186
3-9-77

LA-6341-T
Thesis

UC-32
Dr 774
Issued: February 1977

# A Metric Graph Structure for Information Retrieval

by

Karl Jerry Melendez

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87545

An Affirmative Action/Equal Opportunity Employer

UNITED STATES
ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

This thesis was accepted by the University of New Mexico, Albuquerque, NM, Mathematics and Statistics Department in partial fulfillment of the requirements for the degree of Doctor of Philosophy. It is the independent work of the author and has not been edited by the Technical Information staff.

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $4.50 Microfiche $3.00

## TABLE OF CONTENTS

| Section | Page |
|---|---|
| INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . | 1 |
| THE DOCUMENT SPACE . . . . . . . . . . . . . . . . . . . | 2 |
| PROGRESSIVE GRAPHS . . . . . . . . . . . . . . . . . . . | 3 |
| PROGRESSIVE GRAPHS AND INFORMATION RETRIEVAL . . . . . | 10 |
| OPTIMAL PROGRESSIVE GRAPHS . . . . . . . . . . . . . . . | 18 |
| BUILDING PROGRESSIVE GRAPHS . . . . . . . . . . . . . . | 26 |
| ENUMERATION OF GRAPHS . . . . . . . . . . . . . . . . . | 37 |
| ORDER MATRICES . . . . . . . . . . . . . . . . . . . . . | 39 |
| CONCLUSIONS . . . . . . . . . . . . . . . . . . . . . . | 58 |
| APPENDIX 1 - Madcap 6 programs . . . . . . . . . . . . . | 59 |
| APPENDIX 2 - List of terms for sample problem. . . . . | 65 |
| ACKNOWLEDGMENTS. . . . . . . . . . . . . . . . . . . . . | 66 |
| REFERENCES . . . . . . . . . . . . . . . . . . . . . . . | 67 |

[illegible]

## LIST OF FIGURES

| figure | | page |
|---|---|---|
| 1 | Progressive path in the Euclidean plane . . . . . | 4 |
| 2 | Regressive path in the Euclidean plane . . . . . | 5 |
| 3 | The neighborhood N(r,X) in a progressive graph. . | 11 |
| 4 | Progressive graphs with a minimum number of arcs. . . . . . . . . . . . . . . . . . | 22 |
| 5 | Number of documents vs. average degree. . . . . . | 33 |
| 6 | Logarithmic fit of the average degree data. . . . | 34 |
| 7 | Number of documents vs. average DWPL. . . . . . . | 35 |
| 8 | Logarithmic fit of the average degree data. . . . | 36 |
| 9 | $(N+1)x(N+1)$ order matrix $0^{N+1}$ . . . . . . . . . . | 50 |
| 10 | $\hat{r}$ step cycle N+1,m,j,...,k,N+1,m . . . . . . . . | 52 |

## LIST OF TABLES

| table | | page |
|---|---|---|
| 1 | Results for unoptimized graph . . . . . . . . . . . | 30 |
| 2 | Results for optimized graph . . . . . . . . . . . . | 31 |
| 3 | Enumeration of labeled graphs . . . . . . . . . . . | 37 |
| 4 | 4x4 semi-canonical order matrices . . . . . . . . . | 55 |

## ABSTRACT

Document retrieval systems accept a user request for information and respond with a list of documents which contain information relevant to the request. When the documents (or abstracts of the documents) are stored in a computer memory, a function can be defined which estimates the semantic distance between documents. If this function together with the set of documents forms a metric space, a graph, which I call a progressive graph, can be constructed to aid the search for the documents with relevant information.

Progressive graphs are studied and the search algorithms which use this graph structure are presented. The search algorithms always perform correctly on any progressive graph, but the presence of the progressive property in a graph is not sufficient to insure that the algorithms will work efficiently. The characteristics of a progressive graph which will optimize the search algorithms are discussed and algorithms to build and optimize progressive graphs are given. The results of a small problem show that the search process using the graph created by these algorithms can be very efficient. Finally, the distance function property which determines when a graph is a progressive graph is isolated and studied.

# INTRODUCTION

When information is required from a large document library, the first problem is to formulate a query which describes the nature of the information desired. After the query is formulated, the next problem is to find the documents which contain relevant information. Furthermore, if the number of documents with relevant information is very large, the documents with the most relevant information must be identified. This suggests that a measure of the similarity between the query and the documents must be performed. In addition to measuring the similarity between the query and the documents it is sometimes useful to measure the similarity between documents.

One such measure is correlation. Correlation between documents increases as the documents become semantically alike. On the other hand, if the measure becomes smaller as the documents become semantically alike, then the measure can be considered a distance. When the distance function satisfies the properties of a metric, then the documents may be considered to be in some metric space.

Any query may be considered as a point in this metric space, and using the properties of the distance function the following questions can be answered.

(1) Which document is closest to the query?
(2) Which documents are within distance 'r' of the query?
(3) Which are the 'n' closest documents
    (given in order of increasing distance)?

## THE DOCUMENT SPACE

A general document space is a set of documents and a distance function or a correlation function. The function must measure the semantic similarity between documents. The distance between documents may be determined by manually scanning the information. However, unless the number of documents is very small, this task is much too time consuming.

Burd and Morrison investigated the usefulness of computing lexicographical correlation using the PATRICIA indexing algorithm[1]. Lexicographical correlation was computed between five documents of approximately the same length. Their results showed that the correlation between related documents was about twice the correlation between unrelated documents.

Correlation and distance can also be computed for document pairs by representing the documents as vectors and then computing the correlation or distance between the corresponding vector pairs. The vector representation of a document $D_i$ using t index terms is

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it})$$

where $d_{ij}$ represents the weight of the $j^{th}$ term in the $i^{th}$ document. The cosine of the angle between vector pairs can be used as a correlation measurement and the angle itself can be used as a distance function[7].

## PROGRESSIVE GRAPHS

When one knows how to measure the semantic closeness between two documents or between a document and a query, the questions posed in the introduction can be answered by comparing the query with each document in the library. This process may be very time consuming. One method of reducing the amount of work to answer these questions is to cluster the documents into groups of related documents[8, p. 323]. Then one needs only to compare the query with documents in clusters which may contain relevant documents. I am investigating an alternate method which requires that the distance function be a metric. A distance function is a metric if it satisfies the following conditions for arbitrary documents X, Y and Z.

(1) $d(X,Y) \geq 0$ and $d(X,Y)=0$ iff $X=Y$

(2) $d(X,Y) = d(Y,X)$

(3) $d(X,Y) \leq d(X,Z)+d(Z,Y)$

A distance function can be used to construct a graph which will help in the search for relevant documents. The points in the metric space correspond to descriptions of documents or queries, while the nodes in the graph correspond to descriptions of documents which have been catalogued.

The following definitions classify finite graphs whose nodes are points in a metric space. A finite graph (S,L) is a finite set of nodes S and a set L of arcs between nodes in S. If the nodes in S are points in a metric space with a distance function d, then (S,d) will be a finite metric space. An arc in L between two nodes X and Y will be denoted by the unordered pair [X,Y]. In this case we say that X is adjacent in L to Y and Y is adjacent in L to X. (S',L') is a subgraph of (S,L) if S' is a subset of S and L' is a subset of L. For any subset S' of S the induced subgraph <S'> of (S,L) is the subgraph (S',L') where L' contains all arcs in L between points in S'. A path from X to Y in (S,L) is a sequence of nodes $X=X_1,X_2,\ldots,X_n=Y$ with the property that $X_i$ is adjacent to $X_{i+1}$ for $i=1,\ldots,n-1$. The following two special kinds of paths depend on both the graph and the distance function.

DEFINITION:(Progressive Path) A path $X=X_0,X_1,\ldots,X_n=Y$ in a graph (S,L) is progressive with respect to the distance function d if and only if $i<j$ implies that

$$d(X_i,Y) > d(X_j,Y).$$

Traveling along a progressive path, the distance to the last node in the path is getting progressively smaller.

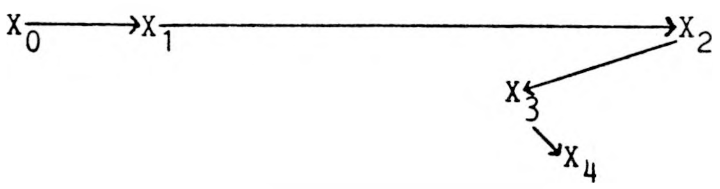

Figure 1.
Progressive path in the Euclidean plane.

DEFINITION: (Regressive Path) A path $X=X_0,X_1,\ldots,X_n=Y$ in a graph (S,L) is regressive with respect to the distance function d if and only if i<j implies that
$$d(X_i,X) < d(X_j,X).$$

Traveling along a regressive path, the distance from the first node in the path is getting larger.

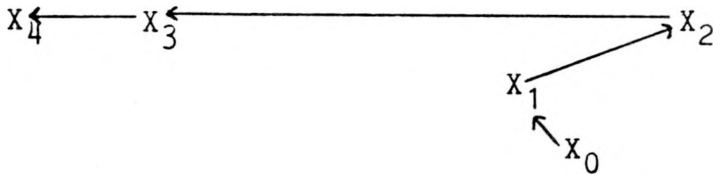

Figure 2.
Regressive path in the Euclidean plane.

Progressive and regressive graphs can now be defined as follows.

DEFINITION: (Progressive Graph) A graph (S,L) is progressive with respect to the distance function d if and only if for every pair of nodes X and Y in S there exists a progressive path from X to Y.

DEFINITION: (Regressive Graph) A graph (S,L) is regressive with respect to the distance function d if and only if for every pair of nodes X and Y in S there exists a regressive path from X to Y.

The following lemma shows the relationship between a progressive path and a regresssive path.

LEMMA 1: A path $X_0,X_1,\ldots,X_n$ is progressive if and only if the path $X_n,X_{n-1},\ldots,X_0$ is regressive.

PROOF: In any regressive path, the distance from the first node in the path must increase as the number of steps from the first node increases. The path $X_n,X_{n-1},\ldots,X_0$ has indices which decrease as the number of steps from $X_n$ increases. It follows that

$X_n, X_{n-1}, \ldots, X_0$ is regressive

if and only if

$j > i$ implies that $d(S_j, X_n) < d(X_i, X_n)$

if and only if

$i < j$ implies that $d(X_i, X_n) > d(X_j, X_n)$

if and only if

$X_0, X_1, \ldots, X_n$ is progressive

A path is progressive or regressive depending on which direction you are traveling. The following theorem says that the progressive and regressive properties for graphs are equivalent.

THEOREM 1: A graph (S,L) is progressive with respect to the distance function d if and only if (S,L) is regressive with respect to the distance function d.

PROOF: Let X and Y be arbitrary nodes in (S,L). By lemma 1 we know that a path from X to Y is progressive if and only if the reverse path from Y to X is regressive.

The complete graph is a graph in which every node is adjacent to every other node. It is progressive with respect to any distance function, because a one step path exists between every pair of nodes. Every progressive graph must be a subgraph of the complete graph. Theorem 2 shows that arcs between closest neighbors must always be in every progressive graph.

<u>THEOREM</u> 2: Let the graph (S,L) be progressive with respect to the distance function d. Let X be an arbitrary node in S. If Y is a node in S such that $X \neq Y$ and $d(X,Y) \leq d(X,Z)$ for all nodes $Z \neq X$, then Y is adjacent to X.

PROOF: Let X and Y be arbitrary distinct nodes in S with the property that $d(X,Y) \leq d(X,Z)$ for all $Z \neq X$. Since (S,L) is progressive with respect to d, (S,L) must also be regressive with respect to d. Therefore, there exists a regressive path $X=X_0,\ldots,X_n=Y$ from X to Y. If $n > 1$, then $d(X,X_1) < d(X,Y)$. This contradicts the assumption that $d(X,Y) \leq d(X,Z)$ for all $Z \neq X$. Therefore, the regressive path from X to Y is $X=X_0,X_1=Y$. This shows that X is adjacent to Y.

Theorem 2 tells us which arcs must be in every progressive graph, but the existence of these arcs is not sufficient to show that a graph is progressive. The following theorem gives a criterion by which one can determine if a graph is progressive.

<u>THEOREM</u> 3: (First Step Rule) A graph (S,L) is progressive (and hence regressive) with respect to a distance function d if and only if for every pair of nodes X and Y in (S,L) the following holds:

(I) There exists Z in (S,L) such that Z is adjacent to X and is closer to Y than X is to Y (i.e., $d(Z,Y)<d(X,Y)$ ).

PROOF: Let (S,L) be a progressive graph with respect to the distance function d. Let X and Y be arbitrary nodes in (S,L). Since (S,L) is progressive, there exists a progressive path $X=X_0,X_1,\ldots,X_n=Y$. By the definition of progresive path, $X_1$ is adjacent to X and $d(X_1,Y)<d(X,Y)$. Hence, condition (I) holds.

Conversely, assume that condition (I) holds. Let X and Y be arbitrary nodes in (S,L). Condition (I) says that there exists a node $X_1$ such that $d(X_1,Y) < d(X,Y)$. If $X_1 = Y$, then $X=X_0,X_1=Y$ is a one step path from X to Y. In general, let $X=X_0,\ldots,X_k$ be a path such that $i>j$ implies that $d(X_i,Y) < d(X_j,Y)$. For $k=1$, I have shown that such a path exists. If $X_k=Y$, then $X=X_0,\ldots,X_k=Y$ is a progressive path from X to Y. If $X_k \neq Y$, then there exists $X_{k+1}$ such that $d(X_{k+1},Y) < d(X_k,Y)$. Thus, $X=X_0,\ldots,X_k,X_{k+1}$ is a path of length k+1 with the same property. If the terminal point of the path is Y, it is a progressive path from X to Y. Otherwise, the path length can be increased by one. Since (S,L) has a finite number of nodes, this process must end with a progressive path from X to Y. Therefore, the graph (S,L) is progressive with respect to the distance function d.

The following example shows a case where the set of arcs between the closest neighbors is sufficient to make the graph progressive.

Example 1:

$$S = \{1,2,3,4\}$$

$$L = \{ [1,2],[2,3],[3,4] \}$$

The distance function is the normal distance function on the integers.

$$d(X,Y) = |X-Y|$$

## PROGRESSIVE GRAPHS AND INFORMATION RETRIEVAL

Each document in the library must have a description with sufficient information to calculate the distance between two documents or between a document and a query. The points in the metric space correspond to these descriptions. The nodes in the graph correspond to the entries in the library catalogue. Therefore, in addition to a list of adjacent nodes, each node in the graph must contain a description of the document and any other information a catalogue must contain. Since the node contains all the necessary information, the following two functions will be trivial.

des(location) - Given the location of a node in the graph, return the description of the document.

adj(location) - Given the location of a node in the graph, return the set of all node locations which are adjacent to this node.

The properties of a progressive graph can be used to find the location of the node corresponding to a document description. If (S,L) is a progressive graph with respect to the distance function d, then Theorem 2 says that for any two nodes X and Y in S there exists a node adjacent to X which is closer to Y than X is to Y. The closer node may be Y itself. Thus, to find a node Y in (S,L) one needs only to start at any arbitrary node X in (S,L) and examine all nodes adjacent to X. One of these nodes must be Y or must be closer to Y than X is to Y. This suggests the following algorithm to find any node in (S,L).

ALGORITHM 1: (Given a description of a document Y, find the location of the node corresponding to this document.)

```
function loc(Y)
description Y
location X
X := location of any node in (S,L)
while des(X) ≠ Y do
   X:= location of the node which is closest to Y
       among all nodes adjacent to X
loc := X
return
end
```

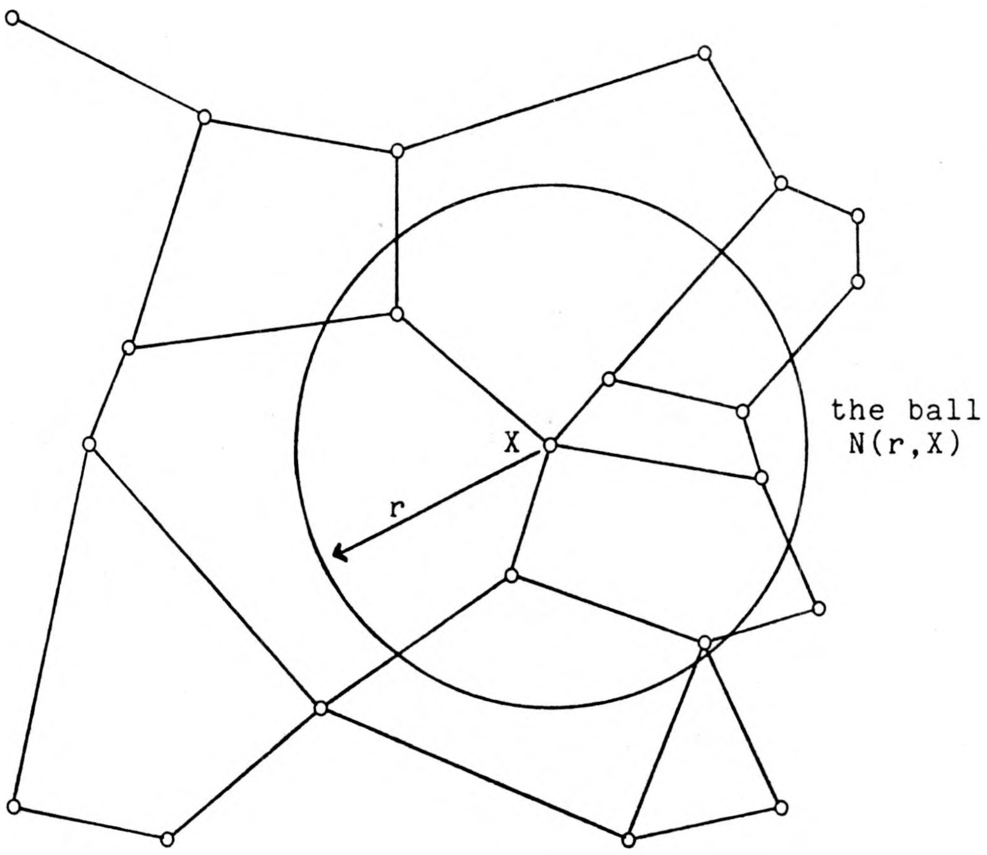

Figure 3.
The neighborhood N(r,X) in a progressive graph.

Algorithm 1 must terminate because each iteration is a step in a progressive path, and there are only a finite number of documents. At each step in this path, the adjacent node closest to the terminal node is chosen as the next step. The work performed to find a location is proportional to the sum of the degrees of the nodes in the path.

The next problem is to find all nodes which are within a fixed distance 'r' of X. See Figure 3. This set of nodes will be denoted by N(r,X). In the metric space (S,d), N(r,X) is a ball of radius 'r' with center at X. The following theorem shows that the induced subgraph <N(r,X)> is not only connected, but that regressive paths exist from its center to every other point.

THEOREM 4: If (S,L) is progressively connected with respect to the distance function d, then the induced subgraph <N(r,X)> contains a regressive path from X to every other point in N(r,X).

PROOF: For every node Y in N(r,X) there exists a regressive path $X=X_0,X_1,\dots,X_n=Y$ in (S,L) from X to Y. By the definition of regressive path, $d(X.X_i)<d(X,Y)\leq r$. Thus, each node $X_i$ is in the neighborhood N(r,X) and the path is in the subgraph <N(r,X)>.

To find all the nodes in N(r,X) it is sufficient to use the subgraph <N(r,X)>. Let Y be the nth closest point to X in N(r,X). There exists a regressive path in <N(r.X)> from X to Y. Therefore, by Lemma 1 the reverse of this path is a progressive path from Y to X. Since a progressive path exists from Y to X, Y must be adjacent to X or adjacent to one of the n-1 closer nodes. Thus, having found the n-1 closest nodes, one need only search through nodes which are adjacent to X or the n-1 closest. This is the same property that assures the correctness of algorithms to find the shortest path through a network[2, 3]. Since Algorithm 2 finds all the documents in the neighborhood N(r,X) in order of increasing distance from document X, it can easily be modified to find the closest 'n' documents.

ALGORITHM 2: (Given r and X, find the documents in N(r,X) in order of increasing distance from X.)

```
function NRX(r,X)
sequence NRX
nodes X,nth,a
number r
sets Frontier,Periphery,A

NRX := <> "Initalize NRX to a null sequence."
Frontier := { a in adj(X) : d(a,X) ≤ r}
Periphery := { a in adj(X) : d(a,X) > r}
until Frontier = {} do
   nth := (closest node in Frontier)
   NRX := NRX,nth "Append nth to NRX."
   A := adj(nth) - (NRX .union. Frontier .union. Periphery)
   Periphery := Periphery .union. {a in A : d(a,X) > r}
   Frontier := Frontier .union. {a in A : d(a,X) ≤ r}
   Frontier := Frontier - {nth}
return
end
```

After the closest k nodes have been found, the (k+1)st closest node must be in the Frontier. The (k+1)st closest node can be found by a simple search of the nodes in the

Frontier. This search can be eliminated by keeping the Frontier as a list in increasing order of distance from X. An alternative algorithm is to simply find all nodes in N(r,X) which are one step away from X, then all nodes which are two steps away from X, etc. When the complete neighborhood has been found, it can then be sorted in increasing distance from X. In any case the number of comparisons required to sort the nodes in increasing order of their distance from X depends only on the number of documents in the neighborhood.

When a node is added to the neighborhood, any of its adjacent nodes which have been reached before can be discarded. A node has been reached before if it is already in the neighborhood N(r,X), the Frontier, or the Periphery. If each node is marked when it is placed in any of these lists, the lists will not need to be searched to determine when a node has been reached before. At the end of the algorithm, all marks must be removed.

Algorithm 1 finds only nodes which are in the graph. A query will not be a node in the graph, but the properties of a progressive graph can be used to find the document node which is closest to a query. If a node X in (S,L) has no adjacent nodes which are closer to the query q, then X is locally closest to q. If there exist nodes in (S,L) which are closer to q they will be members of the neighborhood

N(2*d(X,q),X). Suppose X is locally closest, but is not the closest node to q. Then there exists a node Y such that d(Y,q) < d(X,q). Using the triangle inequality

$$\begin{aligned} d(X,Y) &\le d(X,q) + d(q,Y) \\ &= d(X,q) + d(Y,q) \\ &< d(X,q) + d(X,q) = 2*d(X,q) \end{aligned}$$

If d(X,q) is large, the neighborhood N(2*d(X,q),X) may contain many nodes. Therefore, if X has no adjacent nodes which are closer to q it is sufficient to find any node in N(2*d(X,q),X) which is closer to q. In this case Algorithm 3 finds all nodes in N(2*d(X,q),X) which are one step away from X, then all nodes which are two steps away from X and so on until a closer node is found or until all nodes in N(2*d(X,q),X) have been found. Only arcs in the induced subgraph <N(2*d(X,q),X)> need to be examined, because <N(2*d(X,q),X)> has progressive paths from X to every node and is therefore connected.

Algorithm 3 is a combination of algorithms 1 and 2. Algorithm 1 is used until a locally closest node is found. Then, a modification of Algorithm 2 is used to verify that the current location is the closest or to find a closer location. If a closer location is found, Algorithm 1 is again used to step closer to the query.

ALGORITHM 3: (Algorithm to find the document closest to a query.)

```
function closest(q)
description q
sets A,OF,NF,NRX
locations closest,f,x,x0,a,n
numbers r
boolean closer
closer := true
x := any location in S
while closer = true do
   "Check all locations adjacent to x
    for locations closer to the query q."
   "If any exist, choose the location closest to q."
   x0 := x
   for all a in adj(x) do
      if d(des(a),q) < d(des(x),q) then x := a
   if x = x0 then
      "If no points adjacent to x are closer to q,
       check a neighborhood of radius 2*d(x,q) for
       closer documents."
      r := d(x,q)
      NRX := {x}
      OF := {x}
      until OF={} or x≠x0 do
         NF := {}
         for all f in OF while x=x0 do
            A := adj(f) - NRX
            NRX := NRX .union. A
            for all a in A do
               if d(des(a),q) < r then x:=a
               if d(des(a),des(x)) ≤ 2*r then NF := NF.union.{a}
         OF := NF
   if x=x0 then closer := false
closest := x
return
end
```

By finding successively larger neighborhoods of the closest document, the following algorithm will find the 'n' closest. If Y is the closest document to the query q and Z is the kth closest document to q, then the neighborhood N(d(Z,q)+d(Y,q),Y) will contain all the k-1 closest documents. This follows from the fact that the neighborhood N(d(Z,q),q) is contained in N(d(Z,q)+d(Y,q),Y).

ALGORITHM 4: (Algorithm to find the 'n' closest documents to a query.)

```
 function findn(n,q)
 "At stage k, all the k closest nodes have been found."
    "findn - A list of the k closest nodes <N(1),....,N(k)>"
    "NBR - Nodes in the neighborhood N(r,N(1)) but
            not in findn.  r= d(N(k),N(1)) + d(N(1),q)"
    "PER - Nodes adjacent to a node in NBR or findn which
           are not already members of NBR or findn."
 sequence findn
 description q
 numbers n,dqy,count,total
 locations y,z,p,a
 sets NBR,OF,NF,PER,IN,NEW

 findn := <>
 if n ≤ 0 return
 y := closest(q)
 dqy := d(des(y),q)
 findn := <y>

 count := 1
 total := n
 PER := { p in adj(y) }
 until PER={} or count>total do
    NBR := { element in PER which is closest to y }
    until NBR={} or count>total do
       z := (element in NBR which is closest to y)
       r := d(des(z),q) + dqy
       OF := {z} .union. {p in PER : d(des(p),des(y)) ≤ r}
       PER := PER - OF
       until OF={} do
          NF := {}
          NBR := NBR .union. OF
          for all f in OF do
             A:={a in adj(f)}-{elements in NF,NBR,PER, or findn}
             IN := {a in A : d(des(a).des(y)) ≤ r }
             NF := NF .union. IN
             PER := PER .union. (A-IN)
          OF := NF
       "At this point NBR must contain all nodes which are not
        already in findn but are closer to q than z is."
       "If d(des(a),q) < d(des(z),q) , then
         d(des(a),des(y)) ≤ d(des(a).q) + d(des(y).q)
                          ≤ d(des(z).q) + dqy = r."
       NEW := {a in NBR : d(des(a).q) ≤ d(des(z),q) }
       NBR := NBR - NEW
       "Sort the elements in NEW and append them to findn."
       findn := append(findn,<sorted elements of NEW>)
       count := count + (number of elements in NEW)
 return
 end
```

## OPTIMAL PROGRESSIVE GRAPHS

The complete graph is a graph in which every node is adjacent to every other node. Such a graph is clearly progressive since a one step progressive path exists between any two nodes. However, a simple sequential search is more efficient than using algorithms 1 through 4 on the complete graph.

Given any two distinct nodes X and Y in a complete graph with n nodes, Algorithm 1 will make n-1 distance calculations to find Y starting at node X. Since the graph is complete, X has degree n-1 and Algorithm 1 will sequentially search all n-1 adjacent nodes for the closest node to Y. Since Y is adjacent to X, Algorithm 1 will find Y during this search. A sequential search of a file would require at most n distance calculations and the average number of distance calculations a sequential search makes is n/2.

The difficulty Algorithm 1 encountered with the complete graph was that the degree of each node was very large. One may come to the conclusion that all Algorithm 1 requires is a progressive graph with the smallest number of arcs possible. However, consider the case of a path graph. A path graph (P,L) is a set of nodes $\{X_i : i=1,2,\ldots,n\}$ and a set of arcs $\{[X_i,X_{i+1}] : i=1,2,\ldots,n-1\}$. Progressive graphs must first be connected, and any connected graph with n nodes must have at least n-1 arcs[9]. Therefore, no progressively connected graph can have fewer arcs than a path graph. If the path graph (P,L) is progressively connected (see example 1) then Algorithm 1 will make 2k-3 distance calculations to find $X_k$ starting at $X_1$. There are k-1 steps and every step requires two distance calculations except the first step, which requires only one distance calculation. The average number of distance calculations to find $X_k$ , k=2,3,...,n will be

$$[\ 1 + \sum_{k=1}^{n} (2k-3)\ ] \ / \ (n-1) = (n-1).$$

Thus, although the path graph has the smallest number of arcs possible for a connected graph, the average number of distance calculations required by Algorithm 1 is proportional to the total number of documents in the graph.

This indicates that both high node degrees and long search paths will cause Algorithm 1 to be inefficient. The following definition incorporates both these measurements.

DEFINITION: (degree weighted path length) The degree weighted path length of a path $X=X_0,\ldots,X_n=Y$ is

$$\sum_{i=0}^{n-1} \text{degree}(X_i).$$

Starting at any node X in a progressively connected graph, Algorithm 1 will follow a unique path searching for the node Y. At each step $X_i$ in the path Algorithm 1 will make degree($X_i$) distance calculations to determine the next step. Thus, the degree weighted path length of this path from X to Y is the total number of distance calculations Algorithm 1 makes to find Y starting at X. The degree weighted path length of this unique path from X to Y will be denoted by

DWPL(X,Y). I have already shown that for a complete graph DWPL(X,Y) = n-1 if $X \neq Y$ and for the path graph $DWPL(X_1,X_k) = 2k-3$.

Algorithm 1 is a search algorithm which will start at a location X and find the location of a node Y which may be unrelated to X. On the other hand, Algorithm 2 starts at a node location X and finds all locations within a given distance from X, i.e., all locations in the neighborhood N(r,X). To find a neighborhood N(r,X) Algorithm 2 will make at most

$$\sum_{N(r,X)} \text{degree}(Y)$$

comparisons of the distance to X and the radius of the neighborhood 'r'. If the graph is the complete graph, then Algorithm 2 must determine if $d(X,Y) \leq r$ for all Y adjacent to X. This requires degree(X) = n-1 distance calculations where n is the total number of nodes in the graph. However, if X is a node in a path graph, Algorithm 2 will make at most

$$\sum_{N(r,X)} \text{degree}(Y) \leq 2 * \#N(r,X)$$

distance calculations, where #N(r,X) is the number of nodes in N(r,X). The number of distance calculations in the complete graph is proportional to the total number of nodes in the graph while the number of distance calculations in a path graph is proportional to the number of nodes in the neighborhood N(r,X). Therefore, the complete graph is undesirable for both algorithms 1 and 2, and a path graph is undesirable only for Algorithm 1. In general it is clear that Algorithm 2 requires progressive graphs whose nodes have low degree.

Let S' be the set of all graphs on the set of nodes S which are progressive with respect to a distance function d. A partial order can be defined on S' by:

$$(S,L_1) \leq (S,L_2)$$
$$\text{if and only if}$$
$$L_1 \text{ is contained in } L_2$$

The minimal elements in this partial order are the progressive graphs of interest. If (S,L) is a minimal element, then the removal of any line will cause the graph to lose the progressive property. Minimal progressive graphs can be found by removing unnessary lines from a graph which is already progressive. Theorem 3 gives a criterion to determine when an arc is unnecessary. Theorem 3 says that an arc [X,Y] is unnecessary if its removal does not cause X or Y to fail the first step requirement. A much harder problem is to find a minimal progressive graph with the property that no other progressive graph has fewer arcs. Such a graph must exist, but the following example shows that it is not unique.

Example 2:

$S = \{ X_1=(0,0) , X_2=(0,4) , X_3=(3,8) , X_4=(3,0) \}$

$L_1 = \{ [X_1,X_2] , [X_1,X_4] , [X_2,X_3] , [X_2,X_4] \}$

$L_2 = \{ [X_1,X_2] , [X_1,X_4] , [X_2,X_3] , [X_3,X_4] \}$

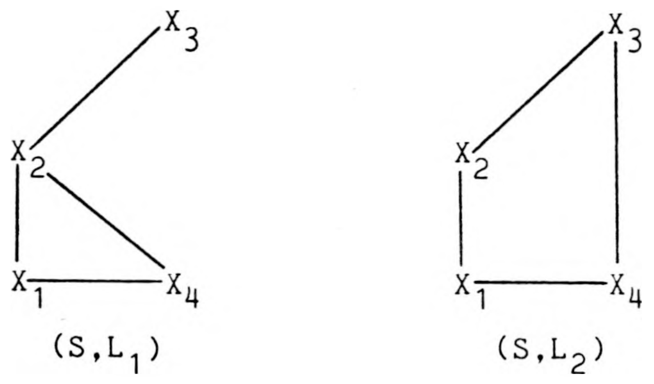

Figure 4.
Progressive graphs with a minimum number of arcs.

Both $L_1$ and $L_2$ have the same number of arcs and are progressive with respect to ordinary Euclidean distance. Furthermore, no progressive graph can have fewer than 4 arcs on this set of nodes.

Since graphs in S' with the smallest number of arcs are not unique, it follows that the intersection of two graphs in S' may not be a graph in S'. However, by Theorem 2 it follows that the intersection of two graphs in S' cannot result in a graph with no arcs. The following set of arcs must be in every progressive graph.

$$\left\{ [X,Y] \text{ in } S \;\middle|\; \begin{array}{l} X \text{ is a closest node to } Y \\ \text{or} \\ Y \text{ is a closest node to } X \end{array} \right\}$$

When two nodes A and B are equidistant to a third node X and no other nodes are closer to X, then A and B are closest to X. Any progressive graph must contain both arcs [A,X] and [B,X]. However, if the distance function does not measure A and B as exactly equidistant to X, then only the closest node may need to be adjacent to X.

A distance function d will be called an isosceles distance function on the set S if there exist three points X, Y and Z in S such that $d(X,Y) = d(X,Z)$. Otherwise, the function will be called non-isosceles. The triangle with vertices at X, Y and Z will be an isosceles triangle if distance is measured with an isosceles distance function.

If d is an isosceles distance function on the set S and $a:S \dashrightarrow \{1,2,\ldots,n\}$ is a one to one function from S onto $\{1,2,\ldots,n\}$, then a non-isosceles distance function $d_a$ can be defined by

$$d_a(X,Y) = \begin{cases} 0 & \text{if } X=Y \\ d(X,Y) + e*[a(X)+a(Y)] & \text{if } X\neq Y \end{cases}$$

where

$$e = \frac{\min \{|d(X,Y)-d(X,Z)| \ : \ d(X,Y)\neq d(X,Z)\}}{2n} \ .$$

To show that the triangle inequality holds, assume that X, Y and Z are arbitrary points in S. Then,

$$\begin{aligned}
d_a(X,Z) &= d(X,Z) + e^*[a(X)+a(Z)] \\
&\leq d(X,Y) + d(Y,Z) + e^*[a(X)+a(Z)] \\
&< d(X,Y) + d(Y,Z) + e^*[a(X)+2a(Y)+a(Z)] \\
&= d(X,Y) + e^*[a(X)+a(Y)] + d(Y,Z) + e^*[a(Y)+a(Z)] \\
&= d_a(X,Y) + d_a(Y,Z)
\end{aligned}$$

The following theorem shows that $d_a$ is a non-isosceles distance function on S.

THEOREM 5: For any three points X, Y and Z in S,

i) $d(X,Y)<d(X,Z)$ implies that $d_a(X,Y) < d_a(X,Z)$

ii) $d(X,Y)=d(X,Z)$ implies that

$$d_a(X,Y)<d_a(X,Z) \text{ if and only if } a(Y)<a(Z)$$

PROOF:

Assume that $d(X,Y) < d(X,Z)$.

$$\begin{aligned}
d_a(X,Y) &= d(X,Y) + e^*[a(X)+a(Y)] \\
&< d(X,Y) + 2ne \\
&= d(X,Y) + 2n \frac{\min\{|d(r,s)-d(r,t)| : d(r,s)\neq d(r,t)\}}{2n} \\
&\leq d(X,Y) + 2n \frac{|d(X,Z)-d(X,Y)|}{2n} \\
&= d(X,Y) + |d(X,Z)-d(X,Y)| \\
&= d(X,Y) + d(X,Z)-d(X,Y) \quad \text{since } d(X,Y)<d(X,Z) \\
&= d(X,Z) \\
&< d(X,Z) + e^*[a(X)+a(Z)] \\
&= d_a(X,Z)
\end{aligned}$$

This proves i). Now assume $d(X,Y)=d(X,Z)$.

$$\begin{aligned} d_a(X,Y) &= d(X,Y) + e*[a(X)+a(Y)] \\ &= d(X,Z) + e*[a(X)+a(Y)] \quad \text{since } d(X,Y)=d(X,Z) \\ &= d(X,Z) + e*[a(X)+a(Z)+a(Y)-a(Z)] \\ &= d(X,Z) + e*[a(x)+a(Z)] + e*[a(Y)-a(z)] \\ &= d_a(X,Z) + e*[a(Y)-a(Z)] \end{aligned}$$

This proves ii).

This theorem shows that $d_a$ uses the function 'a' to resolve any ties that d may encounter when measuring distances from a common point. Thus, any progressive path with respect to d will be a progressive path with respect to $d_a$.

<u>COROLLARY</u>: Let $a:S \dashrightarrow \{1,2,\ldots,n\}$ be a one to one function from S onto $\{1,2,\ldots,n\}$. If (S,L) is a progressive graph with respect to a distance function d, then (S,L) is a progressive graph with respect to $d_a$.

In the study of progressive graphs it is convenient to use a non-isosceles distance function, as it avoids troublesome special cases caused by pairs of points equidistant from a third point. In the remainder of this paper I shall use only non-isosceles distance functions.

## BUILDING PROGRESSIVE GRAPHS

The previous algorithms illustrate how progressive graphs can be useful in document retrieval. Now I turn my attention to the problem of building a progressive graph. A progressive graph can be built by initializing the graph to contain one document and no arcs. Then, the rest of the documents can be added one at a time with a sufficient number of arcs to insure that the graph will remain progressive. Theorem 3 tells us that a graph is progressive if and only if there exists a first step in a progressive path from each node to every other node. Therefore, when a new node X is added to a progressive graph, it is sufficient to insure that for every old node Y there exists a first step in a progressive path from Y to X and from X to Y. Recall that a first step in a progressive path from X to Y exists if there is a node Z adjacent to X such that $d(Z,Y)<d(X,Y)$.

When a new node X is added to a progressive graph, it will initially be isolated from the rest of the graph, i.e., it will not have any adjacent nodes. Therefore, it is best to start by insuring that there exists a first step from each old node to the new node. If a first step does not exist for some node Y, then one can insure a first step by adding the arc [Y,X] to the graph. If $Y_0$ is the closest old node to X, then $Y_0$ cannot have an adjacent node which is closer to X. The arc $[Y_0,X]$ must therefore be added to the graph. Thus, after a first step is assured from all the old nodes to the new node X, then X will no longer be isolated.

Next, one must insure that a first step exists from the new node X to every old node. If a first step does not exist for some node Y, then a first step can be insured by adding the arc [X,Y] to the graph.

ALGORITHM 5: (Add a document to a progressive graph.)

```
subroutine add1(x)
 "add1(x) adds one more node to a progressive graph
  to create a new progressive graph."

description x,y,z
location locx,locy,locz
global Graph
set Frontier

"Create a new node for the new document."
"Save the location of the new node in locx."
locx := create(x)

"Check for progressive paths to locx."
for locy in Graph do
   y := des(locy)
   if [ for all locz in adj(locy) : d(y,x)≤d(des(locz),x)] do
      call tie(locx,locy)  "form the arc [locx,locy]"

for locy in Graph do
   "Check for a first step in a progressive."
   "Path from locx to locy."

   y := des(locy)
   if [ for all locz in adj(locx) : d(x,y)≤d(des(locz),y)] do
      call tie(locy,locx) "form the arc [locy,locx]"

return
end
```

When a progressive graph is built by adding documents one at a time, the final graph will depend on the sequence in which the documents are added to the graph. The following algorithm will use the first step rule to eliminate unnecessary lines from the graph.

ALGORITHM 6: (Optimization)

```
function removable(x,y)
boolean removable,stepxz,stepyz
locations x,y,z,a
global Graph

removable := true
"Check for a first step from x to every other location."
for z in Graph while removable=true do
   stepxz := false
   if z=x then stepxz := true
   else "Check for a first step which is not y."
      for a in adj(x) while stepxz=false do
         if a≠y and d(a,z)<d(x,z) then stepxz := true
      removable := stepxz

"Check for a first step from y to every other location."
for z in Graph while removable=true do
   stepyz := false
   if z=y then stepyz := true
   else "Check for a first step which is not x ."
      if a≠x and d(a,z)<d(y,z) then stepyz := true
   removable := stepyz
return
end

subroutine optimize
"acc(x) is a function which will return
   the accession number of the location z."
"untie(x,y) is a subroutine which will remove
   x from adj(y) and y from adj(x)."
locations x,y
sets X,Y

X := {all locations}

for all x in X do
   Y := {y in adj(x)}
   for all y in Y such that acc(x)>acc(y) do
      if removable(x,y) then call untie(x,y)
return
end
```

Subroutine 'optimize' uses the function 'removable' to determine which arcs can be removed without causing the graph to lose the progressive property. When an arc can be removed, subroutine 'untie' is used to remove the arc from the graph. After all arcs which can be removed are eliminated, the graph will be a minimally progressive graph.

The function 'removable' uses the first step rule to determine when an arc can be removed without the loss of the progressive property. An arc [X,Y] can be removed if for every node in the graph, there exists a first step from X which is not Y and a first step from Y which is not X.

These algorithms were applied to a small problem with 200 documents. The documents were program abstracts from the program library maintained by the Computer Science and Services Division of the Los Alamos Scientific Laboratory. Each document $D_i$ was represented by a 266-dimensional vector

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{i266})$$

where $d_{ij}=1$ if term j occurred in document i, otherwise $d_{ij}=0$. See Appendix 2 for a list of the terms. The angle between the document vectors was used as the distance measure[7]. The graph was initalized by setting $S=\{D_1\}$ and $L=\{\}$. Algorithm 5 was used to add the rest of the documents to the graph one at a time. At each multiple of 10, the following values were computed.

$$\text{average degree} = (1/n) \sum_{i=1}^{n} \text{degree}(D_i)$$

$$\text{average DWPL} = (1/n^2) \sum_{i=1}^{n} \sum_{j=1}^{n} \text{DWPL}(D_i, D_j)$$

$$\text{maximum DWPL} = \max \{ \text{DWPL}(D_i, D_j) : i,j = 1,2,\ldots,n \}$$

Table 1 summarizes the results.

| number of documents | average degree | average DWPL | maximum DWPL |
|---|---|---|---|
| 10 | 2.60 | 5.74 | 13 |
| 20 | 3.90 | 10.21 | 24 |
| 30 | 5.27 | 13.50 | 33 |
| 40 | 5.30 | 15.99 | 40 |
| 50 | 5.48 | 18.18 | 54 |
| 60 | 5.53 | 19.80 | 50 |
| 70 | 5.63 | 21.01 | 55 |
| 80 | 6.05 | 23.12 | 61 |
| 90 | 6.42 | 24.98 | 69 |
| 100 | 6.80 | 26.33 | 73 |
| 110 | 7.11 | 27.41 | 75 |
| 120 | 7.75 | 28.86 | 77 |
| 130 | 8.20 | 29.94 | 80 |
| 140 | 8.66 | 30.92 | 83 |
| 150 | 8.99 | 32.46 | 91 |
| 160 | 9.44 | 33.69 | 101 |
| 170 | 9.40 | 34.53 | 102 |
| 180 | 9.54 | 35.45 | 103 |
| 190 | 9.70 | 36.35 | 104 |
| 200 | 10.12 | 37.38 | 109 |

Table 1.
Results for unoptimized graph.

When a document is added to the graph with Algorithm 5, arcs which were essential to keep the graph progressively connected may not be essential after the addition of the new document. Table 2 summarizes the results of using Algorithm 6 to optimize the graph. At each multiple of ten, the graph was optimized and average degree, average DWPL and maximum DWPL were computed.

| number of documents | average degree | average DWPL | maximum DWPL |
|---|---|---|---|
| 10 | 2.40 | 5.56 | 14 |
| 20 | 3.20 | 9.15 | 21 |
| 30 | 4.07 | 11.81 | 29 |
| 40 | 4.30 | 14.30 | 38 |
| 50 | 4.72 | 16.61 | 45 |
| 60 | 4.70 | 18.34 | 53 |
| 70 | 4.77 | 19.38 | 58 |
| 80 | 4.88 | 20.64 | 63 |
| 90 | 5.61 | 22.13 | 67 |
| 100 | 5.52 | 23.31 | 75 |
| 110 | 5.76 | 24.24 | 94 |
| 120 | 6.07 | 25.28 | 85 |
| 130 | 6.38 | 26.27 | 100 |
| 140 | 6.60 | 26.97 | 103 |
| 150 | 6.72 | 27.56 | 105 |
| 160 | 7.11 | 28.80 | 101 |
| 170 | 7.14 | 29.45 | 103 |
| 180 | 7.34 | 30.27 | 104 |
| 190 | 7.38 | 30.92 | 107 |
| 200 | 7.78 | 31.90 | 111 |

Table 2.
Results for optimized graph.

Figure 5 shows that the average degree for the unoptimized graph is increasing at a faster rate than the average degree of the optimized graph. However, when a least squares method is used to fit the equation $A+B*\log_2(X)$ to the data, it is apparent that the average degree of both the optimized graph and the unoptimized graph is increasing at a rate proportional to the log of the number of documents. The data points from 100 to 200 were used in order to estimate the rate of increase when the graph is large. The best fit for the unoptimized graph was

$$-15.1 + 3.3*\log_2(X)$$

and the best fit for the optimized graph was

$$-8.93 + 2.2*\log_2(X) \ .$$

Figure 6 shows how well these functions fit the data. In this range, the average degree is increasing at a rate proportional to the log of the number of documents.

Figure 7 shows that the average degree weighted path length is also increasing at a faster rate for the unoptimized graph than for the optimized graph, but the rate of increase for both graphs is proportional to the log of the number of documents. The best fit for the unoptimized graph is

$$-48.74 + 11.24*\log_2(X)$$

and the best fit for the optimized graph is

$$-33.32 + 8.48*\log_2(X).$$

Figure 8 shows how well these functions fit the data.

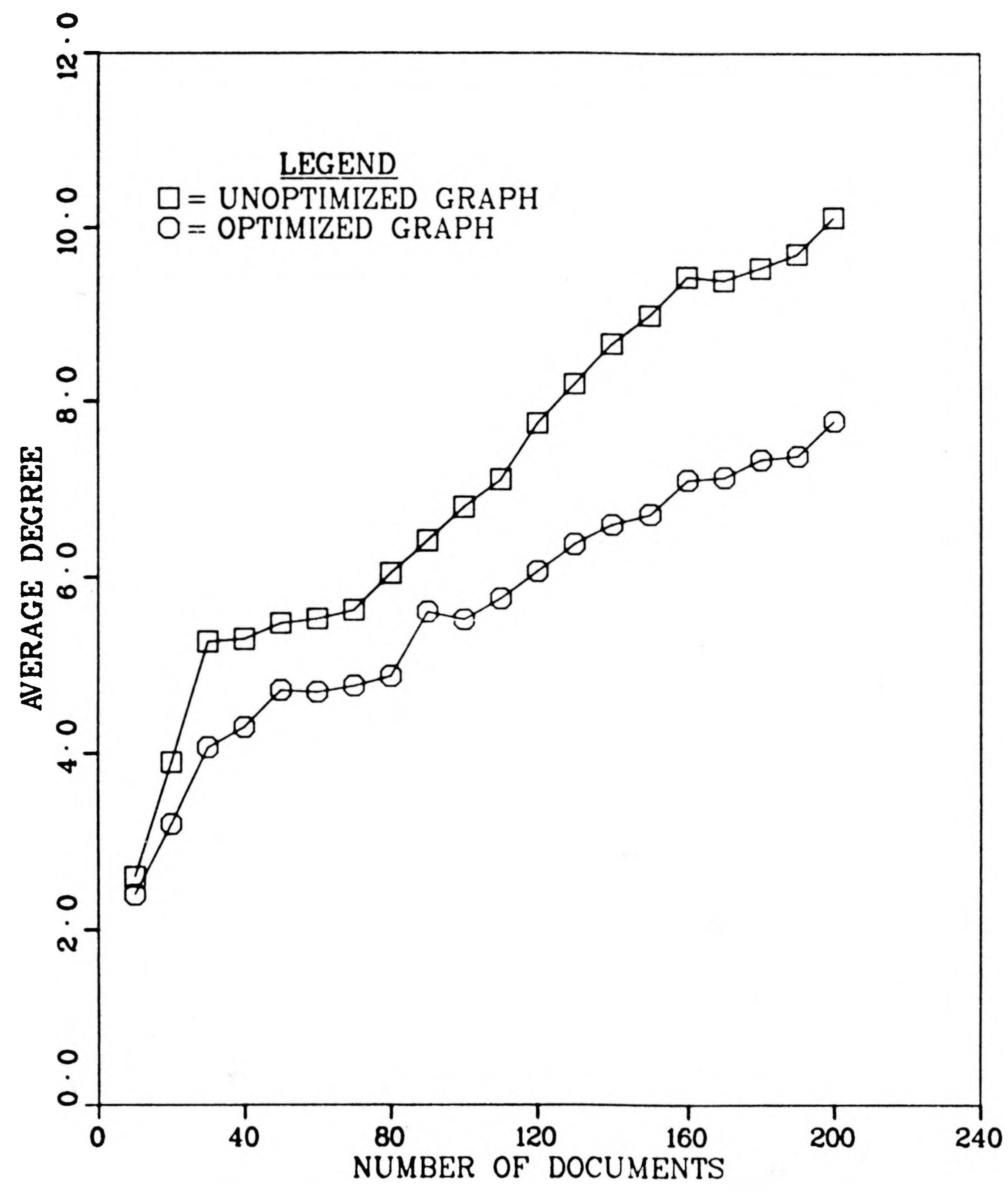

FIGURE 5.
NUMBER OF DOCUMENTS VS. AVERAGE DEGREE

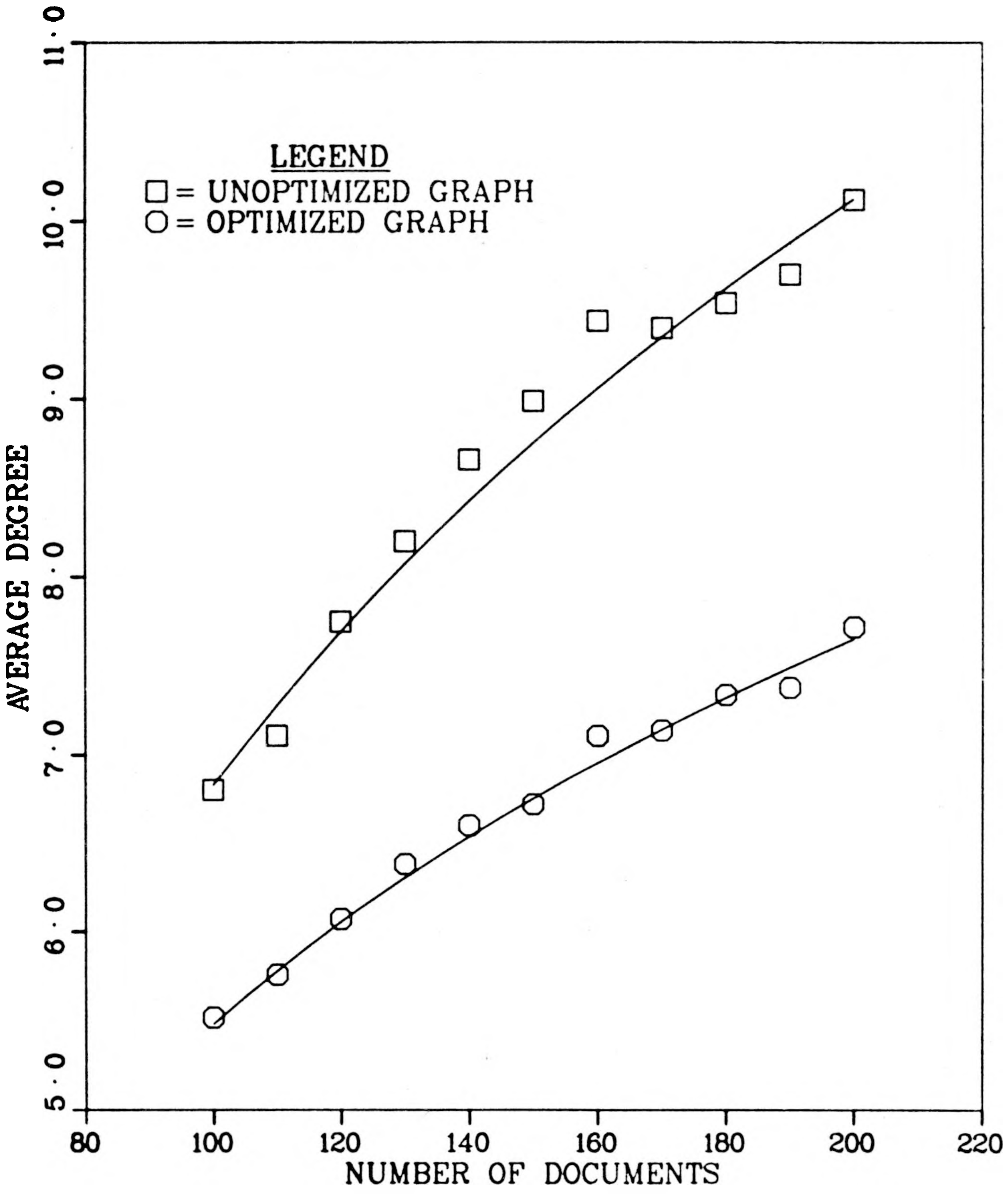

FIGURE 6.
LOGARITHMIC FIT OF THE AVERAGE DEGREE DATA.
SOLID LINE IS THE LOG CURVE.

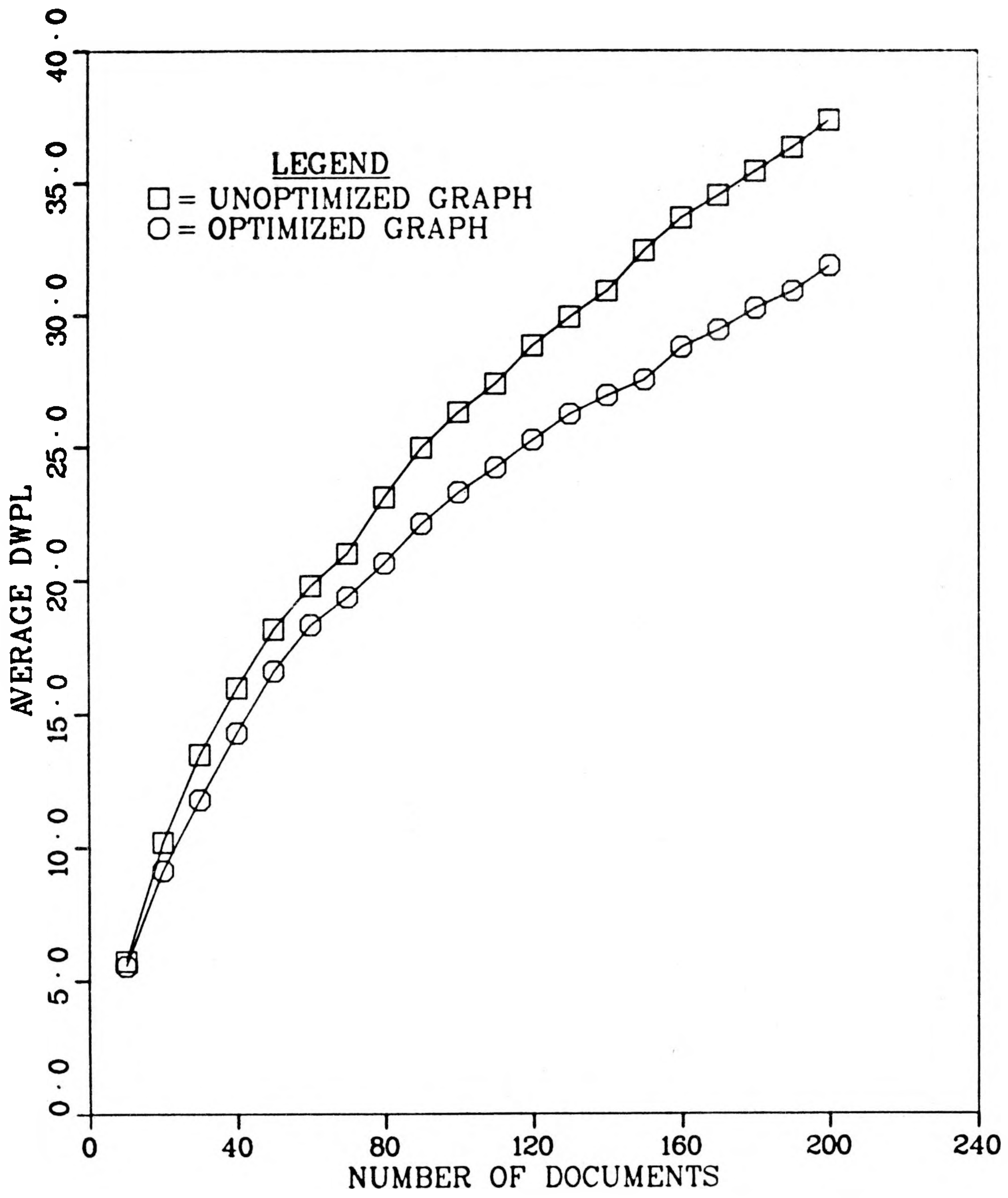

FIGURE 7.
NUMBER OF DOCUMENTS VS. AVERAGE DWPL.

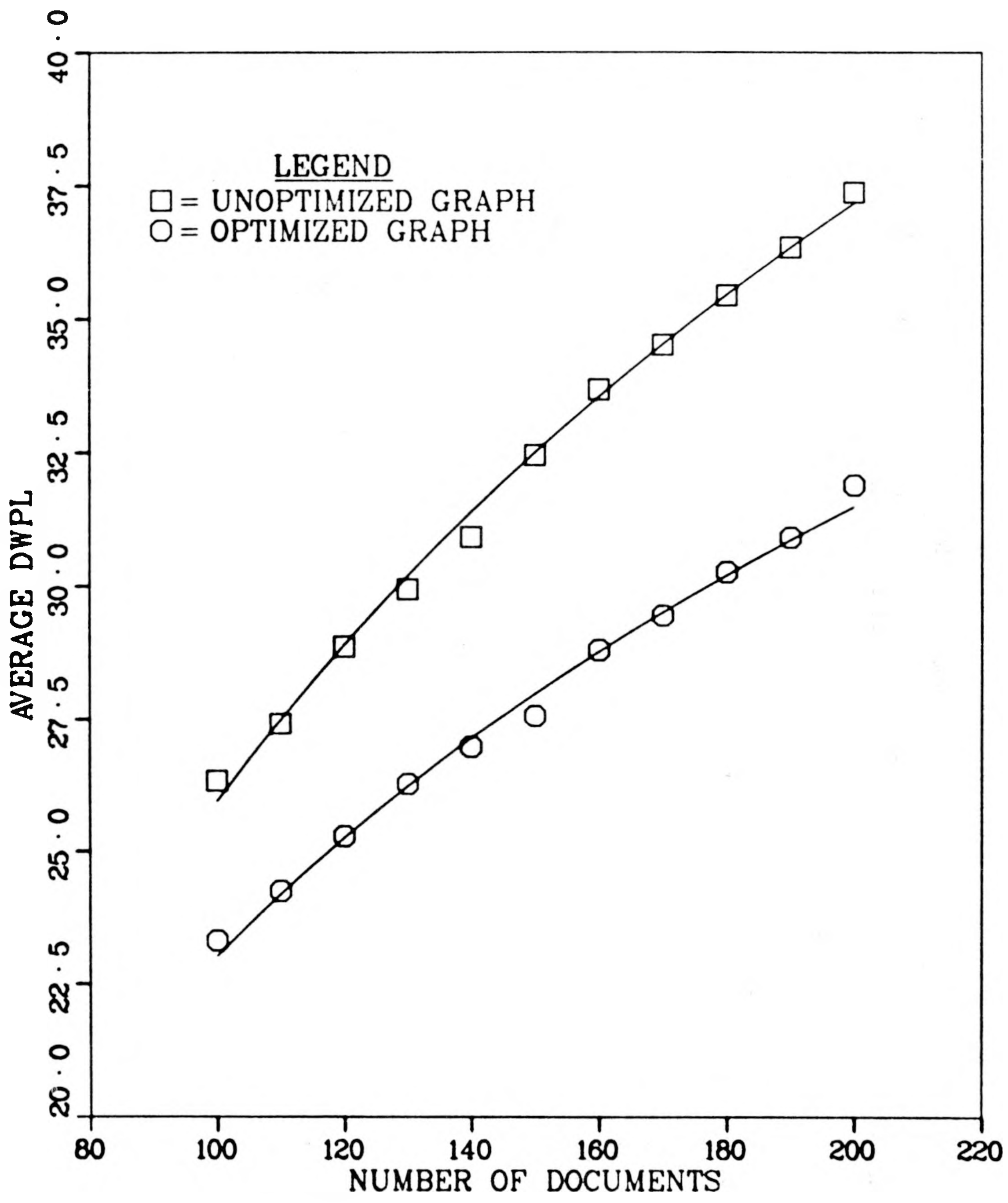

FIGURE 8.
LOGARITHMIC FIT OF THE AVERAGE DWPL DATA.
SOLID LINE IS THE LOG CURVE.

## ENUMERATION OF GRAPHS

A progressive graph on a document space with n documents will be a labeled graph of order n. Harary and Palmer give the following formulas for the enumeration of labeled graphs[4]. The total number of labeled graphs of order n is

$$G_n = 2^{(n,2)}.$$

Where (n,2) is the binomial coefficient.

$$(n,2) = n! \ / \ (n-2)! \ 2! \quad = n(n-1)/2$$

The number of connected labeled graphs of order n is

$$C_n = 2^{(n,2)} - (1/n) \sum_{k=1}^{n-1} k*(n,k)*C_k*G_{n-k}.$$

The number of labeled trees with n points is

$$T_n = n^{(n-2)}.$$

The following table shows the number of each kind of graph for $n \leq 8$.

| n | $T_n$ | $C_n$ | $G_n$ |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 |
| 3 | 3 | 4 | 8 |
| 4 | 16 | 38 | 64 |
| 5 | 125 | 728 | 1024 |
| 6 | 1296 | 26704 | 32786 |
| 7 | 16807 | 1866256 | 2097152 |
| 8 | 262144 | 251548592 | 268435546 |

Table 3.
Enumeration of labeled graphs.

Table 3 shows the magnitude of the problem if one were to make an exhaustive search for progressive graphs. The large number of connected graphs makes it impractical to search all connected graphs for progressive graphs with the smallest number of arcs. A tree is a connected graph with a minimal number of arcs. Any progressive graph is connected and must contain a spanning tree as a subgraph. A minimal progressive graph can be found by starting with one of its spanning trees and adding a sufficient number of arcs to make the graph progressive. However, the number of trees on n nodes grows exponentially as n increases. Thus, it is also impractical to find a progressive graph with the smallest number of arcs possible by adding arcs to the set of trees.

## ORDER MATRICES

A graph (S,L) is progressively connected with respect to a distance function d if and only if there exists a first step in a progressive path from each node in the graph to every other node. A first step in a progressive path from A to B exists if there is a node C which is adjacent to A and $d(C,B)<d(A,B)$. Therefore, the relationship of all pairs of distances d(A,B) and d(C,B) for arbitrary A, B and C is sufficient information to determine when a metric graph is progressively connected. When the points in the set S are labeled $X_1, X_2, \ldots, X_N$ and d is a non-isosceles distance function, this information can be organized in an NxN matrix O. The ith row $O(i,1), O(i,2), \ldots, O(i,N)$ will be a permutation of the set of integers $\{0,1,\ldots,N-1\}$ such that $O(i,j)$ is the rank of the distance $d(X_i,X_j)$ among all distances $d(X_i,X_k)$ for $k=1,2,\ldots,N$. Thus, the matrix O will have the property that $O(i,i)=0$ and $O(i,j)<O(i,k)$ if and only if $d(X_i,X_j)<d(X_i,X_k)$. I will call matrices of this form, order matrices.

DEFINITION:(Order Matrix) An NxN order matrix is a matrix with zero diagional and rows that are permutations of the set of integers $\{0,1,\ldots,N-1\}$.

Every NxN order matrix defines a relation $\hat{r}$ on the set of <u>unordered</u> pairs $\{(i,j) : i,j=1,2,\ldots,N\}$ by the following rule. $(i,j)\ \hat{r}\ (i,k)$ if and only if $O(i,j) < O(i,k)$. Thus, two pairs are unrelated unless they have one point in common and the ith row of the order matrix determines how (i,j) and (i,k) are related. I will call a sequence $X_0, X_1, \ldots, X_k$ an $\hat{r}$ step sequence if $(X_{i-1}, X_i) \ \hat{r} \ (X_i, X_{i+1})$ for all $i=1,2,\ldots,k-1$. Since two pairs are unrelated by $\hat{r}$ unless they have one point in common, any cycle will be of the form $(X_0,X_1) \ \hat{r} \ (X_1,X_2) \ \hat{r} \ \ldots \ \hat{r} \ (X_k,X_0) \ \hat{r} \ (X_0,X_1)$. This is equivalent to the $\hat{r}$ step sequence $X_0,X_1,X_2,\ldots,X_k,X_0,X_1$. Thus, any $\hat{r}$ step sequence whose first two points and last two points are equal will be called an $\hat{r}$ step cycle. If O is the matrix where O(i,j) is the rank of the distance $d(X_i,X_j)$ among all distances $d(X_i,X_k)$ in some finite metric space, then $d(X_i,X_j) < d(X_i,X_k)$ implies that $(i,j) \ \hat{r} \ (i,k)$.

DEFINITION:(Compatible order matrix) The NxN order matrix O is compatible with a finite labeled metric space $(\{X_1,\ldots,X_N\},d)$ if it is true that $d(X_i,X_j)<d(X_i,X_k)$ if and only if $(i,j)\hat{r}(i,k)$.

Given a set of three points, S={1,2,3}, there are only six ways the distances d(1,2), d(1,3) and d(2,3) can be related when d is a non-isosceles distance function.

(1) $d(1,2) < d(1,3) < d(2,3)$

(2) $d(1,2) < d(2,3) < d(1,3)$

(3) $d(1,3) < d(1,2) < d(2,3)$

(4) $d(1,3) < d(2,3) < d(1,2)$

(5) $d(2,3) < d(1,2) < d(1,3)$

(6) $d(2,3) < d(1,3) < d(1,2)$

Each relation corresponds to one of the six different ways a three point metric space can be labeled. The following six order matrices are compatible with any three point metric space whose distances satisfy relations (1) through (6) respectively.

$$\begin{array}{cccccc}
(1) & (2) & (3) & (4) & (5) & (6) \\
\begin{matrix} 0 & 1 & 2 \\ 1 & 0 & 2 \\ 1 & 2 & 0 \end{matrix} &
\begin{matrix} 0 & 1 & 2 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{matrix} &
\begin{matrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 2 & 0 \end{matrix} &
\begin{matrix} 0 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{matrix} &
\begin{matrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \end{matrix} &
\begin{matrix} 0 & 2 & 1 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \end{matrix}
\end{array}$$

There are $2^3=8$ ways to arrange the rows of a 3x3 order matrix. The following two order matrices are not listed above.

$$\begin{array}{cc}
(7) & (8) \\
\begin{matrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{matrix} &
\begin{matrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{matrix}
\end{array}$$

Order matrix (7) defines the following relation on the unordered pairs.

$$(1,2)\ \hat{r}\ (1,3)$$

$$(2,3)\ \hat{r}\ (2,1)$$

$$(3,1)\ \hat{r}\ (3,2)$$

Since the pairs are unordered, this relation has the following cycle.

$$(1,2)\ \hat{r}\ (1,3)=(3,1)\ \hat{r}\ (3,2)=(2,3)\ \hat{r}\ (2,1)=(1,2)$$

If order matrix (7) were compatible with some metric space, then this would imply that $d(X_1,X_2)<d(X_1,X_2)$. Since this cannot be true for any metric space, order matrix (7) cannot be compatible with any three point metric space. Similarly, order matrix (8) cannot be compatible with any three point metric space because the relation defined by order matrix (8)

has the following cycle.

$(1,3)\ \hat{r}\ (1,2)=(2,1)\ \hat{r}\ (2,3)=(3,2)\ \hat{r}\ (3,1)=(1,3)$

Order matrices (1) through (6) are all compatible with the same three point metric space. This is true because each of the relations (1) through (6) corresponds to one of the six different ways a three point metric space can be labeled. The following definition gives a semi-canonical form for order matrices which will eliminate most reorderings.

DEFINITION:(Semi-Canonical order matrix) An NxN order matrix O is semi-canonical if

1) $O(1,i) = i-1$ for $i=1,2,\ldots,N$
2) $O(2,1) = 1$
3) $O(3,1) < O(3,2)$

Order matrix (1) is a 3x3 semi-canonical order matrix. A semi-canonical order matrix is compatible with a metric space which has been labeled such that points 2 through N are labeled in increasing order of their distnace from point 1, points 1 and 2 are mutually closest, and $d(1,3)<d(2,3)$. In a three point non-isosceles metric space there is a unique labeling which satisfies these conditions. This unique labeling is the labeling which specifies that $d(1,2)<d(1,3)<d(2,3)$.

DEFINITION:(Consistent order matrix) An NxN order matrix is consistent if the relation $\hat{r}$ defined by O on the unordered pairs does not contain any cycles.

If the relation defined by an order matrix does not contain any cycles, then it can be embedded in a linear order. Knuth's topological sort algorithm can be used to check an order matrix for consistency[5,p. 258]. If the order matrix is consistent, the topological sort algorithm will find a linear order which contains the relation $\hat{r}$. If the order matrix is inconsistent, the topological sort algorithm will find a cycle in $\hat{r}$. Matrices (1) through (6) are consistent while matrices (7) and (8) are inconsistent.

<u>THEOREM</u> 6: Every non-isosceles metric space can be labeled so that it is compatible with a consistent semi-canonical order matrix. Conversely, every consistent semi-canonical order matrix is compatible with some non-isosceles metric space.

PROOF: Let (S,d) be a non-isosceles metric space with N points. Since there are only a finite number of points, there exist two points X and Y such that d(X,Y) is a minimal distance. Let $Z_x$ be the second closest point to X and $Z_y$ be the second closest point to Y. One of the following inequalities must hold.

$$d(Z_x,X) < d(Z_x.Y)$$
or
$$d(Z_y,Y) < d(Z_y,X)$$

If neither inequality holds, then the following cycle will occur.

$$d(Z_x,Y) < d(Z_x,X) \leq d(Z_y,X) < d(Z_y,Y) \leq d(Z_x,Y)$$

This contradicts the fact that $\leq$ is a partial order on the real numbers. If $d(Z_x,X)<d(Z_x,Y)$ then label X as one, Y as two, and $Z_x$ as three. Otherwise, label Y as one, X as two and $Z_y$ as three. Having labeled points

one through three, the remaining nodes are labeled in increasing order of their distance from point one. If O is the order matrix with $O(i,j)$ equal to the rank of the distance $d(i,j)$ among all distances $d(i,k)$, then O is compatible with the metric space and O will have the following properties.

| | |
|---|---|
| $O(1,k)=k-1$ | $k=1,2,\ldots,N$ |
| $O(2,1)=1$ | since $d(1,2)$ was minimal |
| $O(3,1)<O(3,2)$ | since $d(3,1)<d(3,2)$ |

This shows that O is semi-canonical. Conversely, suppose O is an NxN consistent semi-canonical order matrix. Since O is a consistent order matrix, the relation defined by O does not contain any cycles. This relation on the $M=N(N-1)/2$ pairs $\{(i,j) : i\neq j\}$ can be embedded in a linear order $p_1<p_2<\ldots<p_M$. Define a distance function $d:\{1,\ldots,N\}\times\{1,\ldots,N\}\text{-->}\{\text{real numbers}\}$ by

| | |
|---|---|
| $d(i,i)=0$ | for $i=1,2,\ldots,N$ |
| $d(i,j) = 1 + k/M$ | for $i\neq j$ and $(i,j)$ is the kth pair in the linear order $p_1<p_2<\ldots<p_M$ |

This distance function satisfies the triangle inequality since all distances are less than or equal to two and the sum of any two distances is strictly greater than two. The order matrix O is compatible with the metric space $(\{1,2,\ldots,N\},d)$.

Two metric spaces $(S_1,d_1)$ and $(S_2,d_2)$ are isometric if there exists a one to one function $f:S_1 \dashrightarrow S_2$ which preserves distances (i.e., $d_1(a,b) = d_2(f(a),f(b))$ ). A more general morphism is one that only preserves the order relation on the distances. $(S_1,d_1)$ is order isomorphic to $(S_2,d_2)$ if for all a, b, c and d in $S_1$, it is true that

$d_1(a,b)<d_1(c,d)$ implies $d_2(f(a),f(b))<d_2(f(c),f(d))$.

Order isomorphic metric spaces will be compatible with the same order matrix when the order isomorphism f maps the $i^{th}$ point in one metric space to the $i^{th}$ point in the other metric space. The following theorem shows that every finite metric space is order isomorphic to a Euclidean metric space. It follows by Theorem 6 that every consistent order matrix is compatible with some Euclidean metric space.

THEOREM 7: Let $(S,d)$ be any finite non-isosceles metric space with n points. Then there exists a subspace of Euclidean n-space which is order isomorphic to $(S,d)$.

PROOF: Label the metric space $(S,d)$ so that its corresponding order matrix is semi-canonical. I will show that there exists a set of vectors $\{X_i\}_{i=1}^{n}$ in Euclidean n-space such that for some real number $e>0$,

$$|X_i-X_j| = 1 + e*d(i,j).$$

If such a set of vectors exists, then the function $f:S \dashrightarrow \{X_i\}$ defined by $f(i)=X_i$ is an order isomorphism because

$$d(i,j)<d(n,m) \text{ implies } 1 + e*d(i,j) < 1 + e*d(n,m)$$
$$\text{implies } |X_i-X_j| < |X_n-X_m|.$$

The following proof constructs a set of n vectors close to the n scaled unit vectors

$$\begin{array}{llllll}
( & (1/2)^{1/2}, & 0, & 0, & 0, & \dots ) \\
( & 0, & (1/2)^{1/2}, & 0, & 0, & \dots ) \\
( & 0, & 0, & (1/2)^{1/2}, & 0, & \dots ) \\
 & \vdots & \vdots & \vdots & \vdots &
\end{array}$$

Define the vector functions

$$X_i(e) = (X_{i1}(e),\dots,X_{ii}(e))$$

by the following recursive process. First set

$$X_{1\ 1}(e)=(1/2)^{1/2}.$$

Suppose that for all $j<i$, $X_j(e)$ is defined and continuous on some interval $[0,t_{i-1}]$ such that,

i) $|X_j(e)-X_k(e)| = 1 + e*d(j,k)$

ii) $|X_j(e)|^2 = 1/2$

iii) $X_j(0) = (0,0,\dots,(1/2)^{1/2})$

A vector function $X_i(e)$ must be defined such that it is continuous on some interval $[0,t_i]$ and satisfies condtions i) through iii). $X_i(e)$ can be defined in terms of $\{X_j(e):j=1,\dots,i-1\}$. In order to satisfy condition i), $X_i(e)$ must be a solution to the following equation for all $j<i$.

$$[1 + e*d(i,j)]^2 = |X_i(e)-X_j(e)|^2$$
$$= \sum_{k=1}^{i} [X_{ik}(e) - X_{jk}(e)]^2$$
$$= |X_i(e)|^2 + |X_j(e)|^2 + 2\sum_{k=1}^{i} X_{ik}(e)*X_{jk}(e)$$

In order to satisfy condition ii), $X_i(e)$ must be a solution to these equations with the constraint that $|X_i(e)|=1/2$. Using this constraint, the system of i-1 equations becomes,

$$\sum_{k=1}^{j} X_{ik}(e)*X_{jk}(e) = - e*d(i,j) - e^2*d^2(i,j)/2.$$

Since $X_{jk}(e)=0$ for all $k>j$, this is a triangular system of i-1 equations and will have the following solution everywhere $X_{jj}(e)\neq 0$ for all $j<i$.

$$X_{ij}(e) = \frac{- e*d(i,j) - e^2*d^2(i,j)/2 - \sum_{k=1}^{j-1} X_{ik}(e)*X_{jk}(e)}{X_{jj}(e)}$$

Since $X_{jj}(0)=(1/2)^{1/2}$ for all $j<i$, there exists an interval $[0,t]$ in which $X_{jj}(e)>0$. Thus, all the $X_{ij}(e)$ are defined and continuous on $[0,t]$. As long as $X_{ii}(e)$ is chosen such that $|X_i(e)|^2=1/2$, $X_i(e)$ will satisfy condition i). Since, $X_{jj}(0)=(1/2)^{1/2}$ and $X_{jk}(0)=0$ for all $k<j$, it is clear that $X_{ij}(0)=0$. Finally, $X_{ii}(e)$ can be defined by:

$$X_{ii}(e) = [1/2 - \sum_{k=1}^{i-1} X_{ik}^2(e)]^{1/2}$$

For all $k<j$, $X_{ik}(e)$ is continuous on $[0,t]$ with

$X_{ik}(0)=0$. Therefore, function $X_{ii}(e)$ is defined and continuous on some possibly smaller interval $[0,t_i]$. By the definition of $X_{ii}(e)$, it is clear that $X_i(e)$ satisfies conditions ii) and iii) and is continuous on the interval $[0,t_i]$.

In the paragraphs that follow, a recursive method is given for constructing all NxN consistent semi-canonical order matrices. If $O^{N+1}$ is an (N+1)x(N+1) order matrix, is the NxN matrix generated by eliminating the $N^{th}$ row and $N^{th}$ column an order matrix? In general the resulting NxN submatrix will not be an order matrix because its rows will not be a permutation of the integers $\{0,1,\ldots,N-1\}$. The following theorem describes how to generate an NxN order matrix from an (N+1)x(N+1) order matrix.

<u>THEOREM</u> 8: Let $O^{N+1}$ be an (N+1)x(N+1) matrix, define the mapping to an NxN matrix $O^N$ by:

$$o^N(i,j) = \begin{cases} O^{N+1}(i,j) & \text{if } O^{N+1}(i,j) < O^{N+1}(i,N+1) \\ O^{N+1}(i,j) - 1 & \text{otherwise} \end{cases}$$

If $O^{N+1}$ is a consistent semi-canonical order matrix then so is $O^N$. In fact, $O^N$ is compatible with the labeled metric space formed by deleting the point $X_{N+1}$ from a labeled metric space which is compatible with $O^{N+1}$.

PROOF: It is sufficient to show that $O^{N+1}(i,j) <$ $O^{N+1}(i,k)$ implies that $O^N(i,j) < O^N(i,k)$. This shows that both matrices $O^{N+1}$ and $O^N$ define the same relation on pairs of the first N points.

Suppose that $O^{N+1}(i,j) < O^{N+1}(i,k)$. If $O^{N+1}(i,N+1)$ is less than both $O^{N+1}(i,j)$ and $O^{N+1}(i,k)$ then $O^{N}(i,j) = O^{N+1}(i,j)-1 < O^{N+1}(i,k)-1 = O^{N}(i,k)$. If $O^{N+1}(i,N+1)$ is greater than both $O^{N+1}(i,j)$ and $O^{N+1}(i,k)$ then $O^{N}(i,j) = O^{N+1}(i,j) < O^{N+1}(i,k) = O^{N}(i,k)$. If $O^{N+1}(i,N+1)$ is greater than $O^{N+1}(i,j)$ but less than $O^{N+1}(i,k)$ then $O^{N}(i,j) = O^{N+1}(i,j) < O^{N+1}(i,k)-1 = O^{N}(i,k)$.

The mapping $O^{N+1} \dashrightarrow O^{N}$ is not one to one. There may be more than one (N+1)x(N+1) order matrix which maps into the same NxN order matrix by this process. When a consistent semi-canonical NxN order matrix is known, it is of interest to find all the consistent semi-canonical (N+1)x(N+1) order matrices which map into it. The following theorem characterizes these matrices.

THEOREM 9: Let $O^{N+1}$ be an (N+1)x(N+1) order matrix which maps into a consistent NxN order matrix $O^{N}$. $O^{N+1}$ is consistent if and only if the relation defined on the unordered pairs by $O^{N+1}$ does not contain any $\hat{r}$ step cycles of the form $N+1,i,\ldots,N+1,i$.

PROOF: If the relation defined by $O^{N+1}$ has an $\hat{r}$ step cycle $N+1,i,\ldots,N+1,i$ , then by definition it is inconsistent.

Conversely, assume that $O^{N+1}$ is inconsistent. Then, the relation defined by $O^{N+1}$ contains an $\hat{r}$ step cycle. By the proof of theorem 8, we know that $O^{N+1}$ and $O^N$ define the same relation on pairs of the first N points. Since $O^N$ is consistent, there are no $\hat{r}$ step cycles in the first N points. Therefore, the $\hat{r}$ step cycle which exists must contain the (N+1)st point. By starting at the (N+1)st point the cycle has the form $N+1, i, \ldots, N+1, i$.

Let A be a permutation and B a sequence of the integers $\{1,2,\ldots,N-1\}$. Define a matrix $O^{N+1}$ by:

$$O^{N+1}(N+1,N+1) = 0$$

$$O^{N+1}(N+1,i) = A(i)$$

$$O^{N+1}(i,N+1) = B(i)$$

$$O^{N+1}(i,j) = \begin{cases} O^N(i,j) & \text{if } O^N(i,j) < B(i) \\ O^N(i,j) + 1 & \text{if } O^N(i,j) \geq B(i) \end{cases}$$

$$O^{N+1} = \left[\begin{array}{cc} \left[\begin{array}{c} \\ O^N \\ \text{modified} \\ \\ \end{array}\right] & \begin{array}{c} B(1) \\ B(2) \\ \cdot \\ \cdot \\ \cdot \\ B(N) \end{array} \\ A(1)\ A(2)\ \ldots\ A(N) & 0 \end{array}\right]$$

Figure 9.
(N+1)x(N+1) order matrix $O^{N+1}$

Every (N+1)x(N+1) order matrix which maps into $O^N$ will be of this form. Thus, to find all the consistent semi-canonical (N+1)x(N+1) order matrices which map into $O^N$, it is sufficient to find all permutations A and sequences B such that $O^N$, A and B define a consistent semi-canonical order matrix.

If $O^{N+1}$ is semi-canonical, then $O^{N+1}(1,N+1)=N$ and $O^{N+1}(2,1)=1$. Therefore, the sequence B must have $B(1)=N$ and $B(2) \geq 2$. If $B(i)$ is set to N for all i, then $O^{N+1}(i,N+1)=B(i)=N > O^N(i,k)$ for all k. This implies that N+1,i,k cannot be an $\hat{r}$ step sequence for any k. Therefore, $O^{N+1}$ is a consistent semi-canonical order matrix regardless of what permutation is chosen for the (N+1)st row. Since there are N! permutations of $\{1,2,\ldots,N\}$, there are at least N! consistent semi-canonical order matrices which map into $O^N$.

The previous discussion shows that $B(m)=N$ is a sufficient condition to assure that N+1,m,...,N+1,m is not an $\hat{r}$ step sequence. A necessary and sufficient condition is that $B(m) > O^N(m,j)$ for all j such that there is an $\hat{r}$ step sequence of the form m,j,...,N+1. If $B(m) \leq O^N(m,j)$ for some j such that m,j,...,N+1,m is an $\hat{r}$ step sequence, then N+1,m,j,...,N+1,m is an $\hat{r}$ step cycle. This is because $B(m) \leq O^N(m,j)$ implies that $O^{N+1}(m,N+1) < O^{N+1}(m,j)$.

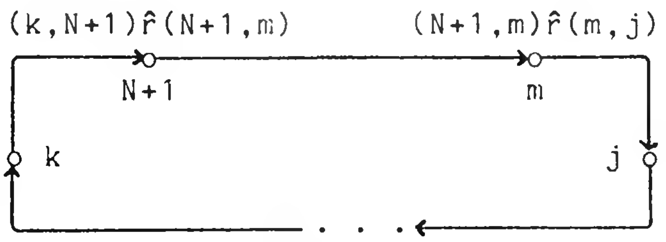

Figure 10.
$\hat{r}$ step cycle N+1,m,j,...,k,N+1,m

Conversely, assume that $B(m) > O^N(m,j)$ for all j such that m,j,...,N+1,m is an $\hat{r}$ step sequence. Let N+1,m,j,...,N+1,m be any sequence. If m,j,...,N+1,m is not an $\hat{r}$ step sequence, then N+1,m,j,...,N+1,m cannot be an $\hat{r}$ step cycle. On the other hand, if m,j,...,N+1,m is an $\hat{r}$ step sequence, then $B(m) > O^N(m,j)$. This implies that $(m,j)\hat{r}(m,N+1)$, so j,m,N+1 is an $\hat{r}$ step sequence. Therefore, N+1,m,j is not an $\hat{r}$ step sequence, and the complete sequence N+1,m,j,...,N+1,m cannot be an $\hat{r}$ step cycle.

Given a consistent semi-canonical order matrix $O^N$, the following algorithm generates all consistent semi-canonical (N+1)x(N+1) order matrices which map into $O^N$. For each permutation A, the algorithm generates all possible sequences B such that $O^N$, A and B define a consistent semi-canonical order matrix $O^{N+1}$. Let S be a permutation such that A(S(i))=i for all i (i.e., S(i) is the index of i in permutation A). The algorithm uses the fact that if S(m),j,...,N+1,S(m) is an $\hat{r}$ step sequence then there exists k<m such that S(m),j,...,S(k),N+1,S(m) is an $\hat{r}$ step sequence. Therefore, B(S(m)) must be greater than $O^N(S(m),j)$ for all j such that S(m),j,...,S(k),N+1 is an $\hat{r}$ step sequence for some k<m. It

is possible that j=S(k) in this sequence.

ALGORITHM 7: (Generate all (N+1)x(N+1) consistent semi-canonical order matrices from an NxN consistent semi-canonical order matrix)

```
subroutine genorder(n,O)
    "O is an NxN order matrix."
sequence S,A,B
array O(N,N),K(N,N)
number i,k
S := A := <1,2,...,N>
until A=<> do
   for i=1,2,...,N do
      for j=1,2,...,N do
         k(i,j) := 0
   call order(1,K)
      "Subroutine order generates all sequences B
       such that O, A and B define a (N+1)x(N+1)
       consistent semi-canonical order matrix."
   A := nextperm(A)
       " Nextperm generates the next permutation in
         lexicographical order.  If the input permutation
         is <N,N-1,...,1> then nextperm returns the null
         sequence <>."
   S := gradeup(A)
       " gradeup generates a sequence where A(S(i))=i."
return
end
```

```
subroutine order(m,K)
   "For all i<m, B(S(i)) has been defined such that cycles
    of the form N+1,S(i),...,N+1,S(i) cannot occur."

   "K is an NxN array with the property that K(i,j)=1
    if there exists an $\hat{r}$ step sequence i,j,...,S(k),N+1
    for some k<m.  Otherwise, K(i,j)=0."
global N,O,S,B
   "O - The NxN matrix being expanded."
   "S - A permutation such that A(S(i))=i."
   "B - The sequence specifying the (N+1)st column."
sets F,NF
number N,n,i,j,m,r,t
array K(N,N),O(N,N)

if m $\le$ N then
   n := 1 + max{ O(S(m),j)*K(S(m),j) : j=1,2,...,N}
      "n is the smallest value B(S(m)) can
       assume and assure that there cannot be
       any cycles N+1,S(m),...,N+1,S(m)."
   if S(m)=1 then n := N  "B(1) must be N."
   if S(m)=2 then n := max{n,2}"B(2) must be greater than 1."
   for n $\le$ t $\le$ N do
      B(S(m)) := t
      F := { <S(m),j> : O(S(m),j)<B(S(m)) and K(j,S(m))=0}
         "Put in F all <S(m),j> such that j,S(m),N+1 is an
          $\hat{r}$ step sequence and K(j,S(m))=0.  If K(j,S(m))=1,
          then all $\hat{r}$ step sequences ending in j,S(m),...,N+1
          have been found before."
      until F={} do
         NF := {}
         for <i,j> in F do
            K(j,i) := 1
            NF := NF.union.{<j,r>:O(j,r)<O(j,i) and K(r,j)=0}
               "O(j,r)<O(j,i) implies that r,j,i,...,S(m),N+1
                is an $\hat{r}$ step sequence.  If K(r,j)=0,
                then an $\hat{r}$ step sequence r,j,...,S(k),N+1
                has been found before for some k$\le$m."
         F := NF
      call order(m+1,K)
         "B(S(m)) has been defined such that $\hat{r}$ step
          cycles N+1,S(m),...,N+1,S(m) cannot occur."
         "K has been updated so that K(i,j)=1 when
          there is an $\hat{r}$ step sequence i,j,...,S(m),N+1."
   return

else
   "If n>N then every element B(i) has been defined
    so that cycles of the form N+1,i,...,N+1,i
    cannot occur."
   "By theorem 9, O,A, and B define an (N+1)X(N+1)
    consistent semi-canonical order matrix."
end
```

After B(S(i)) has been defined for all i<m, procedure order generates an order matrix for every permissible value of B(S(m)). Thus, every (N+1)x(N+1) order matrix must be generated by Algorithm 7. Since each NxN matrix generates at least N! (N+1)x(N+1) order matrices, the number of semi-canonical order matrices is a very rapidly growing function of N. Table 4 lists all the 4x4 semi-canonical order matrices in the order they were generated by Algorithm 7 from matrix (1) listed on page 41. Recall that this is the only 3x3 semi-canonical order matrix.

| | | | | | |
|---|---|---|---|---|---|
| (1) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\1&2&3&0\end{matrix}$ | (2) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&2&0&3\\1&2&3&0\end{matrix}$ | (3) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\1&2&3&0\end{matrix}$ | (4) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\1&3&2&0\end{matrix}$ | (5) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&3&0&2\\1&3&2&0\end{matrix}$ | (6) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\1&3&2&0\end{matrix}$ |
| (7) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\2&1&3&0\end{matrix}$ | (8) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&2&0&3\\2&1&3&0\end{matrix}$ | (9) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\2&1&3&0\end{matrix}$ | (10) $\begin{matrix}0&1&2&3\\1&0&3&2\\2&3&0&1\\2&3&1&0\end{matrix}$ | (11) $\begin{matrix}0&1&2&3\\1&0&2&3\\2&3&0&1\\2&3&1&0\end{matrix}$ | (12) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\2&3&1&0\end{matrix}$ |
| (13) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&3&0&2\\2&3&1&0\end{matrix}$ | (14) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\2&3&1&0\end{matrix}$ | (15) $\begin{matrix}0&1&2&3\\1&0&3&2\\2&3&0&1\\3&1&2&0\end{matrix}$ | (16) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\3&1&2&0\end{matrix}$ | (17) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&2&0&3\\3&1&2&0\end{matrix}$ | (18) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\3&1&2&0\end{matrix}$ |
| (19) $\begin{matrix}0&1&2&3\\1&0&3&2\\2&3&0&1\\3&2&1&0\end{matrix}$ | (20) $\begin{matrix}0&1&2&3\\1&0&2&3\\2&3&0&1\\3&2&1&0\end{matrix}$ | (21) $\begin{matrix}0&1&2&3\\1&0&3&2\\1&3&0&2\\3&2&1&0\end{matrix}$ | (22) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&3&0&2\\3&2&1&0\end{matrix}$ | (23) $\begin{matrix}0&1&2&3\\1&0&2&3\\1&2&0&3\\3&2&1&0\end{matrix}$ | |

Table 4.
4x4 semi-canonical order matrices.

Since there was only one 3x3 semi-canonical order matrix, a three point metric space is compatible with a unique semi-canonical order matrix. The following discussion will show that there exist some four point metric spaces which are compatible with two semi-canonical order matrices.

Let O be an order matrix which is compatible with an N point labeled metric space. If the $i^{th}$ and $j^{th}$ labels are interchanged in the metric space, then the order matrix O' which is compatible with this modified labeling is simply matrix O with the $i^{th}$ and $j^{th}$ rows interchanged and the $i^{th}$ and $j^{th}$ columns interchanged. Since any permutation of the labels is a product of transpositions, the order matrix compatible with a relabeling of this metric space can be generated by a series of interchanges of the rows and columns of order matrix O. A series of interchanges of the rows and columns is equivalent to the transformation

$$P \; O \; P^T$$

where P is a permutation matrix and $P^T$ is the transpose of P. Thus, if there exists a permutation matrix P such that $O' = P \; O \; P^T$, then O' and O are compatible with a relabeling of the same metrix space. In particular, order matrices (7) and (17) in Table 4 are compatible with a relabeling of the same metric space. The following transformation shows that if labels one and two are interchanged and labels three and four are interchanged in a metric space compatible with order matrix (17), then the metric space will be compatible with order matrix (7).

$$\begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 1 & 3 & 0 & 2 \\ 2 & 1 & 3 & 0 \end{matrix} = \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix} * \begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 1 & 2 & 0 & 3 \\ 3 & 1 & 2 & 0 \end{matrix} * \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

## CONCLUSIONS

Porgressive graphs are useful when they have been constructed so that the average degree weighted path length and average degree are small. The graphs constructed in the test problem have the property that both degree weighted path length and average degree are increasing at a rate proportional to the number of nodes in the graph. This implies that the amount of work the search algorithms must perform is proportional to the log of the number of nodes. However, the time required to construct the graphs is increasing at a rate approximately proportional to the square of the number of nodes, and the time to optimize the graphs was approximately proportional to the cube of the number of nodes.

The consistent order matrices contain all the information about compatible metric spaces which is relevant to the study of progressive graphs. Therefore, progressive graphs can be studied relative to order matrices. Although only the non-isosceles metric spaces can be compatible with an order matrix, this restriction to non-isosceles distance functions is not significant. Any isosceles distance function in a finite metric space can be converted to a non-isosceles distance function without changing the relative distances in any non-isosceles triangle.

# APPENDIX 1

Madcap 6 programs[6,10]

ALGORITHM 1:

```
loc ← ∝
Y : description

X ← location
Z ← location

X ← seedloc
while des(X)≠Y :

      A ← adj(X)

      for Z∈A ∋ dist(des(Z),Y)

           < dist(des(X),Y) : X←Z

 ¢return¢ X

 ↣
```

ALGORITHM 2:

```
acx ← «
r : real
x : location
(a,f) ← location

NBR ← realset ¢neighborhood¢
PER ← realset ¢periphery¢
NF  ← realset ¢new frontier¢
OF  ← realset ¢old frontier¢

NBR ← {}
PER ← {}
OF ← {x}

until #OF = 0 :
      NF ← {}
      NBR ← NBR ∪ OF
      for  f ∈ OF :
            A ← {a:a∈adj(f)} ~ (NBR∪NF∪PER)

            IN ← { a : a∈A while true ∋
             dist(des(a),des(x)) < r }

            NF ← NF ∪ IN
            PER ← PER ∪ (A~IN)

      OF ← NF

¢return¢ NBR
»
```

ALGORITHM 3:

```
closest ← «
q : description

(a,f,x) ← location
(NBR,OF,NF) ← realset
r ← real

closer ← true
x ← seedloc

while closer ¢nodes are found¢ :
      x0 ← x

      ¢check nodes adjacent to x¢
      for a∈adj(x) ∋ dist(des(a),q)<dist(des(x),q) :

             x←a


     if x=x0 : ¢check a neighborhood of radius 2r¢
           r  ← dist(des(x),q)
           (NBR,OF)←{x}
           until #OF=0 ∨ x≠x0 :
                 NF ← {}
                 for f∈OF while x=x0 :
                       A ← {a:a∈adj(f)} ~ NBR

                       NBR ← NBR ∪ A

                       if [ ∃_{a∈A} dist(des(a),q) < r ] :

                             x←a

                         else:
                          IN ← {a:a∈A while true ∋
                           dist(des(a),des(x)) ≤ 2 r }

                          NF ← NF ∪ IN

                 OF ← NF

       if x=x0 : closer ← false
 ¢return¢ x
 »
```

ALGORITHM 4:

```
findn ← «
n : real
q : description

(a,r,p) ← location
y ← closest(q)
dqy ← dist(des(y),q)
list ← (y)
total ← n
PER ← {p:p∈adj(y)}

count ← 1
until #PER=0  ∨  count≥total :
     NBR ← {nearest(des(y),PER)}
     until #NBR=0 ∨ count≥total :
          z ← nearest(des(y),NBR)
          r ← dist(des(z),q) + dqy
          OF ← {p:p∈PER while true ∋
           dist(des(p),des(y)) ≤ r } ∪ {z}
          PER ← PER ~ OF
          until #OF=0 :
               NF ← {real: 0 items}
               NBR ← NBR ∪ OF
               for f∈OF :
                    A ← {a:a∈adj(f) while true ∋
                     ¬(a∈list ∨ a∈NBR ∨ a∈PER ∨
                       a∈NF) }
                    IN ← {a:a∈A while true ∋
                     dist(des(a),des(y)) ≤ r }
                    NF ← NF ∪ IN
                    PER ← PER ∪ (A~IN)
               OF ← NF
          NEW ← {a:a∈NBR while true ∋
           dist(des(a),q) ≤ dist(des(z),q) }
          NBR ← NBR ~ NEW
          ¢sort the elements in NEW and append¢
          ¢them to list¢
          list ← append(list,setsort(q,NEW))
          count ← count + #NEW
¢return¢list
»
```

ALGORITHM 5:

```
add1 ← «
x : description

(y,z) ← description
(locx,locy,locz) ← location

¢Create a new node for the new document.¢
N ← ⟨↓des:⟨real:0 items⟩;↓adj:⟨real:0 items⟩⟩
des↑N ← x
locx ← #graph
graph ← append(graph,⟨N⟩)

for 0≤locy<locx :
     ¢Check for a first step in a progressive path¢
     ¢from all old documents to the new document.¢

     y ← des(locy)

     if $[\forall_{z \in adj(locy)} dist(y,x) \leq dist(des(z),x)]$ :
          tie(locx,locy)

for 0≤locy<locx :
     ¢check for a first step in a progressive path¢
     ¢from the new document to all old documents¢

     y ← des(locy)

     if $[\forall_{z \in adj(locx)} dist(x,y) \leq dist(des(z),y)]$ :
          tie(locx,locy)
graph
»
```

algorithm 6:

```
optimize ← «
(x,y) ← location

removable ← «
(x,y) : location
(a,z) ← location

d ← «(x,y):location ;  dist(des(x),des(y))»

$\forall_{0\le z<\#graph}$(
      z=x ∨ [$\exists_{a\in adj(x)}$ a≠y ∧ d(a,z)<d(x,z)]
∧
      z=y ∨ [$\exists_{a\in adj(y)}$ a≠x ∧ d(a,z)<d(y,z)]
)
»
for 0≤x<#graph :

      for y∈adj(x) ∋ y>x :

            if removable(x,y) : untie(x,y)
»
```

## APPENDIX 2

List of terms for sample problem.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AA | AB | ABLE | ABSO | AC | ACTE | AGE | AL |
| ALS | AN | ANGE | ANGL | ANOT | ANS | APPR | AR |
| ARE | ARGU | ARLY | ARRA | ARY | AT | ATE | ATIO |
| AUTO | BASE | BE | BESS | BOUN | BY | CALC | CALL |
| CE | CH | CHAR | CHEC | CLOC | CODE | COEF | COMP |
| CONT | COPY | COSI | DATA | DATE | DAY | DD | DECO |
| DERI | DETE | DIME | DISP | DMPX | DOUB | DRAW | DUMP |
| ED | EIGE | EL | EM | END | ENT | ENTS | EQUA |
| ER | ERRO | ERS | ES | EST | ESTS | ETRI | EV |
| EVAL | EXCH | EXPO | FACT | FAST | FICI | FILE | FILM |
| FIND | FIRS | FLOA | FOR | FORM | FORT | FOUR | FROM |
| FUNC | GENE | GIVE | GRAL | GRAP | GRID | GT | HER |
| HERM | HTS | IAGO | IAL | IDE | IER | IFIE | IN |
| INDE | INE | INFO | ING | INTE | INTO | INVE | ION |
| IONS | IS | ISIO | ITIA | IX | KIND | LANG | LARG |
| LAY | LCM | LE | LEAS | LEX | LIBR | LINE | LOG |
| LUTE | LY | MAIN | MATR | MAXI | MENT | MICR | MM |
| MPOS | MUM | NAL | NAME | NATU | ND | NDIN | NE |
| NENT | NG | NOMI | NSIO | NT | NUMB | NV | NVAL |
| NVEC | OFIL | OLIC | OM | ON | ONAL | ONE | OPTI |
| OR | ORDE | ORS | OUTP | OUTS | OW | OXIM | PASS |
| PLOT | POIN | POLY | PORT | POSI | PREC | PROD | PROG |
| QUIC | RADI | RAL | RAM | RAN | RAND | RATE | RE |
| READ | REAL | RELA | REQU | RES | RETU | RMAT | RMIN |
| RN | RNS | ROOT | ROUT | RPOL | RS | RSE | RY |
| SCAL | SCM | SEAR | SECO | SELE | SERI | SET | SFOR |
| SINE | SING | SOLV | SORT | SOUR | SPEC | SPL | SQUA |
| SS | STAT | STOR | STRI | SYMB | SYMM | SYST | TABL |
| TAIN | TD | TE | TH | THAN | THE | TIME | TING |
| TION | TIVE | TO | TORS | TRAN | TRID | TS | TWO |
| TYPE | UATE | UCES | UCT | UES | ULAT | UNIT | US |
| USE | USER | USIN | UT | UTE | VALU | VARI | VATI |
| VE | VECT | WEIG | WHIC | WHOS | WIND | WITH | WORD |
| WRIT | ZERO | | | | | | |

## ACKNOWLEDGMENTS

The research reported in this dissertation was done while I was employed by the Los Alamos Scientific Laboratory. Support was provided by the U.S. Energy Research and Development Administration under contract with the Laboratory. I am grateful to both Administration and the Laboratory for their assistance.

Very special thanks are due to Professor Donald R. Morrison of the University of New Mexico for his advice, guidance, and encouragement while supervising this research. I would also like to thank the other members of my committee, Professors John W. Ulrich and Cleve B. Moler, for their personal and professional interest.

I wish to thank my friends and colleagues at the Laboratory for helpful comments and suggestions.

## REFERENCES

[1] Burd, William C. and Donald R. Morrison, "Lexicographic Correlation of Documents," Proc. of the NSF-CBMS Regional Research Conf. on Automatic Information Organization and Retrieval, The University of Missouri-Columbia. July 16-20, 1973. pp. 1-20.

[2] Dantzig, George B., "On the Shortest Route Through a Network," Management Science, Volume 6, no. 2, January, 1960.

[3] Dijkstra, E.W., "A Note on Two Problems in Connection With Graphs," Numer. Math., Volume 1, 1959, pp. 269-271.

[4] Harary, Frank and Edgar M. Palmer, Graphical Enumeration. Academic Press, New York, 1973.

[5] Knuth, Donald E., The Art of Computer Programming. Volume 1, Addison-Wesley Publishing Co., 1968.

[6] Morris, J.B. and Mark B. Wells, "The Specification of Program Flow in Madcap 6," SIGPLAN Notices, Volume 7, no. 11, November, 1972, pp. 28-35.

[7] Salton, Gerald, "A Vector Space Model for Automatic Indexing," Communciations of the ACM, Volume 18, no. 11, November, 1975, pp. 613-620.

[8] Salton, Gerald, Dynamic Information and Library Processing. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[9] Stein, Paul R., "Introductory Lectures on Graph Theory," unpublished.

[10] Wells, Mark B. and Fred L. Cornwell, "A Data Type Encapsulation Scheme Utilizing Base Language Operators," Proc. of Conference on Data: Abstraction, Definition and Structure, Volume 8, no. 2, 1976, pp. 170-178.

☆ U.S. GOVERNMENT PRINTING OFFICE 1977—777-018/18