Università di Pisa
Dipartimento di Informatica

Technical Report: TR-02-08

Distributed Construction of Almost-Edge-Disjoint Spanning Trees via Dense Trees

Fabrizio Luccio and Linda Pagli
Dipartimento di Informatica, Università di Pisa, Italia.

Address: Corso Italia 40, 56125 Pisa, Italy.
Tel: +39 050 2212700
Fax: +39 050 2212726

Distributed Construction of Almost-Edge-Disjoint Spanning Trees via Dense Trees

Fabrizio Luccio and Linda Pagli
Dipartimento di Informatica, Università di Pisa, Italia.

Abstract

A k-dense tree, k integer ≥ 1, is a natural extension of a tree. A leaf is a vertex connected to the rest of the structure through $\leq k$ edges. After the removal of a leaf, the structure still has other leaves. Among other applications, dense trees can be used as interconnection structures in a network. Representing the network in connected graph form $G = (V, E)$, we construct a k-dense tree T as a subgraph of G spanning all its vertices. T is then decomposed in k *super-root spanning trees* sharing a k-clique C of vertices. These trees are edge-disjoint except for the common edges in C. Each vertex is labeled with a set of k integers in $\{0, 1, \ldots, n-1\}$, $n = |V|$, to set up an interval routing scheme for G along the edges of T. The constructions are implemented as distributed algorithms all requiring polynomial time.

Keywords. k-dense tree, spanning tree, SRS-tree, decomposition, interconnection structure, labeling, interval routing, distributed algorithm, distributed computing.

1 Introduction

Crucial issues in distributed systems are the decomposition, distribution, and accessing of data through the variuos sites of a network. Many studies have been directed to the relationship between the topology of the underlying communication network, the way in which data are distributed, and the definition of routing paths for data access and exchange (for example see [13]).

The problems of *labeling* and *compact routing* are related to this issue. In a labeling scheme the n vertices of the network are generally labelled with the integers $0, 1, ..., n-1$ so that relevant information about the graph structure can be inferred directly from the labels. Highly regular topologies like trees, arrays and hypercubes admit good labeling schemes, but some interesting results have also been proved for more general graphs. The aim of compact routing is to avoid storing very large routing tables in each node of the network, with a crucial use of a proper labeling scheme (see for example [7, 8]). The *interval routing scheme* (shortly *IRS*), introduced in [16] and then developed in [10] and in other successive works, is an example of this strategy. Each edge e leaving from a vertex v is associated with one or more intervals in $[0, n-1]$ which specify the vertices that can be reached from v through e. When a message with destination x arrives at vertex $v \neq x$, the routing information stored at v is searched for the interval containing x, and the corresponding edge is selected. The maximum number k of intervals per edge is generally seen as a measure of the overall quality of the scheme, with $k = 1$ corresponding to an optimal IRS. [6] contains an excellent review on the subject.

For routing sensitive data or for broadcasting commands through the network a subset of channels may be used, defined as the edges of a subgraph spanning all the vertices of the network. The subgraph most commonly used is a spanning tree [13]. An optimal IRS can be easily built on it but the failure of one of its edges disconnects the system. A certain degree of fault tolerance is obtained defining a set of edge-disjoint spanning trees on the network [9], or with other more sophisticated connectivity systems [2].

In this paper we study a spanning-tree decomposition, and consequent labeling, on a family of graphs called *dense trees*. These graphs find applications as interconnection structures in a network [11, 13]; to describe catastrophic patterns of faulty processors in distributed systems [15, 1]; and as distributed data structures [12]. Here we refer to the first application. Given a graph $G = (V, E)$ representing a network we construct a dense tree T as a subgraph of G spanning all the vertices in V, in order to route the messages along the edges of T. We construct a decomposition of T in k *super-root spanning trees* sharing a k-clique of vertices (the *super-root*), and establish a labeling scheme where a vertex label is a set of k integers in $\{0, \ldots, n-1\}$, $n = |V|$. This allows to define an IRS for G along the edges of T, where each edge is labeled with exactly one (wrap-around) interval in $[0, n-1]$. All the constructions are done in polynomial time with distributed algorithms.

2 Basic results on dense trees.

As an introduction to dense trees we take some basic definitions and results from [11] where the underlying theory was developed, and from [4] were further results and applications were presented.

Definition 1 *Let $G = (V, E)$ be an undirected and connected graph, and k be an integer, $1 \leq k < |V|$.*

1. *A* k-cycle *C in G is a connected subgraph of G, such that all the vertices in C have degree $> k$. (A standard cycle in G is a 1-cycle).*

2. *A vertex $v \in V$ of degree $\leq k$ is a* leaf *of G (if G is a tree and $k = 1$ this corresponds to the usual definition of leaf).*

3. *If G has no k-cycles, then G is a* k-dense tree*. Equivalently a* k-dense tree *G is a connected graph consisting of a single vertex, or containing a vertex (leaf) v of degree $\leq k$ and the restriction of G to $V - \{v\}$ is a k-dense tree.*

The graph of figure 1 contains several 2-cycles (e.g. a, b, c, d) but no 3-cycle, hence is a 3-dense tree. Definition 1 is relevant for graphs with at least $k + 2$ vertices, where k-cycles may occur. Note that a standard tree is a 1-dense tree, and a k-dense tree is also a $(k+1)$-dense tree although we shall refer to the smallest possible value of k whenever known. Note also that a standard tree with two or more vertices contains at least two leaves, while a $(k \geq 2)$-dense tree may contain only one leaf (this would be the case of the graph in figure 1 if a new vertex were attached to j, e, g thus becoming the only leaf).

For a dense tree with n vertices and m edges we have from [11]:

$$n - 1 \leq m \leq kn - k(k+1)/2.$$

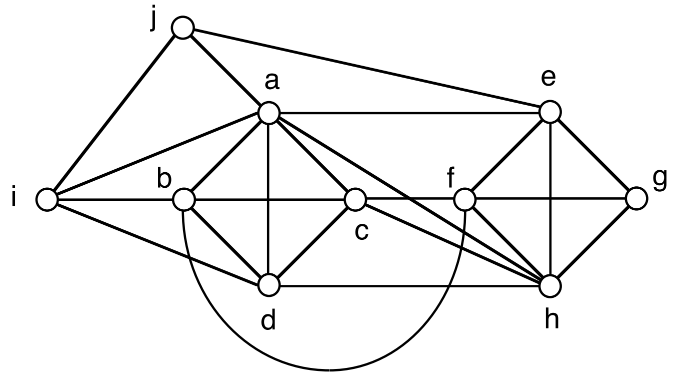

Figure 1: A complete 3-dense tree. Vertices j and g are the leaves.

This relation reduces to $m = n - 1$ for $k = 1$. If the upper bound $m = kn - k(k+1)/2$ is met, the k-dense tree is said to be *complete* (see figure 1, where we have $k = 3$, $n = 10$, $m = 24$). In a complete k-dense tree no two pair of leaves is connected by an edge.

Checking whether a graph G is a k-dense tree for a given k is a polynomial time problem in the family of *elimination sequence* problems, whose members are generally NP-complete [5]. The task can be accomplished as follows:

1. eliminate a vertex of degree $\leq k$ and its incident edges, and update the edge information of the remaining vertices;

2. repeat step 1 until k vertices remain; the restriction of G to this group of vertices is called the *seed* of the elimnination scheme.

Graph G is a k-dense tree iff the process can be brought to its end. If G is complete each vertex eliminated at step 1 has exactly degree k, and the seed is a k-clique.

For the 3-dense tree of figure 1 a possible elimination sequence is: j, i, g, e, f, h, d. The seed is the 3-clique composed of the vertices a, b, c and their connecting edges.

An interesting fact is that a dense tree can be naturally decomposed into a family of spanning subgraphs, that are special extensions of spanning trees. We have from [4]:

Definition 2 *Let T be a k-dense tree.*

1. *A* super-root spanning tree *(* SRS-tree *for short) of T is a subgraph S spanning all the vertices of T, consisting of a k-clique C of T (the "super-root"), and of a set of standard disjoint trees rooted in vertices of C and spanning all the other vertices of T.*

2. *An SRS-*decomposition *for T is a set of SRS-trees covering all the edges of T and sharing exactly their big root.*

An SRS-tree S for the complete 3-dense tree of figure 1 is shown in figure 2 with solid edges. The super-root of S is the clique $\{a, c, d\}$. One tree rooted in a and one tree rooted in d are also contained in S. Two other SRS-trees with the same super-root are shown with dashed and dotted edges, respectively. All together they form an SRS-decomposition.

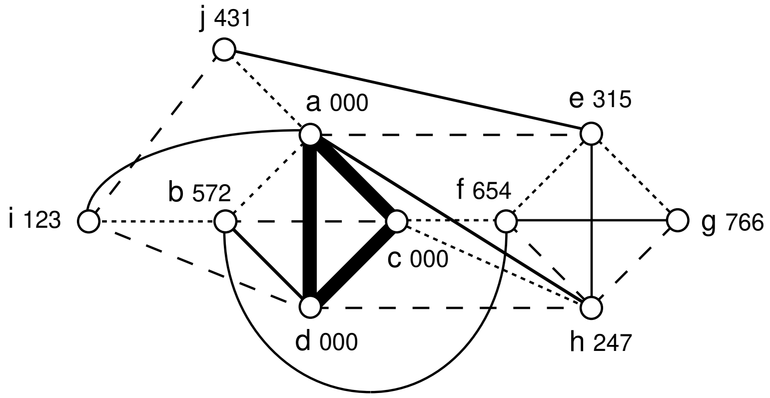

Figure 2: SRS decomposition of the 3-dense tree of figure 1. The three SRS-trees have super-root $\{a, c, d\}$, and are indicated with solid, dashed, and dotted edges, respectively. Numeric labels indicate the positions of each vertex in the SRS-trees (see section 4).

Note that an SRS-tree for T may not exist. It has been shown in [4] that for a complete k-dense tree an SRS-decomposition in k SRS-trees always exists, and a sequential algorithm for its construction has been given. So we start studying the complete case, then discuss the meaning of our results in general.

3 A distributed SRS decomposition

The only distributed algorithms known for k-dense trees were presented in [11] for constructing a 2-dense spanning tree as a subgraph of an arbitrary graph G. In particular if a complete 2-dense tree exixts, it can be built in time quadratic in the size of G, exchanging a quadratic number of messages.

Our first aim here is to build a k-dense tree as a subgraph of G, either for an arbitrary value of k, or for the maximal possible value of k. The first problem can be solved with an extension of the previous algorithm for $k = 2$. The second can be solved with $\log |V|$ iterations of this extended algorithm. Both solutions are rather straightforward and are omitted here. Instead we present below a distributed algorithm 1 to build an SRS decomposition of a complete k-dense tree T. The purpose is building k almost-edge-disjoint spanning trees of an arbitrary graph G admitting T as a subgraph, where T is determined as explained above.

Let $T = (V, E)$. After having assigned a different integer identifier *ID* to each vertex in V, algorithm 1 starts from the leaves and proceeds according to an elimination sequence interpreted in a distributed fashion. Each vertex assigns the incident edges to the SRS-trees under construction, becoming active for this purpose when k of such edges are still to be assigned, and assigning each one of these edges to a different tree. If two active vertices share the same edge and try to assign it to different trees, only the one holding higher identifier *ID* brings the operation to completion. The algorithm terminates when a k-clique (the super-root) is reached. The assignment of colors to edges performed by algorithm 1 corresponds to the assignment of the same edges to the different SRS-trees under construction, one SRS-tree for each color. The edges of the super-root are not colored, and are implicitly assigned to all the SRS-trees.

Algorithm 1 *Distributed decomposition of a complete k-dense tree into k BRS-trees.*
Input*: Complete k-dense tree $T = (V, E)$, $|V| \geq k$, with each vertex displayed in a different processor P_i, and edges displayed in the processors' interconnections.*
Global variables*: $[C_1, \ldots, C_k]$ colors. $[ID_1, \ldots, ID_k]$ integer identifiers.*
Output*: SRS-trees $S_1 = (V, E_1), \ldots, S_k = (V, E_k)$ for T with common super-root $R = (V_R, E_R)$, where $E_1 \cup \ldots \cup E_k = E$, and $E_i \cap E_j = E_R$ for any i, j. Each S_i is denoted by a color C_i assigned to the edges of $E_i - E_R$.*

Preprocessing. *Assign a different identifier to each vertex of V, using any reasonable distributed algorithm. W.l.o.g. let ID_i be assigned to the vertex in processor P_i.*

Code for processor P_i, $1 \leq i \leq |V|$.
Local variables*: ADJ set of non colored edges incident to P_i (initially all the edges incident to P_i are in ADJ). Counter $COUNT$ and flag $STAY$, both initially set to 0.*

while in still (initial) state do
    upon receiving $< act, ID_j >$ **from neighbour** P_j **do**
        if $|ADJ| \neq k$ **then send** $< accept >$ **to** P_j;
    upon receiving $< C_h >$ **from neighbour** P_j **do**
        record *that edge (i, j) is in SRS-tree S_h;*
        delete *edge (i, j) from ADJ;*
    if $(STAY = 0 \wedge |ADJ| = k)$ **then**
        on all $e \in ADJ$ **send** $< act, ID_i >$; **enter active state**;
while in active state do
    upon receiving $< act, ID_j >$ **from neighbour** P_j **do**
        if $ID_i > ID_j$ **then send** $< no - accept >$ **to** P_j
            else send $< accept >$ **to** P_j; $STAY \leftarrow 1$; **enter still state**;
    upon receiving $< no - accept >$ **do** $STAY \leftarrow 1$; **enter still state**;
    upon receiving $< accept >$ **do** $COUNT \leftarrow COUNT + 1$;
    if $COUNT = k$ **then**
        for $h \leftarrow 1$ **to** k **do**
            assign *color C_h to the h-th edge $(i, j) \in ADJ$;*
            send $< C_h >$ *to* P_j; $ADJ \leftarrow \emptyset$; **enter still state**;

To prove the correctness of algorithm 1 we must prove first:

Lemma 1 *Let A be the subset of vertices that have already terminated their coloring activity at a given step of algorithm 1. If two active vertices $u, v \in V - A$ attempt to color the common edge (u, v) we must have $|V - A| \leq k + 1$.*

Proof By inspection of algorithm 1 it can be immediately seen that each vertex in $V - A$ is connected to other vertices in $V - A$ only with non colored edges, and may be connected to vertices in A only with colored edges. By contradiction let $|V - A| > k + 1$. W.l.o.g. let v have a higher identifier ID than u. Since u has k incident non colored edges, one of which is (u, v), there is at least another vertex $x \in V - A$ such that $x \neq u, v$ and $(x, u) \notin E$. If edge (x, u) were added to E, u would have $k + 1$ non colored incident edges and would not be active. v would then color its incident edges including (u, v), and u would start being active then. The algorithm would terminate as before, proving that T with the addition of edge (x, u) is a k-dense tree, against the hypothesis that T is complete. $\square$

We then have:

Theorem 1 *Given a complete k-dense tree $T = (V, E)$, algorithm 1 builds an SRS decomposition for T in $O(|V|)$ time exchanging $\Theta(|E|)$ messages.*

Proof *Correctness.* (i) If $|V| = k$ (T is a k-clique) all vertices have $k - 1$ incident edges and algorithm 1 stops immediately on the super-root.

(ii) If $|V| = k + 1$ all vertices have k incident edges (T indeed is a $(k + 1)$-clique). All the vertices attempt coloring teir incident edges. Only the vertex v with highest ID succeeds, and the k-clique formed by all the others becomes the super-root. v is connected to the super-root with edges of k different colors, thus forming k SRS-trees.

(iii) If $|V| > k + 1$ let us follow the coloring process in the reverse ordering of algorithm 1. Let subset A be defined as in lemma 1, consider a step at which $|V - A| > k + 1$, and let $B \subset V - A$ be the subset of vertices currently active in coloring. By lemma 1 all the vertices in B can complete their coloring activity without conflicts, hence each one of them will be attached to the vertices in $V - A - B$ with edges of k different colors. Assuming inductively that all the vertices in $V - A - B$ form k SRS-trees, also the vertices of $V - A$ form k SRS-trees. The basis of induction is given in points (i) and (ii).

Complexity. The worst case $O(|V|)$ for time complexity occurs if the vertices become active one at a time, and the algorithm becomes essentially sequential except for coloring the edges incident to each vertex that is done in parallel. $\Theta(|E|)$ messages are always exchanged for coloring. Preprocessing is done within the same bounds. □

Assuming that the vertices a, c, d of the super-root have the lowest *ID's*, algorithm 1 generates the SRS decomposition of figure 2. If the original k-dense tree T is not complete, an extension of algorithm 1 generates a decomposition in *SRS-forests*, with obvious meaning. In such a decomposition the super-root still has k vertices, but we do not insist that it be a k-clique.

4 Labeling and routing in SRS-trees

Once an SRS decomposition of T is built, the problems of vertex labeling and interval routing along the SRS-trees become reasonably simple. We propose the labeling procedure of algorithm 2 below. This procedure applied to the SRS decomposition of figure 2 generates the labels shown in the same figure (e.g. vertex j has label 431, that should be written as 4,3,1). Note that the vertices of the super-root have all zero labels.

Algorithm 2 *Labeling the vertices of a complete k-dense tree $T = (V, E)$.*

Input*: SRS-trees $S_1 = (V, E_1), \ldots, S_k = (V, E_k)$ for T with super-root $R = (V_R, E_R)$.*

Output*: A numeric label L_v assigned to each vertex $v \in V$. L_v is a sequence of k integers $\ell_1, \ldots, \ell_k$, where ℓ_i is relative to S_i.*

Procedure.

  for $i \leftarrow 1$ **to** k **do**

    traverse in parallel S_i *in preorder, to assign the i-th component ℓ_i to the label of each vertex; assign the value 0 to the vertices of R, and consecutive integers $1, \ldots, |V| - k$ to the other vertices, in the order in which they are encountered during the traversal.*

Algorithm 2 clearly requires $O(|V|)$ time and exchanges $\Theta(|E|)$ messages. Once the vertices have been labeled, the routing intervals on the edges are determined as for standard interval

edge	solid	dashed	dotted		edge	solid	dashed	dotted
ab			[2,3]		ha	[5,1]		
ac		[7,7]	[4,7]	$\Leftarrow$	hc			[0,6]
ad	[5,7]	[2,6]		$\Leftarrow$	hd		[7,3]	
ae		[1,1]			he	[3,4]		
ah	[2,4]				hf		[5,5]	
ai	[1,1]				hg		[6,6]	
aj			[1,1]					

Table 1: Routing intervals on the edges incident to vertices a and h, for the SRS decomposition of figure 2. The edges of the super-root, like (a, c) and (a, d), may have one interval for each SRS-tree.

routing for trees, applied to the SRS-trees taken one by one. Table 4 shows the intervals on a set of sample edges for our running example, namely the ones incident to a (vertex of the super-root) and to h. Note that each edge carries exactly one (possibly wrap-around) interval relative to the SRS-tree to which it belongs, except for the edges of the super-root that may carry up to k intervals, as they belong to all the SRS-trees. We have:

Theorem 2 *Given a complete k-dense tree $T = (V, E)$ and the SRS-trees $S_1 = (V, E_1), \ldots$, $S_k = (V, E_k)$ of a decomposition of T with super-root $R = (V_R, E_R)$. The labeling produced by algorithm 2 induces single routing intervals on the edges of $E - E_R$, and up to k routing intervals on each edge of E_R.*

Proof Let $U = [0, |V| - k]$. Consider a vertex v, an SRS-tree S_i, and let p be the ℓ_i component of the label of v.

(i) Let $v \in V - V_R$. The ℓ_i components of the labels of all the vertices of the subtree of S_i rooted in v form an interval $I \subset [p, |V| - k]$. The edge incident to v and leading to the super-root R carries the (wrap-around) interval $U - I$. (E.g. interval $[5, 1]$ on edge (h, a) of our example is formed as $[0, 7] - [2, 4]$). Each edge leading from v to one of its subtrees S' carries the interval containing the ℓ_i components of the vertices of S'.

(i) Let $v \in V_R$. Each edge of E_i, if any, leading from v to a subtree $S' \subset S_i$ outside R carries the interval containing the ℓ_i components of the vertices of S'. Any other subtree $S'' \subset S_i$ outside R is accessed from another vertex $w \in V_R$. Being a clique, R contains the edge (v, w), therefore in the routing table of v, edge (v, w) will carry the interval relative to S''. $\Box$

5 Concluding Remarks

In this paper we have seen how k-dense trees can be embedded in a distributed system, where they play the role of "reinforced" spanning trees of the network, allowing some degree of fault tolerance. Limiting our study to a complete k-dense trees T, we have presented a distributed protocol to build a decomposition of T in BRS-trees that are extensions of the standard spanning trees. BRS-trees are edge-disjoint, except for the edges of a common super-root of k vertices. The purpose is building k almost-edge-disjoint spanning trees of an arbitrary graph G admitting T as a subgraph. An additional distributed protocol for vertex labeling leads to an almost optimal interval routing scheme for T.

The proposed strategy is efficient if G contains a complete k-dense tree. The extension to arbitrary graphs, although simple, has been merely sketched for brevity and deserves further attention.

References

[1] de la Noval Y., Luccio F., Nuñez Rodriguez Y. and Pagli L. Dynamos in Three-Dimensional Meshes, *Proc. 7-th. International Congress on Computer Science Research (ISADS2000)*, Tampico (2000).

[2] S. Even, G. Itkis, and S. Rajsbaum. On mixed connectivity certificates. *Theoretical Computer Science* 203 (1998) 253–269.

[3] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM* 29 (1982) 24-32.

[4] L. Flores Pacheco, G. Franceschini, F. Luccio, and L. Pagli. Decomposition of k-Dense Trees", *Proc. Distributed Data and Structures 3.* Carleton Scientific, Ottawa 2001, 11-23.

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco 1979.

[6] C. Gavoille A survey on interval routing schemes. *Theoretical Computer Science* (2000) 217–253.

[7] C. Gavoille Routing in distributed networks: overview and open problems. *ACM SIGACT News - Distributed Computing Column* (2001) 36–52.

[8] C. Gavoille, D. Peleg, S. Perennes, R. Raz. Distance labeling in graphs. *Proc. SODA 01* (2001) 210-219.

[9] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation* 79,1 (1988) 3.59.

[10] J. van Leeuwen and R. Tan. Interval Routing *The Computer Journal*, 30 (1987) 298-307.

[11] F. Luccio, and L. Pagli, Dense Trees: a New Structure for Interconnection, *Distributed Data and Structures 2.* Carleton Scientific, Ottawa 2000, 56-72.

[12] Luccio F., Mesa A. and Pagli L., A distributed tree data structure, *Proc. 1-st Internat. Symp. on Advanced Distributed Systems (ISADS2000)*, Guadalajara, Mexico (2000) 1–6.

[13] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publ. Inc., San Francisco 1996.

[14] D.W. Matula. A min-max theorem for graphs with applications to graph coloring. *SIAM Rev.* 10 (1968) 481-482.

[15] D. Peleg. Local majority voting, small coalitions and controlling monopolies in graphs: A review. In *Proc. of 3rd Colloquium on Structural Information and Communication Complexity* (1996) 152-169.

[16] M. Santoro and R. Khatib. Labeling and Implicit Routing in Networks. *The Computer Journal* 28 (1985) 5-8.