

## A multi-tree routing scheme using acyclic orientations

Fred S. Annexstein<sup>a,1</sup>, Kenneth A. Berman<sup>a,1,\*</sup>, Tsan-Sheng Hsu<sup>b</sup>,  
Ram Swaminathan<sup>c</sup>

<sup>a</sup>*Department of ECE and Computer Science, University of Cincinnati, P.O. Box. 210030,  
Cincinnati, OH 45221, USA*

<sup>b</sup>*Institute of Information Science, Academia Sinica, Nankang 11529, Taipei, Taiwan,  
People's Republic of China*

<sup>c</sup>*Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974, USA*

---

### Abstract

We propose a mathematical model for fault-tolerant routing based on acyclic orientations, or acorns, of the underlying network  $G = (V, E)$ . The acorn routing model applies routing tables that store the set of parent pointers associated with each out-neighborhood defined by the acorn. Unlike the standard single-parent sink-tree model, which is vulnerable to faults, the acorn model affords a full representation of the entire network and is able to dynamically route around faults. This fault tolerance is achieved when using the acorn model as a multi-tree generator for gathering data at a destination node, as well as an independent tree generator for global point-to-point communication. A fundamental fault-tolerant measure of the model is the capacity of an acorn, i.e., the largest integer  $k$  such that each vertex outside the neighborhood  $N(v)$  of the destination  $v$  has at least  $k$  parent pointers. A capacity- $k$  acorn  $A$  to destination  $v$  is  $k$ -vertex fault-tolerant to  $v$ . More strongly, we show  $A$  supports a  $k$  independent sink-tree generator, i.e., the parent pointers of each vertex  $w \in V - N(v)$  can be partitioned into  $k$  nonempty classes labeled  $1, 2, \dots, k$  such that any set of sink trees  $T_1, T_2, \dots, T_k$  are pairwise independent, where tree  $T_i$  is a sink tree generated by parent pointers labeled  $i$  together with the parent pointers into  $v$ . We present an linear time optimization algorithm for finding an acorn  $A$  of maximum capacity in graphs, based upon a minimax theorem. We also present efficient algorithms that label the parent pointers of capacity- $k$  acorn  $A$ , yielding a  $k$ -independent sink tree generating scheme. © 2000 Elsevier Science B.V. All rights reserved.

---

\* Corresponding author.

*E-mail addresses:* fred.annexstein@uc.edu (F.S. Annexstein), ken.berman@uc.edu (K.A. Berman), tshsu@iis.sinica.edu.tw (T.-S. Hsu), swamy@bell-labs.com (R. Swaminathan).

<sup>1</sup>Supported in part by an Ohio Board of Regents Research Investment Grant, and NSF Grant No. CCR-9877139.

## 1. Introduction

Routing schemes for communication networks often rely on local tables that associate with each destination a parent link on which to forward messages to the destination [3, 6, 5]. The set of parent links form a directed *sink tree* with a single destination sink. Such tables suffer from the fact that they do not provide a full representation of the entire network, and therefore are vulnerable to faults and other dynamic network changes.

In this paper we propose a mathematical model for network routing that generalizes sink-tree-based schemes. We call the model *acorn routing* for it is based on acyclic orientations of the underlying network. Acorn routing provides dynamic fault-tolerant tree-path generation in the sense that packet forwarding to a destination can proceed in a manner that is oblivious to the source and previous packet history. The acorn model achieves high fault-tolerance by employing routing tables that store the set of parent pointers associated with each out-neighborhood defined by the acyclic orientation. Unlike the sink-tree model, which is vulnerable to faults, the acorn model affords a full representation of the entire network and is able to dynamically and obliviously route around faults. Not only does the acorn model achieve high fault-tolerance for gathering data at a destination node when used as a multi-tree generator, but with minimal additional space requirements it achieves fault-tolerance for global point-to-point communication when used as an independent tree generator.

Let  $G = (V, E)$  be a graph and  $v \in V$ , a *v-acorn* is an acyclic orientation of  $G$  with a single sink  $v$ , where we assign to each edge  $e \in E$  a direction so that the resulting directed graph is acyclic. A distributed representation of a *v-acorn* consists of storing at each vertex  $w$  the set of parent pointers corresponding to the edges directed out of  $w$ . Each edge of the original network is stored precisely once in this representation. The *v-acorn* acts as a multi-tree generator in the sense that a sink tree to  $v$  is obtained when each vertex chooses arbitrarily a parent pointer.

In this paper we investigate some fault-tolerant properties associated with *v-acorns*. Since every acyclic orientation of a graph produces a vertex with only a single parent pointer, it is meaningful to consider the neighborhood of the vertex  $v$  as a destination set and consider the fault-tolerance to this set. Every acyclic orientation of a graph also contains the same number of parent pointers. However, an imbalance in the distribution of such parent pointers can leave those nodes with few parent pointers vulnerable and may negatively affect the global fault-tolerant capabilities. We seek those orientations that can achieve balanced routing schemes by maximizing the minimum cardinality of parents sets. We call this maximum cardinality the *v-acorn capacity*  $\kappa_v(G)$  of the graph  $G$ . We define the *capacity* of a particular *v-acorn*  $A$  to be the largest integer  $k$  such that each vertex outside the destination set has at least  $k$  parent pointers. Thus, the *v-acorn capacity*  $\kappa_v(G)$  of a graph  $G$  is the maximum capacity over all *v-acorns* of  $G$ .

A capacity- $k$  *v-acorn*  $A$  is  $k$  vertex fault-tolerant to  $v$ ; even more strongly, in the presence of up to  $k - 1$  vertex faults simultaneously present in the neighborhood

of each vertex outside the destination set,  $A$  will generate a sink tree (spanning the fault-free vertices) to  $v$ . In Section 2 we present a linear time optimization algorithm for constructing a  $v$ -acorn of maximum capacity, thus determining the  $v$ -acorn capacity of a graph. The correctness of the algorithm follows from a minimax theorem (Theorem 1).

For the reasons described above, acorns of maximum capacity are well suited to facilitate *gather* routing operations which collect data at a single destination vertex. In addition, they provide an effective model for fault-tolerant global point-to-point communication. This is because, as we show in Section 3, acorns of capacity  $k > 1$  support the generation of  $k$  independent sink trees. See [10, 1, 2] for previous work on independent sink trees and fault-tolerant global point-to-point communication. A pair of sink trees to  $v$  are *independent sink trees* if they have the property that for each node  $w \neq v$ , the pair of tree-paths from  $w$  to  $v$  are internally node-disjoint. The independent sink-tree generation of a capacity- $k$   $v$ -acorn routing model is accomplished by partitioning the parent pointers of each vertex  $w \in V - N(v)$  into  $k$  nonempty classes labeled  $1, 2, \dots, k$  such that any set of sink trees  $T_1, T_2, \dots, T_k$  are pairwise independent, where tree  $T_i$  is a sink tree generated by parent pointers labeled  $i$  together with the parent pointers into  $v$ . We present in Section 3 efficient algorithms that label the parent pointers of a capacity- $k$  acorn, yielding such a  $k$ -independent sink-tree generator.

## 2. Algorithm for maximizing acorn capacity

Given a graph  $G = (V, E)$  with vertex  $v \in V$ , let  $N(v)$  denote the neighborhood of  $v$ . Recall the *acorn capacity*  $\kappa_v(G)$  of the graph is the largest integer  $k$ , over all  $v$ -acorns  $A$  of  $G$ , such that each vertex outside  $N(v)$  has at least  $k$  parent pointers in  $A$ . We show that the acorn capacity of  $G$  is related to the size of a certain cutset. An  $N(v)$ -cut,  $\Gamma = (X, X^c)$ , is a partition of the vertex set such that  $N(v) \subset X$ . The *measure* of the cut  $\mu(\Gamma)$  is the maximum neighborhood size of a vertex in  $X^c$  intersecting the cut, i.e.,  $\mu(\Gamma) = \max\{|N(u) \cap X| : u \in X^c\}$ . Below we prove that the acorn capacity is equal to the minimum measure  $N(v)$ -cut. First, we show that the measure of an  $N(v)$ -cut is an upper bound on the capacity.

**Proposition 1.** *The acorn capacity  $\kappa_v(G)$  of a graph  $G$  is no greater than the measure of any  $N(v)$ -cut  $\Gamma$  of  $G$ , i.e.,  $\kappa_v(G) \leq \mu(\Gamma)$ .*

**Proof.** Let  $A$  be a  $v$ -acorn of  $G$ , and let  $\Gamma = (X, X^c)$  be any  $N(v)$ -cut. It follows that  $A$  restricted to vertex set  $X^c$  is a directed acyclic graph (or dag), and this dag must have at least one sink vertex  $w$ . The entire out-neighborhood in  $A$  of this sink vertex must therefore be contained in  $X$ . Hence, by the definitions  $\kappa_v(G) \leq \mu(\Gamma)$ .  $\square$

We now give a constructive proof that there exists a  $v$ -acorn that achieves capacity equal to the minimum measure cut. The proof is based on a constructive algorithm that returns an acorn of maximum capacity.

**Theorem 1.** *The acorn capacity  $\kappa_v(G)$  is equal to the minimum measure of an  $N(v)$ -cut of  $G$ . Moreover, there is an algorithm that constructs a maximum capacity acorn in linear time  $O(|V| + |E|)$ .*

**Proof.** It follows from Proposition 1 that we need only show that there exists some acorn that achieves capacity at least equal to the measure of some  $N(v)$ -cut of  $G$ . The following algorithm returns an acorn and a cut with this property.

---

**Procedure Max-Acorn-Capacity** ( $G, v$ )

*Input:* A graph-vertex pair ( $G, v$ )

*Output:* A maximum capacity  $v$ -acorn  $A$  and a minimum measure  $N(v)$ -cut  $\Gamma = (X, X^c)$  of  $G$ .

*Step 1.* Mark all vertices in  $N(v)$ . Set  $cutmeasure = \infty$ , and  $X = N(v)$ .

*Step 2.* Greedily choose  $w$ , so that  $w$  is not yet marked, and among vertices not yet marked,  $w$  is adjacent to the largest number  $k_w$  of vertices that have been marked. Add to  $A$  the  $k_w$  edges of  $w$  oriented towards the marked vertices. Mark  $w$ .

*Step 3.* If  $cutmeasure \geq k_w$   
then set  $cutmeasure = k_w$ , and set  $X$  equal to the current set of marked vertices.

*Step 4.* If any unmarked vertices remain, then goto Step 2  
else return  $A$  and  $\Gamma = (X, X^c)$ .

---

The algorithm **Max-Acorn-Capacity** maintains the invariant that the value of  $cutmeasure$  is always a lower bound on the size of each out-neighborhood in  $A$  of the marked vertices in  $V - N(v)$ . Hence, it is clear that when the algorithm terminates, the acorn  $A$  has capacity equal to the final value of  $cutmeasure$ . Hence, from Proposition 1 it follows that the constructed acorn  $A$  is of maximum possible capacity.

The running time of the algorithm is  $O(|E| + |V| \log |V|)$  if an ordinary priority queue data structure is used to store the unmarked vertices. The time complexity can be made linear in the size of  $|V| + |E|$  by using the following approach. First, we allocate  $|V|$  buckets. For each  $i$ , the  $i$ th bucket stores the unmarked vertices that currently are adjacent to exactly  $i$  marked vertices. Use an index pointer  $r$  to point to the nonempty bucket with the largest index. The buckets are updated when an unmarked vertex becomes marked, at which point the newly marked vertex is removed, and the remaining vertices stay unchanged or moved ahead to the next

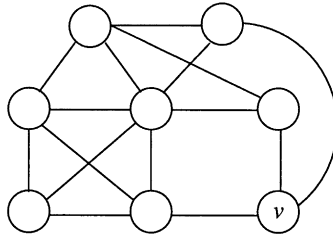


Fig. 1. Example graph  $G$  and destination node  $v$ .

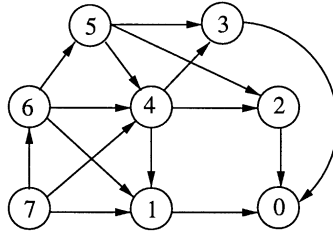


Fig. 2. Resulting  $v$ -acorn  $A$  of  $G$  with maximum capacity  $3 = \kappa_v(G)$ . The vertices are labeled by the order in which they are marked by the algorithm, and consistent with a topological ordering of  $A$ .

largest indexed bucket depending on its adjacency to the newly marked vertex. In addition, the pointer  $r$  is incremented to  $r + 1$ , stays unchanged, or decremented to some  $r' < r$ .

It takes  $O(|V|)$  time to initialize the buckets. Each edge is visited once in the algorithm and can cause the relocation of one vertex in the buckets. Hence it takes  $O(|E|)$  time to relocate the vertices in the buckets for the entire execution of the algorithm. The movement of the pointer takes  $O(|V|)$  time, since the pointer can be incremented and decremented at most  $O(|V|)$  times through the course of the algorithm. Hence, the overall running time of the algorithm is thus  $O(|V| + |E|)$ .  $\square$

We remark that the algorithm **Max-Acorn-Capacity** marks the vertices in an order consistent with a topological ordering of the acorn. Figs. 1 and 2 show an example graph and the maximum capacity acorn returned by the algorithm.

### 3. Acorns and independent sink-tree generators

In this section we show that we can label the parent pointers of a capacity- $k$  acorn  $A$  to support a  $k$ -independent sink-tree generator, as described in Section 1. First, we generate a special labeling of the vertices with  $k - 1$ -tuples. The vertex labeling will induce the labeling of the parent pointers. The labeling of parent pointers has the property that any sink tree generated using parent pointers of one label is independent

from any other sink tree generated using parent pointers of a different label. In the proof of the following theorem we present efficient algorithms for the labeling of the vertices and the parent pointers, yielding a  $k$ -independent sink tree generator.

**Theorem 2.** *Given a capacity- $k$  acorn  $A$  to node  $v$ , the parent pointers of each vertex  $w \in V - N(v)$  can be partitioned into  $k$  nonempty classes labeled  $1, 2, \dots, k$  such that any set of sink trees  $T_1, T_2, \dots, T_k$  are pairwise independent, where tree  $T_i$  is a sink tree generated by parent pointers labeled  $i$  together with the parent pointers into  $v$ . Furthermore, there is a algorithm that computes this labeling in time  $O(k^2|V| + k|E|)$  time requiring  $k|V|$  numbers each with  $O(|V|)$  bits. There is a modified labeling algorithm that runs in  $O(k|V|^2 + k^2|V| + |E|)$  time requiring  $k|V|$  integers of  $O(\log |V|)$  bits each.*

**Proof.** To produce a labeling of the parent pointers that yields an independent sink-tree generator described in the theorem, we present an algorithm for a special labeling of the vertices; and from this vertex labeling the labeling of parent pointers is evidently extracted. Each vertex  $w \in V$  will be labeled  $\mathcal{L}(w) = (L_1(w), \dots, L_{k-1}(w))$ , that is with a  $(k - 1)$ -tuple of real numbers. This labeling will have the property that coordinate values are distinct, that is, given any pair of vertices  $w \neq w'$ , for each  $i$  the labels  $L_i(w) \neq L_i(w')$ .

From this vertex labeling we can induce the labeling of parent pointers as follows: an edge  $w \rightarrow w'$  is labeled  $i$  (where  $1 \leq i \leq k$ ) if  $L_j(w) > L_j(w')$  for all  $1 \leq j < i$ , and  $L_i(w) < L_i(w')$ . For the case  $i = k$  we ignore the second condition. Such a labeling has the property that for each  $i$ , the parent pointers labeled  $i$  yield paths that have monotonically decreasing values in the 1st through  $i - 1$ st coordinates, and monotonically increasing  $i$ th coordinates. From this property it follows that the induced labeled sink trees are indeed independent sink trees, since  $i$ -labeled paths must necessarily be internally node disjoint from  $j$ -labeled paths when these paths originate at the same vertex, for  $i \neq j$ .

To algorithmically produce this vertex labeling, we first perform a topological ordering of the vertices of the acorn with the constraints that  $v$  is numbered  $v_0$  and the vertices of  $N(v)$  are numbered from  $v_1$  to  $v_{|N(v)|}$ . We begin by labeling the vertex  $w = v_j \in N(v)$ , with the topological number  $j$ , with the  $(k - 1)$ -tuple  $(j, \dots, j)$ . Throughout the algorithm, with each new label generated, we maintain the property that coordinate values are distinct. To achieve this we set new label values to lie between two existing values. We use tables  $Next_i$  to record values. The table entry  $Next_i(w)$  returns the value of the next largest real number label value in the  $i$ th component after the label  $L_i(w)$  of vertex  $w$ . The main step of the algorithm determines an ordering of the parent pointers for each vertex by sorting them in a way that can insure that a nonempty partitioning is achieved.

**Procedure Independent-Tree-Generator-Labeling (A)**

*Input:* A capacity- $k$   $v$ -acorn  $A$ .

*Output:* For each vertex  $w \in V$  a  $(k - 1)$ -tuple labeling  $\mathcal{L}(w) = (L_1(w), L_2(w), \dots, L_{k-1}(w))$  of  $w$ , along with a labeling of all parent pointers from the set  $\{1, 2, \dots, k\}$ , yielding an  $k$ -independent sink-tree generator.

*Step 0:* Perform a topological ordering  $\{v_0, v_1, v_2, \dots, v_{|V|}\}$  of the vertices of the acorn  $A$  with  $v = v_0$  and the vertices of  $N(v)$  ordered from  $v_1$  to  $v_{|N(v)|}$ .

*Step 1:* For  $j = 1$  to  $|N(v)|$  do  
     label vertex  $v_j$  with the  $(k - 1)$ -tuple  $(j, \dots, j)$   
     For  $i = 1$  to  $k - 1$  do  
         set  $Next_i(v_j) = j - 1$ .

*Step 2:* (Main Loop) For  $j = |N(v)| + 1$  to  $|V|$  do let  $w = v_j$

*Step 2a:* Choose  $k$  out-neighbors of  $w$ , and  
     order these vertices  $\{w_1, w_2, \dots, w_k\}$  to satisfy the following property:  
     for each  $1 \leq i \leq k$ ,  $w_i$  has the property that  $L_i(w_i)$   
     is the maximum among  $\{L_i(w_i), L_i(w_{i+1}), \dots, L_i(w_k)\}$ .

*Step 2b:* For  $i = 1$  to  $k - 1$  do  
     Set  $L_i(w) = (L_i(w_i) + Next_i(w_i))/2$   
     Set  $Next_i(w) = Next_i(w_i)$ , and set  $Next_i(w_i) = L_i(w)$ .

*Step 2c:* For  $i = 1$  to  $k$  do  
     Set label of parent pointer  $w \rightarrow w_i$  to  $i$   
     For all other parent pointers  $w \rightarrow w'$  we label it  $i$  iff  
      $L_i(w) < L_i(w')$  and  $L_j(w) > L_j(w')$ , for all  $j < i$ .

The correctness of the procedure **Independent-Tree-Generator-Labeling** follows from noting that the ordering of neighbors in Step 2a is always possible when component values are distinct. The distinctness of component values is assured by Step 2b. The proof of the theorem follows from the correctness of the above procedure.

It is clear that the algorithm runs in  $O(k^2|V| + k|E|)$  time. One drawback is that the algorithm requires the use of  $k - 1$  real numbers for each vertex label. It is possible that the number of bits required for such numbers is  $O(|V|)$ . We now present a slightly modified algorithm that runs in  $O(k^2|V| + k|V|^2 + k|E|)$  time, but uses only integers of  $O(\log |V|)$  bits each. To achieve this complexity we modify Step 2b as follows.

*Step 2b':* For  $i = 1$  to  $k - 1$  do  
     Set  $L_i(w) = L_i(w_i)$ ,  
     and for all vertices  $w' \neq w$  that have been labeled do  
     if  $L_i(w') \geq L_i(w)$ , then set  $L_i(w') = L_i(w') + 1$ .

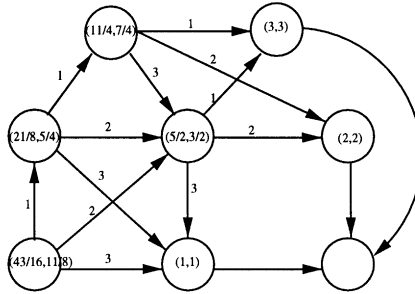


Fig. 3. The vertex and parent pointer labelings obtained by applying the procedure *Independent-Tree-Label* ( $A$ ), where  $A$  is the acorn of Fig. 2. The labeling results in a 3-independent sink-tree generator.

The correctness of the modified algorithm follows from the fact that distinct coordinate values are maintained. The claimed space complexity follows by induction, since after the  $j$ th vertex  $v_j$  is processed and labeled, the set of integers used for each coordinate values is in the range  $\{1, \dots, j\}$ .  $\square$

Fig. 3 shows the results of applying the previous algorithm using real numbers to the acorn of Fig. 2.

The following Corollary follows immediately from Theorem 2. The result was proved independently by Huck [9].

**Corollary 1.** *In an acyclic digraph that is  $k$ -connected to  $v$ , that is, for each vertex  $w \neq v$  there are  $k$  openly disjoint paths from  $w$  to  $v$ , there exist  $k$  pairwise independent sink trees directed towards  $v$ .*

**References**

[1] F.S. Annexstein, K.A. Berman, Directional routing via generalized st-numberings, University of Cincinnati Technical Report, 1999, submitted for journal publication.  
 [2] F. Bao, Y. Igarashi, S.R. Ohring, Reliable broadcasting in product networks, *Discrete Appl. Math.* 83 (1–3) 1998 3–20.  
 [3] D. Bertsekas, R. Gallager, *Data Networks*, Prentice-Hall, New York 1992.  
 [4] J. Cheriyan, S.N. Maheshwari, Finding non-separating induced cycles and independent spanning trees in 3-connected graphs, *J. Algorithms* 9 (1988) 507–537.  
 [5] R. Cohen, B.V. Patel, F. Schaffa, M. Willebeek-LeMair, The sink tree paradigm: connectionless traffic support on ATM LAN's, *IEEE/ACM Trans. on Networking* 4 (1996) 363–374.  
 [6] E.M Gafni, D.P. Bertsekas, Distributed algorithms for generating loop-free routes in networks with frequently changing topology, *IEEE Trans. Commun.* COM-29 (1) (1981) 11–18.  
 [7] Garey, Johnson, *Computers and Intractibility*, W.H. Freeman and Company, New York, 1977.  
 [8] A. Huck, Disproof of a conjecture about Independent branchings in  $k$ -connected directed graphs, *J. Graph Theory* 20 (1995) 235–239.  
 [9] A. Huck, Independent branchings in acyclic digraphs, 1996, submitted manuscript.  
 [10] A. Itai, M. Rodeh, The multi-tree approach to reliability in distributed networks, *Inform. Comput.* 79 (1988) 43–59.  
 [11] A. Zehavi, A. Itai, Three tree-paths, *J. Graph Theory* 13 (1989) 175–188.