Implicit Representation of Graphs*

Sampath Kannan
Computer Science Department
University of Arizona

Moni Naor
IBM Research Division
Almaden Research center

Steven Rudich
Computer Science Department
Carnegie-Mellon University

Abstract

How to represent a graph in memory is a fundamental data structuring question. In the usual representations of an n-vertex graph, the names of the vertices (i.e. integers from 1 to n) betray nothing about the graph itself. Indeed, the names (or labels) on the n vertices are just $\log n$ bit place holders to allow data on the edges to encode the structure of the graph. In our scenario, there is no such waste. By assigning $O(\log n)$ bit labels to the vertices, we completely encode the structure of the graph, so that given the labels of two vertices we can test if they are adjacent in time linear in the size of the labels. Furthermore, given an arbitrary original labeling of the vertices, we can find structure coding labels (as above) that are no more than a small constant factor larger than the original labels. These notions are intimately related to vertex induced universal graphs of polynomial size. For example, we can label planar graphs with structure coding labels of size $< 4\log n$, which implies the existence of a graph with n^4 vertices that contains all n-vertex planar graphs as vertex induced subgraphs. The theorems on finite graphs extend to a theorem about the constrained labeling of infinite graphs.

Key Words: graph representation, universal graphs, arboricity, intersection graphs, data structures.
Abbreviated title: Implicit Representation of Graphs
AMS MOS Classification: 05C75 68P05 68R10

*Research supported by NSF Grant #DCR-85-13926. A preliminary version of this paper appeared in the Proceedings of the 20th ACM Symp. Theory of Computing

1 Introduction

We will consider graphs $G = (V, E)$ with vertex set V and edge set $E \subseteq (V \times V)$. The graphs we consider will not have any self-loops, i.e. edges of the form (v, v). We will also disallow parallel edges between two vertices as the definition of E above indicates.

Definition 1 *A* vertex induced subgraph *or simply an* induced subgraph *G' of G is a vertex set $V' \subseteq V$ together with the edge set $E' = E \cap (V' \times V')$.*

Definition 2 *An* edge induced subgraph *or simply a* subgraph *is a subset $V' \subseteq V$ together with an edge set $E' \subseteq E \cap (V' \times V')$.*

Consider the following problems:

Problem 1 Label the vertices of an n-vertex tree with labels that are $O(\log n)$ bits long such that given the name of two vertices, one can determine adjacency quickly.

Solution 1 Root the tree. Prelabel each vertex with arbitrary, distinct, positive integers. Let the label on each vertex be its prelabel appended with the prelabel of its parent. The same procedure works for infinite trees with infinite degrees as well.

Problem 2 Given a tree prelabeled with distinct positive integers (imagine an adversary choosing the prelabels), label the vertex prelabeled i with $O(\log i)$ bits, so that given two vertices one can determine adjacency quickly.

Solution 2 Root the tree. Each vertex will receive a two part label. For the first part of its label each vertex takes on the prelabel of its smallest child or itself, whichever is smaller. There is also a flag bit in the first part to indicate if the prelabel remembered is its own or a child's. The second part of each vertex's label will be the first part of its parent's label. A vertex with prelabel i is labeled with at most $2(\log i + 1)$ bits. Given any two vertices there is a linear time procedure to tell adjacency. Two vertices are adjacent iff the first part of one vertex's label is the same as the second part of the other vertex's label.

Problem 3 Give an efficient construction of an $O(n^2)$ graph that contains all n-vertex trees as vertex induced subgraphs.

Solution 3 The following works for forests as well: let the vertices be the ordered pairs (i, j), where $1 \leq i, j \leq n$. Place an edge between (i, j) and (i', j'), iff $i = j'$ or $j = i'$.

For a wide variety of graphs these three problems are essentially equivalent. We will formalize the notions above and present them in a more general setting.

2 Definitions

Definition 3 *A family F of finite graphs has a k labeling scheme if there is a polynomial time Turing Machine T, and a function f which labels the vertices of each graph G in F with distinct labels of no more than $k \log n$ bits (n is the number of vertices of G), such that given two vertex labels of a graph G in F, T will correctly decide adjacency of the corresponding vertices in G.*

If a family F has a k-labeling scheme for some k, we say that it has a *labeling scheme.*

For a family of graphs to have a labeling scheme is clearly a desirable property. It enables representing graphs which are members of the family implicitly, just by having available the vertex names. For instance, in a distributed system the graph need not be stored in one place. But whenever a processor needs to determine adjacency, it can do so only by looking at the names of the two vertices. If the vertices of a graph were ordered by the frequency of their usage, we would want to assign the more frequent vertices a shorter label. In general we could have other constraints on the size of the vertex labels. Hence we define constrained labeling.

Definition 4 *A family of graphs, F, has a* constrained k-labeling scheme *if $\forall G \in F$, $|G| = n$, $\forall p$ such that p assigns distinct positive integers to the vertices of G, we can associate $k\lceil \log p(v) \rceil$ bits with each vertex v and use these labels to decide adjacency efficiently.*

As mentioned above labeling schemes are related to vertex induced universal graphs which we define:

Definition 5 *Given a finite set of graphs S, a graph G is* vertex induced universal *for S, if every graph in S is a vertex induced subgraph of G. We say a family F has universal graphs of size $g(n)$, if for every n, there is a graph of size less than or equal to $g(n)$, which is vertex induced universal for the set of all graphs in F with n or fewer vertices.*

It is possible to define the *edge induced* universal graph of a set of graphs S as a graph G which contains each graph in S as an edge induced subgraph. However in our paper we focus on vertex induced universal graphs. So we will refer to vertex induced universal graphs simply as *universal graphs.*

The rest of the paper deals with the relationship among the three concepts defined above. In Section 3 we give labeling schemes for a wide variety of graph families. Section 4 discusses constrained labeling and shows that under a very weak hypothesis, all families which have labeling schemes, also have constrained labeling schemes. Section 5 deals with the implications of labeling on the construction of universal graphs. Section 6 and 7 outline some of the related work and open problems.

3 Labelable Families

In this section we show that a wide variety of families of graph have labeling schemes. We start by showing that trees and transitive closures of trees have labeling schemes. These two schemes will be very useful as building blocks in labeling schemes we give for other families. We then give a labeling scheme for graphs of bounded arboricity which can be viewed as "uniformly sparse" graphs. This yields a 4-labeling scheme for planar graphs. Next we present a general technique that applies to many families of intersection graphs. Finally we give a labeling scheme for c-decomposable graphs, i.e. graphs which have small separators, that decompose the graph.

Not all families of graphs are labelable. Since, we allow only $O(n \log n)$ bits to represent an n-vertex graph, we can represent at most $2^{O(n \log n)}$ graphs. Hence we have

Proposition 1 *A family of graphs that contains more than $2^{\Omega(n \log n)}$ n-vertex graphs cannot be labeled.*

As a corollary bipartite graphs and chordal graphs are not labelable.

3.1 Trees and Transitive Closures of Trees

In this subsection we give a 2-labeling scheme for trees and transitive closures of trees. These two schemes will be used extensively as building blocks in the rest of section 3.

Finite Trees: We give a 2-labeling scheme for trees (not necessarily bounded degree): Given an n-vertex tree, arbitrarily choose a root, and arbitrarily prelabel the vertices with the integers from 1 to n. For each vertex concatenate the prelabel of its parent (if any) to its own prelabel. The new vertex labels have no more than $2 \log n$ bits. We can now quickly decide parenthood given two vertex labels by checking if the first half of one is the second half of the other. Thus we can decide adjacency. This also works for finite forests.

Transitive Closures of Trees: Often we are interested in the ancestorhood relation in trees rather than the adjacency relation. Let F be the family of transitive closures of rooted finite trees. The adjacency relation in F is the same as the ancestorhood relation in rooted finite trees. We can find a 2-labeling scheme for F: Let T be a tree and let T' be the transitive closure of T. The label we assign each vertex is an interval on the integer line. We traverse T in post-order. To each vertex we assign the interval between its smallest numbered descendant and its largest numbered descendant. A vertex u is an ancestor of v if and only if the interval for u contains the interval for v.

3.2 Sparse Graphs or Graphs with Bounded Arboricity

Let G be a graph and let H range over all possible vertex induced subgraphs of G. Then the *arboricity* of G is defined to be

$$\max_H \frac{|E(H)|}{|V(H)|-1}$$

where $|E(H)|$ is the number of edges in H and $|V(H)|$ is the number of vertices in H. Nash-Williams [11] showed that the edges of a graph G with arboricity k can be decomposed into k forests. This means that there is a $k+1$ labeling scheme for graphs of arboricity k. Prelabel the vertices arbitrarily with distinct integers from 1 to n. Concatenate to each vertex label, the label of its parent in each of the k forests. The ordered $k+1$ tuple is then the label of the vertex. To decide adjacency of two vertices just check if one is the parent of the other in any of the forests. It is clear that adjacency can be tested efficiently. Using flow techniques, Picard and Queyranne [13] have shown that the forest decomposition of a graph and thus its labels can be computed efficiently.

Several families fall into this category and hence have labeling schemes:

1. Graphs of bounded degree d clearly have arboricity bounded by $\lfloor d/2 \rfloor + 1$.

2. Graphs of bounded genus g fall in this category, since they have at most $6(g-1)+3n$ edges and hence can be labeled succinctly.

3. In the special case of planar graphs which have arboricity 3, this technique yields a 4-labeling scheme.

3.3 Intersection graphs

In this section we consider many families of graphs which are characterized as intersection graphs for certain types of sets. The general method used is to try to find a succinct representation for the sets representing the vertices.

Definition 6 *An interval graph is a graph where each vertex can be represented by an interval on the real line such that two vertices are adjacent iff the interval representing one vertex intersects the interval representing the other.*

The technique for labeling interval graphs is similar to the one for labeling transitive closures of trees. This yields a 2-labeling scheme. Adjacency can be tested in linear time in the length of the labels. In this case the definition of the sets gives us a labeling scheme. This is so for a number of families: *circle graphs, circular arc graphs, permutation graphs, graphs with bounded interval number*. See Golumbic for definitions of these families [8].
Path Graphs: Path graphs are graphs where each vertex is represented by a path in a tree. Two vertices are adjacent iff the paths representing them intersect. There are several kinds of path graphs mentioned in the literature. See Monma and Wei for details [10]. We will

show how to label path graphs where the paths are undirected and two paths intersect if they have a vertex in common. Similar methods will work for all path graph families, provided the path representation is given. To label path graphs, we first label the transitive closure of the tree containing all the paths. Now we can decide ancestorhood in the tree quickly.

To label the vertices of the path graph we concatenate the labels of the beginning, the apex, and the end of the path representing that vertex. Here the apex of a path is the vertex along the path closest to the root. To test adjacency of two vertices we have to test if the apex of one vertex is sandwiched between the apex and an end vertex of the other path. This requires at most six adjacency tests in a transitive closure of a tree. Finding the path representation and thus the labeling of a graph can be done efficiently [7]. It is interesting to note that the closely related family of *chordal graphs* which can be characterized as the intersection of subtrees in a tree cannot be labeled.

3.4 c-Decomposable Graphs

Definition 7 *A graph G is c-decomposable if for all subgraphs H with more than c vertices, there exist c vertices such that their removal causes H to be disconnected with no component containing more than $2|H|/3$ vertices [6].*

To label a c-decomposable graph G we construct a tree decomposition T, of G in the following manner. Let S be a c-separator for G. Then S is assigned to the root of T. If H_1 and H_2 are the two components obtained by removing S from G, the left subtree will be the tree for H_1 and the right subtree will be the tree for H_2.

Every vertex v of G occurs in a vertex $t(v)$ in T. T is binary, has depth no greater than $\log_{3/2} n$, and each vertex of T is assigned at most c vertices of G. Assume an arbitrary ordering among the vertices of the graph assigned to the same vertex of T. Two vertices in G can be adjacent only if they are assigned to two vertices of T such that one is an ancestor of the other.

To label G, for each vertex v, we remember the following:

1. The path in the tree from the root to $t(v)$ which is of length at most $\log_{3/2} n$,
2. The rank of v in $t(v)$ which is an arbitrary number from 1 to c
3. For each vertex of the tree, s, along the path from the root to $t(v)$, a c-bit vector giving adjacency information with all the vertices in G assigned to s. This is at most $c \cdot \log_{3/2} n$ bits.

Given two vertices u and v of G we determine if $t(u)$ is an ancestor of $t(v)$ (or vice versa). If so we determine the depth i of the ancestor, say u. The i^{th} c-bit vector in v's label has a bit at the position corresponding to u's rank which tells if u and v are adjacent. This is an efficient procedure to determine adjacency.

4 Constrained Labeling Schemes

In this section we prove that any family of graphs that has a labeling scheme also has a constrained labeling scheme as long as it is closed under vertex deletion. For example, let F be the family of graphs bounded by degree k. A constrained k-labeling scheme for F is obtained by associating with each vertex its own prelabel concatenated with the prelabels of its neighbors with smaller prelabels. We have already seen a constrained 2-labeling scheme for trees. Since graphs of bounded arboricity can be decomposed into a constant number of forests, we also have a constrained labeling scheme for these graphs. In fact, all the families of graphs for which we have given labeling schemes, also have constrained labeling schemes.

Theorem 1 *A family of finite graphs which is closed under vertex deletion, has a labeling scheme iff it has a constrained labeling scheme.*

Proof: We will call a family F of graphs *hereditary* if it is closed under vertex induced subgraphs, i.e. under vertex deletion. Let $G = (V, E)$ be a graph belonging to a family F, and let $S = (V', E')$ be some subgraph of G. An *extension* X of S is defined to be a subset of $V - V'$, such that $\forall x, y \in X$, the neighbor sets of x and y in G intersect V' in distinct sets.

For a family F, we define an *extension function*, f_F, as follows: $f_F(n)$ is the cardinality of the largest extension of any n-vertex, vertex induced subgraph of any graph $G \in$ F. With this definition the proof of the theorem reduces to the proof of the following two lemmas.

Lemma 1 *If F is hereditary, k_1-labelable and $f_F(n) \in O(n^{k_2})$, then F is constrained $4k_1k_2$ labelable. Note that the constant is independent of the prelabeling function and is entirely determined by graph-theoretic properties of the family, F.*

Proof: Let $G \in$ F, a k_1-labelable family. Let $p(v)$ denote the prelabel associated with vertex v. Let G_i denote the induced subgraph on the vertex set, $\{v : 2^{2^{i-1}} \leq p(v) < 2^{2^i}\}$. Define G_0 to be the singleton set consisting of the vertex with prelabel 1. The idea is that each vertex in G_j will remember its neighbors in G_i $\forall i \leq j$. In fact, the label for a vertex in G_j will be a $j+1$-tuple, the i^{th} component of which contains information about neighborhood with vertices in G_i, $i \in [0, 1, ..., j]$. Starting with 0, we consider the graphs G_i in turn. We label G_i together with a maximal extension E_i. The labels assigned here will be the i^{th} component of the vertex labels. For any vertex v there is a vertex $u \in E_i$ such that v and u have the same neighborhood in G_i since E_i is maximal. The i^{th} component of v's label will then be the same as the i^{th} component of u's label. We now delete the vertices of G_i and proceed to G_{i+1}.

It remains to prove that the labels assigned above satisfy the requirements of a constrained labeling scheme. The i^{th} component of a label comes from the labeling of G_i with its extension. If G_i has n vertices, then by our hypothesis on F, the maximal extension contains at most n^{k_2} vertices. Labeling a graph in F with $n + n^{k_2}$ vertices requires $k_1(k_2 + \epsilon) \log n$ bits, where $\epsilon \to 0$ as $n \to \infty$, provided $k_2 \geq 1$. By the definition of the G_i, n is less than 2^{2^i}.

Hence the i^{th} component of a label contains approximately $k_1 k_2 2^i$ bits. Hence a vertex in G_j has in all $\sum_{i=0}^{j} k_1 k_2 2^i$ bits. This sum is bounded by $k_1 k_2 2^{j+1}$. But the smallest prelabel in G_j is $2^{2^{j-1}}$ and the log of this value is 2^{j-1}. Hence the label size is bounded by $4k_1k_2$ times the size of the prelabel.

Finally, these labels can be used to determine adjacency efficiently. Given two labels, let the vertex with the smaller prelabel belong to G_i. Then adjacency can be tested by looking at the i^{th} components of both labels. ∎

Conversely the following lemma shows that if the extension function is superpolynomial, then the family of graphs does not even have a labeling scheme.

Lemma 2 *If F is hereditary and the extension function is super-polynomial, then F is not labelable.*

Proof: Let F be a family of graphs with super-polynomial extension. Let $f_F(n)$ be $n^{\alpha(n)}$ where $\alpha(n) \to \infty$ as $n \to \infty$. We have a sequence of graphs, $G_1, G_2, \ldots$ such that G_n contains a subgraph, H_n, with n vertices and H_n has an extension of cardinality at least $n^{\alpha(n)}$. We will prove that there are more than $2^{O(n \log n)}$ n vertex graphs in F, thereby proving that F cannot be labeled.

To H_n we can adjoin n vertices arbitrarily chosen from the extension. The induced subgraph on these $2n$ vertices is in F. We can choose the n vertices from the extension in $\binom{n^{\alpha(n)}}{n}$ different ways. This is approximately $n^{n\alpha(n)}$ choices of graphs. Some of the graphs we get may be isomorphic. However, each isomorphic copy can occur at most $\binom{2n}{n} n!$ times, since fixing the vertices of H_n determines the other vertices as well. Thus, in all we obtain approximately $n^{n\alpha(n)}/(2n)^n$ distinct graphs with $2n$ vertices. This number is $n^{\Omega(n\alpha(n))}$ which is $2^{\Omega(n \log n\alpha(n))}$. ∎

This concludes the proof of Theorem 1. ∎

As a corollary we have constrained labeling schemes for all families we have discussed in section 2. Theorem 1 has consequences for the labeling of infinite graphs:

Theorem 2 *Any infinite graph with the property that the family consisting of all its finite vertex induced subgraphs has a labeling scheme, has a constrained labeling scheme.*

Proof: Let G be an infinite graph and let F be the family of all finite, vertex induced subgraphs of G. Clearly F is hereditary. Therefore, by Theorem 1, F has a constrained labeling scheme. Consider the sequence S_0 of subgraphs of G, $G_1, G_2, \ldots$ where G_i is the induced subgraph on all vertices with prelabels $\leq i$. We have constrained labelings for all the G_i; any vertex has a finite prelabel, and therefore only finitely many possibilities for a label. Thus the vertex with prelabel 1 is labeled with the same label l_1 infinitely often in S_0. Define S_1 to be the infinite subsequence where vertex 1 gets label l_1. Vertex 1 is assigned the label l_1. Continuing in this manner, for the vertex prelabeled i, we can find a subsequence in S_{i-1} where it is labeled with some l_i infinitely often. Vertex i gets the label l_i and S_i is the corresponding subsequence of S_{i-1}. ∎

Corollary 1 *The infinite version of all the classes discussed above has a constrained labeling scheme.*

5 Labeling Schemes and Universal Graphs

Labeling schemes and vertex induced universal graphs are closely related.

Theorem 3 *If a family F has a k-labeling scheme, then it has universal graphs of size n^k constructible in polynomial time.*

Proof: Consider the polynomial time Turing machine T, to determine adjacency given two vertex labels of a graph in F. Form the graph U with vertices labeled from 1 to n^k, placing an edge between a and b iff T(a,b) says "adjacent". U must contain all graphs in F with n or fewer vertices, since all these graphs receive labels no greater than n^k.

Note that we can determine adjacency in U in time polynomial in the length of the vertex names. Hence, families with labeling schemes have efficiently constructible universal graphs. Furthermore the labels on the vertices of the universal graph make the embedding of a graph in the universal graph easy to find. To embed a graph G in its universal graph, all that is required is to label G. The labels then give the information about the embedding. As a non-trivial example, planar graphs have a 4-labeling scheme and hence have n^4 sized universal graphs. ∎

6 Related Work

Several authors have worked on related problems. Breuer [3] and Breuer and Folkman [4] considered the problem of labeling vertices such that adjacency would be determined by the Hamming distance of the labels. This is a restricted labeling scheme. They prove that this can be done for any graph; however, the length of the labels could be very large compared to $\log n$.

Turan [15] considered the problem of representing a graph as succinctly as possible. But the representation is global; it gives an efficient representation for the whole graph without necessarily partitioning the information into small chunks for each vertex. In this context it is clear that our labeling schemes always yield $O(n \log n)$-bit representations of all graphs which belong to labelable families. Furthermore from the representation of any graph it is easy to derive the representation of any of its vertex induced subgraphs.

Fredrickson and Janardan [6], Santoro and Khatib [14] and Peleg and Upfal [12] considered the question of storing routing information at the vertices of a packet switching network so as to compute near-optimal routes. Again, they look at the overall storage requirements rather than the requirements on a per vertex basis. This work has recently been extended to limit the amount of storage in every processor [1].

A number of papers have considered the question of universal graphs for a family of graphs. Most of these papers have focused on *edge induced universal graphs* which have often just been called *universal graphs*. In this paper we reserve the term 'universal graph' for vertex induced universal graphs and we refer to edge induced universal graphs explicitly. A graph G is edge induced universal for a set of graphs S if every graph in S is a subgraph of G.

The important issue in constructing edge induced universal graphs is minimizing the number of edges. Bhatt et al. [2] consider the question of constructing n vertex edge induced universal graphs for n vertex bounded degree trees and planar graphs. They show that for bounded degree trees there is an edge induced universal graph with cn edges where c is a constant depending only on the degree bound. For bounded degree planar graphs they show an edge induced universal graph with $O(n \log n)$ edges.

Recently Chung [5] has shown how to construct an edge induced universal graph for n vertex trees having n vertices and $O(n \log n)$ edges. For planar graphs [5] gives a bound of $O(n^3/2)$ on the number of edges in the edge induced universal graph. [5] also combines these results with the techniques of our paper to improve the bounds on the sizes of (vertex induced) universal graphs for trees and planar graphs. Specifically [5] proves that there is a graph with $O(n \log n)$ vertices and $O(n^2)$ edges that contains all n-vertex trees as induced subgraphs and that there is a graph with $O((n \log n)^3)$ vertices and $O(n^6)$ edges that contains all planar graphs on n vertices.

7 Open Problems

We know from Proposition 1 that any labelable family has at most $2^{O(n \log n)}$ n-vertex graphs. The natural question is whether all such families which are hereditary have a labeling scheme.

Although this paper produces 'small' universal graphs for various families, the question of matching upper and lower bounds for the sizes of universal graphs for these families still remains open.

All our results for labeling graphs are 'static'. It is often useful to consider dynamically changing graphs where vertices get added and deleted. Can labeling be done in such a manner as to permit quick updates of the labels?

Can the ideas for labeling graphs be extended to hypergraphs? If this could be done dynamically as well it would have applications to databases.

Acknowledgements

We would like to thank Manuel Blum, for his many suggestions to the paper. Russell Impagliazzo provided us with many insights. We received helpful remarks on our research and on the paper from Ronitt Rubinfeld, Raimund Seidel and Alejandro Schaffer. The third author would like to thank Ashok Chandra, Larry Carter, Don Coppersmith, and the other researchers at IBM Yorktown for crucial discussions in the embryonic stage of this research.

References

[1] B. Awerbuch, A. Bar-Noy, N. Linial and D. Peleg, *Compact distributed data structures for adaptive routing*, Proc. 21st ACM Symp. Theory of Computing, 1989, pp. 479–489.

[2] S. N. Bhatt, F.R.K. Chung, F.T. Leighton, and A. Rosenberg, *Universal graphs for bounded-degree trees and planar graphs*, SIAM J. Disc. Math. Vol. 2, No. 2, pp. 145–155, May 1989.

[3] M. Breuer, *Coding vertexes of a graph*, IEEE Trans. Info. Theory vol. 12, pp. 148-153, 1966.

[4] M. Breuer and J. Folkman, *An unexpected result on coding vertices of a graph*, J. Math. Anal. and Appl. vol. 20, pp. 583-600, 1967.

[5] F.R.K. Chung, *Universal graphs and induced universal graphs*, preprint.

[6] G. N. Fredrickson and R. Janardan, *Separator-based strategies for efficient message routing*, Proc. 27th IEEE Symp. on Foundation of Computer Science, pp. 428-437, 1986.

[7] F. Gavril, *A recognition algorithm for the intersection of paths in trees*, Disc. Math. vol. 23, pp. 211-227, 1978.

[8] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

[9] S. Kannan, M. Naor, and S. Rudich, *Implicit Representation of Graphs*, Proc. 20th ACM Symp. Theory of Computing, pp. 334-343, 1988.

[10] C. L. Monma and V. K. Wei, *Intersection graphs of paths in a tree*, J. Comb. Theory, Series B vol. 41, pp. 141-181, 1986.

[11] C. Nash-Williams, *Edge-disjoint spanning trees of finite graphs*, J. Lond. Math. So. vol. 36, pp. 445-450, 1961.

[12] D. Peleg and E. Upfal, *A tradeoff between space and efficiency for routing tables*, J. ACM 36, 1989, pp. 510-530.

[13] J. C. Picard and M. Queyranne, *Networks,* vol. 12, pp. 141-159, 1982.

[14] N. Santoro and R. Khatib, *Labeling and implicit routing in networks*, The Computer Journal vol. 28, pp. 5-8, 1985.

[15] G. Turan, *Succinct representation of graphs*, Disc. Appl. Math. vol. 8, pp. 289-294, 1984.