T_4, T_5.

We are ready to present the main result of this subsection.

Lemma 4 *Let G be a DSP-graph with a dominating shortest path $P := (v = x_0, x_1, \ldots, x_l = u)$ and spanning trees T_1, T_2, T_3, T_4, T_5 constructed starting from P as described above. Then, for any two vertices $x, y \in V$ the following is true.*

1. $d_{T_1}(x,y) \leq d_G(x,y) + 4$;
2. *there is a tree* $T' \in \{T_1, T_2\}$ *such that* $d_{T'}(x,y) \leq d_G(x,y) + 3$;
3. *there is a tree* $T'' \in \{T_1, T_2, T_3, T_4, T_5\}$ *such that* $d_{T''}(x,y) \leq d_G(x,y) + 2$.

Proof: Consider two arbitrary vertices $x, y \in N[P]$ ($y \neq x$), and let $x \in N_i(v)$, $y \in N_j(v)$ and $i \geq j$. We know that $d_G(x,y) \geq i - j$. By the construction of tree T_1, $d_{T_1}(y, x_j) \leq 2$ and $d_{T_1}(x, x_i) \leq 2$. Hence, $d_{T_1}(x,y) \leq d_{T_1}(x,x_i) + d_{T_1}(x_i,x_j) + d_{T_1}(y,x_j) \leq 2 + i - j + 2 \leq d_G(x,y) + 4$, thereby proving claim 1 of the lemma.

Equality $d_{T_1}(x,y) = d_G(x,y) + 4$ holds if and only if $d_G(x,y) = i - j$, vertex x is adjacent to x_{i+1} in T_1, vertex y is adjacent to x_{j-1} in T_1 and $x \neq x_i, y \neq x_j$. As in the proof of Lemma 2, we now show that in this case (or more generally, when $d_G(x,y) = i - j$ and y is adjacent to x_{j-1} in G), $d_{T_2}(x,y) \leq d_G(x,y) + 2$.

Consider in G a shortest path $(y = y_0, y_1, \ldots, y_{i-j} = x)$ connecting vertices y and x. Clearly, $y_k \in N_{j+k}(v)$ for each $k = 0, 1, \ldots, i - j - 1$, and since y_k is a neighbor of y_{k+1} in layer $N_{j+k}(v)$, by construction of T_2, we have $dev(y_0) = 1$ and $dev(y_{k+1}) \leq dev(y_k) + 1 \leq k + 2$. Hence, the deviation of vertex x is at most $i - j + 1$. That is, there is a path in T_2 between x and a vertex $x_s \in P$ ($j - 1 \leq s \leq i - 1$) of length $i - s$. The latter implies the existence in T_2 of a path of length $i - j + 1$ between vertices x and x_{j-1}. Therefore, $d_{T_2}(x,y) \leq d_{T_2}(x, x_{j-1}) + 1 = i - j + 1 + 1 = d_G(x,y) + 2$.

Together with claim 1 of the lemma, this proves claim 2. We may assume in what follows that $d_G(x,y) > i - j$ or y is not adjacent to x_{j-1} in G (since otherwise, $d_{T_2}(x,y) \leq d_G(x,y) + 2$).

Case 1: y is not adjacent to x_{j-1} in G.

Then y is adjacent to x_j or x_{j+1} in T_1. Hence, $d_{T_1}(x,y) \leq d_{T_1}(x,x_i) + d_{T_1}(x_i,x_{j+1}) + d_{T_1}(y,x_{j+1}) \leq 2 + i - j - 1 + 2 \leq d_G(x,y) + 3$, and equality $d_{T_1}(x,y) = d_G(x,y) + 3$ holds if and only if $d_G(x,y) = i - j$, vertex x is adjacent to x_{i+1} in T_1, vertex y is adjacent to x_j in T_1 and $x \neq x_i$. Since $y \in N_j(v)$, $x \in N_i(v)$ and $u = x_l$ belongs to $N_l(v)$, we get $d_G(u,x) = l - i$ and $d_G(u,y) \geq l - j$. On the other hand, $d_G(y,u) \leq d_G(y,x) + 1 + d_G(x_{i+1},u) = i - j + 1 + l - (i+1) = l - j$. Hence, $x \in N_{l-i}(u), y \in N_{l-j}(u)$ and $x_{i+1} \in N_{l-i-1}(u)$. From this and since x is adjacent to x_{i+1} in G and there exists a shortest path of length $i - j$ in G between vertices x and y, it is easy to show that for the special breadth-first-search–tree T_3 rooted at u, $d_{T_3}(x,y) \leq d_{T_3}(y,x_{i+1}) + 1 = i + 1 - j + 1 = d_G(x,y) + 2$ (the proof is similar to the proof above for the tree T_2).

Case 2: $d_G(x,y) \geq i - j + 1$.

Again, for the tree T_1 we have $d_{T_1}(x,y) \leq d_{T_1}(x,x_i) + d_{T_1}(x_i,x_j) + d_{T_1}(y,x_j) \leq 2 + i - j + 2 \leq d_G(x,y) + 3$, and equality $d_{T_1}(x,y) = d_G(x,y) + 3$ holds if and only if $d_G(x,y) = i - j + 1$, vertex x is adjacent to x_{i+1} in T_1, vertex y is adjacent to x_{j-1} in T_1 and $x \neq x_i$, $y \neq x_j$. We show that in this case, $\min\{d_{T_4}(x,y), d_{T_5}(x,y)\} \leq d_G(x,y) + 2$.

Consider in G a shortest path $(y = y_0, y_1, \ldots, y_{i-j+1} = x)$ connecting vertices y and x. Clearly, there is only one index h $(0 \leq h \leq i-j)$ such that both vertices y_h and y_{h+1} lie in $N_{j+h}(v)$. For index k $(0 \leq k \leq h-1)$, $y_k \in N_{j+k}(v)$, $y_{k+1} \in N_{j+k+1}(v)$.

If y_{h+1} is adjacent to x_{j+h-1} or to x_{j+h} then, by construction of tree T_4, we have $dev(y_{h+1}) = 1$ and $dev(y_{t+1}) \leq dev(y_t) + 1 \leq t - (h+1) + 2$ for any t, $(h+1 \leq t \leq i-j)$. Hence, the deviation of vertex x is at most $i - (j + h - 1)$, and there is a path in T_4 of length $dev(x)$ from vertex x to either x_s or x_{s-1}, where $s = i - dev(x) + 1 \geq i - i + (j + h - 1) + 1 = j + h$. In any case $d_{T_4}(x, x_{s-1}) \leq dev(x) + 1$ and therefore $d_{T_4}(x, x_{j-1}) \leq d_{T_4}(x, x_{s-1}) + d_{T_4}(x_{s-1}, x_{j-1}) \leq dev(x) + 1 + s - j = dev(x) + 1 + i - dev(x) + 1 - j = i - j + 2$. The latter implies $d_{T_4}(x, y) \leq d_{T_4}(x, x_{j-1}) + 1 \leq i - j + 2 + 1 = d_G(x, y) + 2$.

So, we may assume that y_{h+1} is adjacent neither to x_{j+h-1} nor to x_{j+h}, i.e., y_{h+1} is adjacent to x_{j+h+1}. Assume that y_h is adjacent to x_{j+h-1} and $h \neq 0$. Then, by construction of tree T_4, $dev(y_{h+1}) = 2$ and $dev(y_{t+1}) \leq dev(y_t) + 1 \leq t - (h+1) + 3$ for any t, $(h+1 \leq t \leq i-j)$. Hence, the deviation of vertex x is at most $i - (j + h - 1) + 1$, and there is a path in T_4 of length $dev(x)$ from vertex x to either x_s or x_{s-1}, where $s = i - dev(x) + 1 \geq i - i + (j + h - 1) - 1 + 1 = j + h - 1$. Recall that $h \neq 0$ and hence $s - 1 \geq j - 1$. Again, in any case $d_{T_4}(x, x_{s-1}) \leq dev(x) + 1$ and therefore $d_{T_4}(x, x_{j-1}) \leq d_{T_4}(x, x_{s-1}) + d_{T_4}(x_{s-1}, x_{j-1}) \leq dev(x) + 1 + s - j = dev(x) + 1 + i - dev(x) + 1 - j = i - j + 2$. The latter implies $d_{T_4}(x, y) \leq d_{T_4}(x, x_{j-1}) + 1 \leq i - j + 2 + 1 = d_G(x, y) + 2$.

If now y_h is adjacent to x_{j+h+1} or to x_{j+h} then, by construction of tree T_5, $dev(y_h) = 1$ and $dev(y_{t-1}) \leq dev(y_t) + 1 \leq h - t + 2$ for any t, $(1 \leq t \leq h)$. Hence, the deviation of vertex y is at most $h+1$, and there is a path in T_5 of length $dev(y)$ from vertex y to either x_s or x_{s+1}, where $s = j + dev(y) - 1 \leq j + h + 1 - 1 = j + h$. In any case $d_{T_5}(y, x_{s+1}) \leq dev(y) + 1$ and therefore $d_{T_5}(y, x_{i+1}) \leq d_{T_5}(y, x_{s+1}) + d_{T_5}(x_{s+1}, x_{i+1}) \leq dev(y) + 1 + i - s = dev(y) + 1 + i - j - dev(y) + 1 = i - j + 2$. The latter implies $d_{T_5}(x, y) \leq d_{T_5}(y, x_{i+1}) + 1 \leq i - j + 2 + 1 = d_G(x, y) + 2$. Thus, we may assume that y_h is adjacent to neither x_{j+h+1} nor x_{j+h}.

It remains then to consider only the last case when $h = 0$, y_1 is adjacent to x_{j+1} but not to x_j,x_{j-1}, and $y = y_0$ is adjacent to x_{j-1} but not to x_j,x_{j+1}. Recall that $y_0, y_1 \in N_j(v)$. By construction, in T_5, y is adjacent to y_1, y_1 is adjacent to x_{j+1}, and x is adjacent to x_{i+1}. Therefore, $d_{T_5}(x, y) \leq 1 + d_{T_5}(x_{i+1}, x_{j+1}) + 2 = i - j + 3 = d_G(x, y) + 2$, completing the proof of the lemma. □

Theorem 3 *Any DSP-graph admits one additive tree 4-spanner, a system of two collective additive tree 3-spanners and a system of five collective additive tree 2-spanners. Moreover, given a dominating shortest path of G, all trees are constructable in linear time.*

Note that an induced cycle on six vertices gives a DSP-graph that does not admit an additive tree 3-spanner. Therefore, two trees are necessary to get a collective additive stretch factor 3. However, it is an open question whether to achieve a collective additive stretch factor 2, one really needs five spanning trees.

2.4 Graphs with bounded asteroidal number

It is known [11] that any graph G with asteroidal number $\mathsf{an}(G)$ admits an additive tree $(3\mathsf{an}(G)-1)$-spanner. In this subsection we show that any graph with asteroidal number $\mathsf{an}(G)$ admits a system of $\mathsf{an}(G)(\mathsf{an}(G)-1)/2$ collective additive tree 4-spanners and a system of $\mathsf{an}(G)(\mathsf{an}(G)-1)$ collective additive tree 3-spanners.

In what follows we will use the following two definitions and two theorems from [11]. An asteroidal set A of a graph G is *repulsive* if for every vertex $v \in V \setminus N[A]$, not all vertices of A are contained in one connected component of $G - N[v]$.

Theorem 4 [11] *Any graph has a repulsive asteroidal set.*

A set $D \subseteq V$ in a graph G is said to be a *dominating target*, if $D \cup S$ is a dominating set in G for every set $S \subseteq V$ for which the subgraph of G induced by $D \cup S$ is connected.

Theorem 5 [11] *Any graph G has a dominating target D with $|D| \leq \mathsf{an}(G)$. Furthermore, every repulsive asteroidal set of G is such a dominating target of G.*

We will need a stronger version of the above theorem. Let $G = (V, E)$ be a graph and $D \subseteq V$ be a repulsive asteroidal set of G.

Lemma 5 *For every $x, y \in V$ and dominating target D, there exist $a, b \in D$ such that $x, y \in N[P]$ for any path P of G between a and b.*

Proof: The claim is evident if $x, y \in N[D]$. Therefore assume, without loss of generality, that $x \in V \setminus N[D]$. Then, by the definition of a repulsive asteroidal set, there must exist vertices a, b in D that lie in different connected components of $G - N[x]$. Let $CC_x(a)$ and $CC_x(b)$ be those connected components. Clearly, any path P between a and b intersects $N[x]$ and therefore $x \in N[P]$. We may assume that $y \notin N[\{a, b\}]$, since otherwise the lemma follows. Let, however, $y \in N[D]$ and c be a vertex from $N[y] \cap D$. Since c cannot be adjacent with x (recall that $x \in V \setminus N[D]$), c must lie in some connected component $CC_x(c)$ of $G - N[x]$. Clearly, $CC_x(c)$ cannot coincide with both $CC_x(a)$ and $CC_x(b)$. Assuming $CC_x(c) \neq CC_x(a)$, we conclude that any path connecting a and c dominates both x and y.

Now let $y \in V \setminus N[D]$. Then, there must exist vertices c, d in D that lie in different connected components of $G - N[y]$. Hence, for any path P between c and d, $y \in N[P]$. If c and d lie in different connected components of $G - N[x]$ as well, then the lemma follows, since any path connecting c and d must dominate both x and y. Therefore assume, without loss of generality, that neither c nor d belongs to $CC_x(a)$. Now any path $P(a, c)$ between a and c and any path $P(a, d)$ between a and d must intersect $N[x]$. Since the union of $P(a, c)$ and $P(a, d)$ connects vertices c and d, $N[y]$ must intersect every path between a and c or every path between a and d. Let a and c be vertices lying in different

connected components of $G-N[y]$. Then, since a and c lie in different connected components of $G-N[x]$ too, we are done; any path between a and c dominates x and y. □

Consider two arbitrary vertices a, b of D and a shortest path $P(a,b) := (a = x_0, x_1, \ldots, x_l = b)$ connecting a and b in G. We can build two spanning trees $T_1(a,b)$ and $T_2(a,b)$ for G, both containing the edges of $P(a,b)$, in such a way that for any two vertices $x, y \in N[P(a,b)]$, $d_{T_1(a,b)}(x,y) \leq d_G(x,y)+4$ and $\min\{d_{T_1(a,b)}(x,y), d_{T_2(a,b)}(x,y)\} \leq d_G(x,y)+3$.

Our trees $T_1(a,b)$ and $T_2(a,b)$ are very similar to the trees constructed for AT-free graphs. The tree $T_1(a,b) = (V, E_1)$ is constructed as follows. Add to initially empty set E_1 all edges of path $P(a,b)$. Then, for each vertex $z \in N[P(a,b)]$ choose an arbitrary neighbor w in $P(a,b)$ and add edge zw to E_1. The obtained subtree of G (which covers so far only vertices from $N[P(a,b)]$) extends to a spanning tree $T_1(a,b)$ arbitrarily. The tree $T_1(a,b)$ is an analog of the caterpillar-tree constructed for an AT-free graph. The second tree is an analog of the cactus-tree considered for an AT-free graph. The tree $T_2(a,b) = (V, E_2)$ is a special breadth-first-search-tree T_a with vertex a as the root and is constructed as follows.

set $E_2 :=$ {all edges of the path $P := (a = x_0, x_1, \ldots, x_l = b)$};
set $dev(x_i) := 0$ for each vertex x_i of the path $P(a,b)$;
for $i = 1$ to $ecc(a)$ do
  if $i \leq l$ then set $A := N_i(a) \setminus \{x_i\}$ else set $A := N_i(a)$;
  for each vertex $z \in A$ do
    among all neighbors of z in $N_{i-1}(a)$ choose a neighbor w with minimum deviation $dev(w)$;
    add edge zw to E_2 and set $dev(z) := dev(w) + 1$;
  enddo
enddo.

Lemma 6 *For any two vertices $x, y \in N[P(a,b)]$ the following is true.*

1. $d_{T_1(a,b)}(x,y) \leq d_G(x,y)+4$;
2. *there is a tree $T' \in \{T_1(a,b), T_2(a,b)\}$ such that $d_{T'}(x,y) \leq d_G(x,y)+3$.*

Proof: The proof is similar to the proof of claims 1 and 2 of Lemma 4 and therefore is omitted. □

If we construct trees $T_1(a,b)$ and $T_2(a,b)$ for each pair of vertices $a, b \in D$, from Lemma 5 and Lemma 6, we get (recall that $|D| \leq \mathsf{an}(G)$):

Theorem 6 *Any graph G with asteroidal number $\mathsf{an}(G)$ admits a system of $\mathsf{an}(G)(\mathsf{an}(G)-1)/2$ collective additive tree 4-spanners and a system of $\mathsf{an}(G)$ $(\mathsf{an}(G)-1)$ collective additive tree 3-spanners.*

Corollary 1 *Any graph G with asteroidal number bounded by a constant admits a system of a constant number of collective additive tree 3-spanners. Moreover, given a repulsive asteroidal set of G, all trees are constructable in total linear time.*

3 Routing labeling schemes in AT-free related graphs

In this section, we use the results obtained above to design compact and efficient routing labeling schemes for graphs from the AT-free hierarchy. We first describe a direct routing labeling scheme for all graphs admitting a system of μ collective additive tree r-spanners. This scheme uses the efficient routing labeling scheme developed in [7, 20] for arbitrary trees. Then we show that, using the special structure of the trees constructed in Section 2, better routing labeling schemes can be designed for AT-free and related families of graphs. Among other results we show that any AT-free graph with diameter D and maximum vertex degree Δ admits a $(3\log_2 D + 6\log_2 \Delta + 3)$-bit routing labeling scheme of deviation at most 2. Moreover, the scheme is computable in linear time, and the routing decision is made in constant time per vertex.

It is worth mentioning that any AT-free graph admits a $(\log_2 D + 1)$-bit distance labeling scheme of deviation at most 2 (see [8]). That is, there is a function L labeling the vertices of each AT-free graph G with (not necessarily distinct) labels of up to $\log_2 D + 1$ bits such that given two labels $L(v), L(u)$ of two vertices v, u of G, it is possible to compute in constant time, by merely inspecting the labels of u and v, a value $\widehat{d}(u,v)$ such that $0 \leq \widehat{d}(u,v) - d_G(u,v) \leq 2$. To the best of our knowledge, the method of [8] cannot be used (at least directly) to design a routing labeling scheme for AT-free graphs.

3.1 A direct routing labeling scheme for all graphs admitting a system of μ collective additive tree r-spanners

We will need the following two results on distance and routing labeling schemes for arbitrary trees.

Theorem 7 [16] *There is a function DL labeling in $O(n \log n)$ total time the vertices of an n-vertex tree T with labels of up to $O(\log^2 n)$ bits such that given two labels $DL(v), DL(u)$ of two vertices v, u of T, it is possible to compute in constant time the distance $d_T(v,u)$, by merely inspecting the labels of u and v.*

Theorem 8 [7, 20] *There is a function RL labeling in $O(n)$ total time the vertices of an n-vertex tree T with labels of up to $O(\log^2 n / \log\log n)$ bits such that given two labels $RL(v), RL(u)$ of two vertices v, u of T, it is possible to determine in constant time the port number, at u, of the first edge on the path in T from u to v, by merely inspecting the labels of u and v.*

Now consider a graph G admitting a system $\mathcal{T}(G) = \{T_1, T_2, \ldots, T_\mu\}$ of μ collective additive tree r-spanners. We can preprocess each tree T_i using the algorithms from [16] and [7] and assign to each vertex v of G a distance label $DL_i(v)$ of size $O(\log^2 n)$ bits and a routing label $RL_i(v)$ of size $O(\log^2 n / \log\log n)$ bits associated with the tree T_i. Then we can form a label $L(v)$ of v of size $O(\mu \log^2 n)$ bits by concatenating the μ tree-labels:

$$L(v) := DL_1(v) \circ \cdots \circ DL_\mu(v) \circ RL_1(v) \circ \cdots \circ RL_\mu(v).$$

Assume now that a vertex v wants to send a message to a vertex u. Given the labels $L(v)$ and $L(u)$, v first uses the substrings $DL_1(v) \circ \cdots \circ DL_\mu(v)$ and $DL_1(u) \circ \cdots \circ DL_\mu(u)$ to compute in $O(\mu)$ time the distances $d_{T_i}(v,u)$ $(i = 1, \ldots, \mu)$ and an index k such that $d_{T_k}(v,u) = \min\{d_{T'}(v,u) : T' \in \mathcal{T}(G)\}$. Then, v extracts from $L(u)$ the substring $RL_k(u)$ and forms a header of the message $H(u) := k \circ RL_k(u)$. Now, the initiated message with the header $H(u) = k \circ RL_k(u)$ is routed to the destination using the tree T_k: when the message arrives at an intermediate vertex x, vertex x using its own substring $RL_k(x)$ and the string $RL_k(u)$ from the header makes a constant time routing decision.

Thus, the following result is true.

Theorem 9 *Let G be an n-vertex graph admitting a system of μ collective additive tree r-spanners. Then G has a $O(\mu \log^2 n)$-bit routing labeling scheme of deviation r. Once computed by the sender in $O(\mu)$ time, headers never change, and the routing decision is made in constant time per vertex.*

3.2 A better routing labeling scheme for AT-free graphs

3.2.1 Labels

In subsection 2.1, we showed that any AT-free graph $G = (V, E)$ admits a system of two collective additive tree 2-spanners. During the construction of the cactus-tree T_2 for G, each vertex $z \in V$ received a deviation number $dev(z)$ that is the distance in T_2 between z and the stem $P := (x_0, x_1, \ldots, x_l)$ of T_2 using only "down" edges. To simplify the routing decision, it will be useful to construct one more spanning tree $T' = (V, E')$ for G. Let $P := (x_0, x_1, \ldots, x_l)$ be the dominating path of G described in Lemma 1.

set $E' := \{$all edges of the path $P := (x_0, x_1, \ldots, x_l)\}$;
set $dev'(x_i) := 0$ for each vertex x_i of the path P and $dev'(z) := l + 1$ for any $z \in V \setminus P$;
for each vertex $z \in N_l(x_0)$ that is adjacent to x_l, set $dev'(z) := 1$ and add edge zx_l to E';
for $i = l - 1$ down to 1 do
  for each vertex $z \in N_i(x_0) \setminus \{x_i\}$ do
    if z is adjacent to x_i in G then add edge zx_i to E' and set $dev'(z) := 1$;
    else if z has neighbors in $N_{i+1}(x_0)$ then
      among all neighbors of z in $N_{i+1}(x_0)$, choose a neighbor w with minimum deviation $dev'(w)$ (break ties arbitrarily);
      if $dev'(w) < l + 1$ then add edge zw to E' and set $dev'(z) := dev'(w) + 1$;
  enddo
enddo
for each vertex z with $dev'(z)$ still $l + 1$ do
  let $z \in N_i(x_0)$;
  add edge zx_{i-1} to E';

enddo.

We name tree T' the *willow-tree* of G. As a result of its construction, each vertex $z \in V$ got a second deviation number $dev'(z)$, which is either $l+1$ or the distance in T' (using only "horizontal" or "up" edges) between z and the path $P := (x_0, x_1, \ldots, x_l)$ of T'.

Now we are ready to describe the routing labels of the vertices of G. For each vertex $x_i \in P$ $(i = 0, 1, \ldots, l)$, we have

$$Label(x_i) := (b(x_i), level(x_i), port_{up}(x_i), port_{down}(x_i)),$$

where

- $b(x_i) := 1$, a bit indicating that x_i belongs to P;
- $level(x_i)$ $(= i)$ is the index of x_i in P, i.e., the distance $d_G(x_i, x_0)$;
- $port_{up}(x_i)$ is the port number at vertex x_i of the edge $x_i x_{i+1}$ (if $i = l$, $port_{up}(x_i) := \mathsf{nil}$);
- $port_{down}(x_i)$ is the port number at vertex x_i of the edge $x_i x_{i-1}$ (if $i = 0$, $port_{down}(x_i) := \mathsf{nil}$).

For each vertex $z \in V \setminus P$, we have

$$Label(z) := (b(z), level(z), a_{v\downarrow}(z), port_{v-in}(z), port_{v-out}(z), a_h(z), port_{h-in}(z), port_{h-out}(z), dev(z), port_{down}(z), dev'(z), port_{up}(z)),$$

where

- $b(z) := 0$, a bit indicating that z does not belong to P;
- $level(z)$ is the distance $d_G(z, x_0)$;
- $a_{v\downarrow}(z)$ is a bit indicating whether z is adjacent to $x_{level(z)-1}$;
- $port_{v-in}(z)$ is the port number at vertex $x_{level(z)-1}$ of the edge $x_{level(z)-1}z$ (if z and $x_{level(z)-1}$ are not adjacent in G, then $port_{v-in}(z) := \mathsf{nil}$);
- $port_{v-out}(z)$ is the port number at vertex z of the edge $z x_{level(z)-1}$ (if z and $x_{level(z)-1}$ are not adjacent in G, then $port_{v-out}(z) := \mathsf{nil}$);
- $a_h(z)$ is a bit indicating whether z is adjacent to $x_{level(z)}$;
- $port_{h-in}(z)$ is the port number at vertex $x_{level(z)}$ of the edge $x_{level(z)}z$ (if z and $x_{level(z)}$ are not adjacent in G, then $port_{h-in}(z) := \mathsf{nil}$);
- $port_{h-out}(z)$ is the port number at vertex z of the edge $z x_{level(z)}$ (if z and $x_{level(z)}$ are not adjacent in G, then $port_{h-out}(z) := \mathsf{nil}$);

- $dev(z)$ is the deviation of z in tree T_2;
- $port_{down}(z)$ is the port number at vertex z of the edge zw, where w is the father of z in T_2;
- $dev'(z)$ is the deviation of z in tree T';
- $port_{up}(z)$ is the port number at vertex z of the edge zw, where w is the father of z in T' (if $dev'(z) = l + 1$, $port_{up}(z) :=$ nil).

Clearly, the label size of each vertex of G is at most $3\log_2 l + 6\log_2 \Delta + 3 \leq 3\log_2 D + 6\log_2 \Delta + 3$ bits.

3.2.2 Routing decision

The routing decision algorithm is obvious. Suppose that a packet with the header (address of destination) $Label(y)$ arrives at vertex x. Vertex x can use the following constant time algorithm to decide where to submit the packet. Note that each vertex v of G is uniquely identified by its label $Label(v)$.

function routing_decision_AT-free($Label(x), Label(y)$)

```
if Label(x) = Label(y) then return "packet reached its destination";
else do case
  case (b(x) = 1) /* x belongs to P and routing is performed on the
                     caterpillar-tree T_1 of G */
    do case
      case (level(x) > level(y))
        send packet via port_down(x);
      case (level(x) < level(y))
        if b(y) = 1 then send packet via port_up(x);
        else if level(y) = level(x) + 1 and a_{v↓}(y) = 1 then send packet via
             port_{v-in}(y);
          else send packet via port_up(x);
      case (level(x) = level(y))
        if a_h(y) = 1 then send packet via port_{h-in}(y);
        else send packet via port_down(x);
    endcase;
  /* now x does not belong to P */
  case (level(x) > level(y))
    do case
      case (a_{v↓}(x) = 1)
        send packet via port_{v-out}(x); /* routing is performed on T_1 */
      case (b(y) = 1 or b(y) = 0 and a_h(y) = 1)
        send packet via port_{h-out}(x); /* routing is performed on T_1 */
      otherwise /* here we have d_{T_1}(x, y) = level(x) − level(y) + 3 */
        if dev(x) ≤ level(x) − level(y) + 1 then send packet via port_down(x);
             /* the cactus-tree T_2 of G is used for routing */
```

```
        else send packet via port_{h-out}(x); /* routing is performed on T_1 */
    endcase;
  case (level(x) < level(y))
    do case
      case (a_h(x) = 1)
        send packet via port_{h-out}(x); /* routing is performed on T_1 */
      case (b(y) = 1 or b(y) = 0 and a_{v↓}(y) = 1)
        send packet via port_{v-out}(x); /* routing is performed on T_1 */
      otherwise /* here we have d_{T_1}(x, y) = level(y) − level(x) + 3 */
        if dev'(x) ≤ level(y) − level(x) + 1 then send packet via port_{up}(x);
            /* the willow-tree T' of G is used for routing */
        else send packet via port_{v-out}(x); /* routing is performed on T_1 */
    endcase;
  case (level(x) = level(y)) /* routing is performed on T_1 */
    if a_h(x) = 1 then send packet via port_{h-out}(x);
    else send packet via port_{v-out}(x);
  endcase.
```

Thus, we have the following result.

Theorem 10 *Every AT-free graph of diameter $D := diam(G)$ and of maximum vertex degree Δ admits a $(3\log_2 D + 6\log_2 \Delta + 3)$-bit routing labeling scheme of deviation at most 2 (and also a $(\log_2 D + 2\log_2 \Delta + 2)$-bit routing labeling scheme of deviation at most 3). Moreover, the scheme is computable in linear time, and the routing decision is made in constant time per vertex.*

A $(\log_2 D + 2\log_2 \Delta + 2)$-bit routing labeling scheme of deviation at most 3 can easily be derived from the structure of tree T_1.

3.3 Routing in AT-free related graphs

3.3.1 Permutation graphs

The above idea can be used to design a compact routing labeling scheme for permutation graphs. Note that, by Theorem 2, any permutation graph admits an additive tree 2-spanner. According to the tree construction algorithm, the tree has a spine (shortest path) $(x_0, x_1, \cdots, x_l)$. For every vertex $y \in N_i(x_0)$, $0 < i \leq l + 1$, we have $yx_{i-1} \in E(G) \cap E(T)$. Each vertex x_i on the spine needs only to memorize the port numbers to x_{i+1} and to x_{i-1}. Any other vertex y needs only to memorize the port numbers from x_{i-1} to y and from y to x_{i-1}. Moreover, each vertex w will memorize also one extra bit, indicating whether w is on the spine, and the distance $d_G(w, x_0)$ which is the index of the level $L_i = N_i(x_0)$ it belongs to. Other details are left to the reader. We only formulate the final result for the permutation graphs.

Theorem 11 *Every permutation graph of diameter D and maximum vertex degree Δ admits a $(\log_2 D + 2\log_2 \Delta + 1)$-bit routing labeling scheme of deviation*

at most 2. Moreover, the scheme is computable in linear time, and the routing decision is made in constant time per vertex.

3.3.2 DSP-graphs

Based on Lemma 4, the following result can be proved for DSP-graphs.

Theorem 12 *Every n-vertex DSP-graph of diameter D and of maximum vertex degree Δ admits the following routing labeling schemes.*

Scheme	*Label Size*	*Decision time*	*Deviation*
1	$(\log_2 D + 2\log_2 \Delta + 3)$	$O(1)$	*4*
2	$(3\log_2 D + 8\log_2 \Delta + 4)$	$O(1)$	*3*
3	$O(\log^2 n/\log\log n)$	$O(1)$	*2*

The first routing labeling scheme can easily be constructed using the special structure of additive tree 4-spanner T_1 (see Section 2.3). The second routing labeling scheme can be constructed using the special structure of the system of two collective additive tree 3-spanners $\{T_1, T_2\}$. The construction is very similar to that used for AT-free graphs (see the scheme of deviation 2 in Section 3.2), and its details are left to the reader (consider T_3 instead of T'; since in DSP-graphs one needs to handle also the case when vertex z from $N_i(x_0) \setminus P$ is adjacent to x_{i+1}, $i \in \{1, \ldots, l-1\}$, where we have a $(2\log_2 \Delta + 1)$-bit surplus in the label size). Here we will give a brief description only of the third routing labeling scheme (of deviation 2).

In Section 2.3, we have shown how to construct a system of five collective additive tree 2-spanners for a DSP-graph G with a dominating shortest path $P := (x_0, x_1, \ldots, x_l)$. From Theorem 9, one can conclude that G has a $O(\log^2 n)$-bit routing labeling scheme with deviation 2 and constant routing decision. In what follows, we will show that, to find distances in those trees, shorter labels are sufficient.

We preprocess each tree T_i $(i = 1, 2, 3, 4, 5)$ using the algorithm from [7] and assign to each vertex v of G a routing label $RL_i(v)$ of size $O(\log^2 n/\log\log n)$ bits associated with the tree T_i. Then we form a label $L(v)$ of v of size $6\log_2 D + O(\log^2 n/\log\log n) = O(\log^2 n/\log\log n)$ bits as follows:

$$L(v) := \\ level(v) \circ level'(v) \circ dev_2(v) \circ dev_3(v) \circ dev_4(v) \circ dev_5(v) \circ RL_1(v) \circ \cdots \circ RL_5(v),$$

where $level(v) := d_G(x_0, v)$, $level'(v) := d_G(x_l, v)$ and $dev_2(v), dev_3(v), dev_4(v)$, $dev_5(v)$ are the deviation numbers of v in T_2, T_3, T_4 and T_5, respectively.

Assume now that a vertex x wants to send a message to a vertex y. Given the labels $L(x)$ and $L(y)$, x first uses the substrings $(level(x) \circ level'(x) \circ dev_2(x) \circ dev_3(x) \circ dev_4(x) \circ dev_5(x))$ and $(level(y) \circ level'(y) \circ dev_2(y) \circ dev_3(y) \circ dev_4(y) \circ dev_5(y))$ and the function create_header_DSP (described below) to form in $O(1)$ time the header $H(y) := k \circ RL_k(y)$ of the message by finding an index

$k \in \{1,2,3,4,5\}$ such that $d_{T_k}(x,y) = \min\{d_{T'}(x,y) : T' \in \{T_1,\ldots,T_5\}\}$ and extracting from $L(y)$ the substring $RL_k(y)$. Then, the initiated message with the header $H(y) = k \circ RL_k(y)$ can be routed to the destination using the tree T_k.

function create_header_DSP($L(x), L(y)$)

if $level(x) = level(y)$ then
  do case
    case ($dev_2(x) = 1$ and $dev_2(y) = 1$), set $k := 2$;
    case ($dev_3(x) = 1$ and $dev_3(y) = 1$), set $k := 3$;
    case ($dev_4(x) \leq 2$ and $dev_2(y) = 1$), set $k := 4$;
    case ($dev_5(x) \leq 2$ and $dev_3(y) = 1$), set $k := 5$;
    otherwise, set $k := 1$;
  endcase
else
  if $level(x) > level(y)$ then set $v := x$ and $u := y$;
  else set $v := y$ and $u := x$;
  do case
    case ($dev_2(u) = 1$ and $dev_2(v) \leq level(v) - level(u) + 1$), set $k := 2$;
    case ($dev_3(v) = 1$ and $dev_3(u) \leq level'(u) - level'(v) + 1$), set $k := 3$;
    case ($dev_4(u) = 1$ and $dev_4(v) \leq level(v) - level(u) + 1$), set $k := 4$;
    case ($dev_5(v) = 1$ and $dev_5(u) \leq level(v) - level(u) + 1$), set $k := 5$;
    otherwise, set $k := 1$;
  endcase
set the header of the message to be $(k, RL_k(y))$.

The correctness of this procedure follows from the proof of Lemma 4.

3.3.3 Graphs with bounded asteroidal number

Let $G = (V,E)$ be a graph and let $D \subseteq V$ be a repulsive asteroidal set of G. We have $|D| \leq \mathsf{an}(G)$. For each pair of vertices a_i, b_i ($i = 1,\ldots,\mu, \mu := |D|(|D|-1)/2$) from D we construct three trees $T_1^i := T_1(a_i,b_i)$, $T_2^i := T_2(a_i,b_i)$ and $T_3^i := T_2(b_i,a_i)$ starting from a shortest path $P(a_i,b_i)$ as described in Section 2.4. With each vertex v of G we associate a characteristic vector $\chi(v) = (\chi_1(v), \chi_2(v), \ldots, \chi_\mu(v))$, where $\chi_i(v) = 1$ if v belongs to $N[P(a_i,b_i)]$ and 0 otherwise.

For each index i ($i = 1,\ldots,\mu$), a vertex z of G will store in its label the following routing information. If z belongs to $P_i := P(a_i,b_i)$ then

$$L_i(z) := (b(z), level(z), port_{up}(z), port_{down}(z)),$$

where

- $b(z) := 1$, a bit indicating that z belongs to P_i;
- $level(z) := d_G(z,a_i)$;

- $port_{up}(z)$ is the port number at vertex z of the edge of P_i leading towards b_i (if $z = b_i$, $port_{up}(z) := \mathsf{nil}$);
- $port_{down}(z)$ is the port number at vertex z of the edge of P_i leading towards a_i (if $z = a_i$, $port_{down}(z) := \mathsf{nil}$).

If z does not belong to $P_i := P(a_i, b_i) := (a_i = x_0, x_1, \ldots, x_l = b_i)$ then

$L_i(z) := (b(z), level(z), a_{v\downarrow}(z), port_{v\downarrow-in}(z), port_{v\downarrow-out}(z), a_h(z), port_{h-in}(z), port_{h-out}(z), a_{v\uparrow}(z), port_{v\uparrow-in}(z), port_{v\uparrow-out}(z), dev(z), port_{down}(z), dev'(z), port_{up}(z))$,

where

- $b(z) := 0$, a bit indicating that z does not belong to P_i;
- $level(z)$ is the distance $d_G(z, a_i)$;
- $a_{v\downarrow}(z)$ is a bit indicating whether z is adjacent to $x_{level(z)-1}$;
- $port_{v\downarrow-in}(z)$ is the port number at vertex $x_{level(z)-1}$ of the edge $x_{level(z)-1}z$ (if z and $x_{level(z)-1}$ are not adjacent in G, then $port_{v\downarrow-in}(z) := \mathsf{nil}$);
- $port_{v\downarrow-out}(z)$ is the port number at vertex z of the edge $zx_{level(z)-1}$ (if z and $x_{level(z)-1}$ are not adjacent in G, then $port_{v\downarrow-out}(z) := \mathsf{nil}$);
- $a_h(z)$ is a bit indicating whether z is adjacent to $x_{level(z)}$;
- $port_{h-in}(z)$ is the port number at vertex $x_{level(z)}$ of the edge $x_{level(z)}z$ (if z and $x_{level(z)}$ are not adjacent in G, then $port_{h-in}(z) := \mathsf{nil}$);
- $port_{h-out}(z)$ is the port number at vertex z of the edge $zx_{level(z)}$ (if z and $x_{level(z)}$ are not adjacent in G, then $port_{h-out}(z) := \mathsf{nil}$);
- $a_{v\uparrow}(z)$ is a bit indicating whether z is adjacent to $x_{level(z)+1}$;
- $port_{v\uparrow-in}(z)$ is the port number at vertex $x_{level(z)+1}$ of the edge $x_{level(z)+1}z$ (if z and $x_{level(z)+1}$ are not adjacent in G, then $port_{v\uparrow-in}(z) := \mathsf{nil}$);
- $port_{v\uparrow-out}(z)$ is the port number at vertex z of the edge $zx_{level(z)+1}$ (if z and $x_{level(z)+1}$ are not adjacent in G, then $port_{v\uparrow-out}(z) := \mathsf{nil}$);
- $dev(z)$ is the deviation of z in tree T_2^i;
- $port_{down}(z)$ is the port number at vertex z of the edge zw, where w is the father of z in T_2^i;
- $dev'(z)$ is the deviation of z in tree T_3^i;
- $port_{up}(z)$ is the port number at vertex z of the edge zw, where w is the father of z in T_3^i.

Now we define label $Label(z)$ of a vertex z of G as

$$Label(z) := (\chi(z), L_1(z), L_2(z), \ldots, L_\mu(z)).$$

Clearly, the label size of any vertex of G is at most $\mu+\mu(3\log_2 D+8\log_2 \Delta+4) \leq (\mathsf{an}(G)(\mathsf{an}(G)-1)/2)(3\log_2 D + 8\log_2 \Delta + 5)$ bits.

Assume now that a vertex v wants to send a message to a vertex u. Given the labels $Label(v)$ and $Label(u)$, v first uses the characteristic vectors $\chi(v)$ and $\chi(u)$ to find an index k such that $\chi_k(v) = \chi_k(u) = 1$. Then, v extracts from $Label(u)$ the substring $L_k(u)$ and forms a header of the message $H(u) := k \circ L_k(u)$. Now, the initiated message with the header $H(u) = k \circ L_k(u)$ is routed to the destination using the trees T_1^k, T_2^k, T_3^k: when the message arrives at an intermediate vertex x, vertex x using own substring $L_k(x)$ and the string $L_k(u)$ from the header makes a constant time routing decision by using a function similar (even simpler since we are targeting now deviation 3 only, not 2) to function routing_decision_AT-free (details are omitted again). Thus, we have the following result.

Theorem 13 *Every n-vertex graph G with asteroidal number $\mathsf{an}(G)$, diameter D and maximum vertex degree Δ admits an $(\mathsf{an}(G)(\mathsf{an}(G)-1)/2)(3\log_2 D + 8\log_2 \Delta + 5)$-bit labeling scheme of deviation 3 (and an $(\mathsf{an}(G)(\mathsf{an}(G)-1)/2)(\log_2 D+2\log_2 \Delta+4)$-bit routing labeling scheme of deviation 4). Once computed by the sender in $(\mathsf{an}(G)(\mathsf{an}(G)-1)/2)$ time, headers never change. Moreover, given a repulsive asteroidal set of G the scheme is computable in linear time, and the routing decision is made in constant time per vertex.*

Acknowledgments

DGC wishes to thank the Natural Sciences and Engineering Research Council of Canada for financial assistance and the Department of Computer Science at Kent State University for hosting his visit there.

References

[1] A. Brandstädt, V. B. Le, and J. Spinrad. *Graph Classes: A Survey. SIAM,* Philadelphia, 1999.

[2] L. P. Chew. There are planar graphs almost as good as the complete graph. *J. of Computer and System Sciences*, 39:205–219, 1989.

[3] D. Corneil, S. Olariu, and L. Stewart. Asteroidal triple–free graphs. *SIAM J. Discrete Math.*, 10:399–430, 1997.

[4] D. Corneil, S. Olariu, and L. Stewart. Linear time algorithms for dominating pairs in asteroidal triple–free graphs. *SIAM J. on Computing*, 28:1284–1297, 1999.

[5] F. Dragan, C. Yan, and D. Corneil. Collective tree spanners and routing in AT-free related graphs (extended abstract). In *Proceedings of the 30th International Workshop Graph-Theoretic Concepts in Computer Science (WG '04), Lecture Notes in Computer Science, Springer*, volume 3353, pages 68–80, 2004.

[6] F. Dragan, C. Yan, and I. Lomonosov. Collective tree spanners of graphs. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), Lecture Notes in Computer Science, Springer*, volume 3111, pages 64–76, 2004.

[7] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proceedings of the 28th Intern. Colloq. on Automata, Languages and Program. (ICALP 2001), Lecture Notes in Computer Science, Springer*, volume 2076, pages 757–772, 2001.

[8] C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA'01), Lecture Notes in Computer Science, Springer*, volume 2161, pages 476–487, 2001.

[9] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[10] T. Kloks, D. Kratsch, and H. Müller. Approximating the bandwidth for asteroidal triple-free graphs. *J. of Algorithms*, 32:41–57, 1999.

[11] T. Kloks, D. Kratsch, and H. Müller. On the structure of graphs with bounded asteroidal number. *Graphs and Combinatorics*, 17:295–306, 2001.

[12] A. L. Liestman and T. Shermer. Additive graph spanners. *Networks*, 23:343–364, 1993.

[13] T. H. Ma and J. P. Spinrad. On the two-chain subgraph cover and related problems. *J. of Algorithms*, 17:251–268, 1994.

[14] M. S. Madanlal, G. Venkatesan, and C. Pandu Rangan. Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters*, 59:97–102, 1996.

[15] R. M. McConnell and J. P. Spinrad. Linear-time transitive orientation. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 19–25, 1997.

[16] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Math. Appl. SIAM, Philadelphia, 2000.

[17] D. Peleg and A. A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[18] D. Peleg and J. Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 77–85, 1987.

[19] E. Prisner, D. Kratsch, H.-O. Le, H. Müller, and D. Wagner. Additive tree spanners. *SIAM Journal on Discrete Mathematics*, 17:332–340, 2003.

[20] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the 13th Ann. ACM Symp. on Par. Alg. and Arch*, pages 1–10, 2001.