THE STEINER PROBLEM IN GRAPHS

AND ITS APPLICATION TO PHYLOGENY

A dissertation presented by

M. L. Shore

in partial fulfilment of the requirements

for the degree of

Master of Arts in Computer Science

at

Massey University

November 1979

ABSTRACT

In this thesis we consider two problems, one in Graphy Theory and the second in Evolutionary Biology. The first problem, the Steiner Problem in Graphs, belongs to the class of problems known as NP-complete. That it belongs to this class is a measure of its difficulty; if a polynomial solution can be found for the Steiner Problem in Graphs, then by definition a polynomial solution will have been found for all other NP problems. For this problem we present a solution method based upon a branch and bound approach, and we show that it complements the current methods available for solving this problem, allowing solutions to the Steiner Problem in Graphs to be calculated for any graph of size $\leq$ 30 points in reasonable computing time.

The second problem is associated with evolution, and is an application of the Steiner Problem in Graphs. In trying to determine the evolutionary path from some common ancestor to existing species, a tree may be drawn to show these paths. This tree is called a phylogenetic tree, or phylogeny. There must be some criterion for deciding which of the many phylogenies that may be drawn most closely resembles the actual evolutionary changes. The criterion used in this thesis, used by many researchers in this area, is that of minimising the total number of changes in the phylogeny. It is shown that this problem is similar to the Steiner Problem in Graphs, and various solution methods based on heuristic graph theoretical techniques are discussed. Subsequent to this, a method of proving a phylogeny optimal is looked at, and an extension to this method presented which will allow larger phylogenies to be analysed.

ACKNOWLEDGEMENTS

The direct and indirect encouragement and support of many people has been greatly appreciated during the preparation of this thesis. My special thanks go to Drs Foulds, Gibbons, Hendy and Penny without whom this thesis would not have been possible.

Thanks also go to the Ministry of Defence EDP Directorate for their direct and indirect support throughout the duration of this thesis.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 The scope of this thesis

As its name suggests, this thesis examines two problems, one in graph theory and the other in evolutionary biology.

The Steiner Problem in Graphs (SPG) is a lesser known problem of combinatorial optimization, and although methods for its solution do exist, the size of the problem that can be solved still remains severely limited. The problem belongs to that class of problems known as NP-complete; by this we mean that the problem could be solved in polynomial time on a non-deterministic machine, and all other NP problems can be reduced to it in polynomial time. Thus if any NP-complete problem can be solved with a polynomial algorithm, then all NP problems are polynomially solvable, hence this class of problems are very difficult to solve. The thesis discusses two current methods of solving the problem - one by Hakimi [1971] and the other by Dreyfus and Wagner [1972]. There exists another solution method, that of Levin [1971], but this paper was discovered too late to be included in this thesis. However, from the expressions given for algorithm solution time, the method of Levin will, for non-trivial problems, always take longer than the method of Dreyfus and Wagner. The aim of this part of the thesis is to explain the techniques used in solving the SPG, in order to attempt similar methods to solve the phylogeny problem. At the same time, it is aimed to present a new algorithm for the SPG, and to show that this approach compares favourably with existing solution methods.

The concept of phylogeny - the building of a tree in which points represent species and lines represent evolutionary changes between species - is discussed

in the second part of this thesis. The various methods of building phylogentic trees are discussed briefly, and then a recent method, due to Foulds, et. al. [1978] of generating phylogenies using graph theoretical techniques is looked at in depth, and attempts made to use this and other similar techniques to allow completely automatic generation of trees. It is not known, at this time, whether the phylogeny problem is NP-complete. Then the latest approach used by Hendy et. al. [1979] is examined. This method involves two phases; the building of the tree, and proving that the tree is optimal. Apart from the automatic generation of trees, this part of the thesis aims to present an extension to the proof technique. Due to time constraints, this new approach was not able to be fully implemented and evaluated for inclusion in this thesis, but preliminary results are discussed.

Although a great deal of computer programming was involved in the preparation of this thesis, in evaluating and comparing algorithms for both the SPG and the phylogeny, the only program listing included as part of this thesis is the SPG branch and bound solution method algorithm program, and that is given as an appendix.

CHAPTER 2

THE STEINER PROBLEM IN GRAPHS

2.1 The Original Steiner Problem

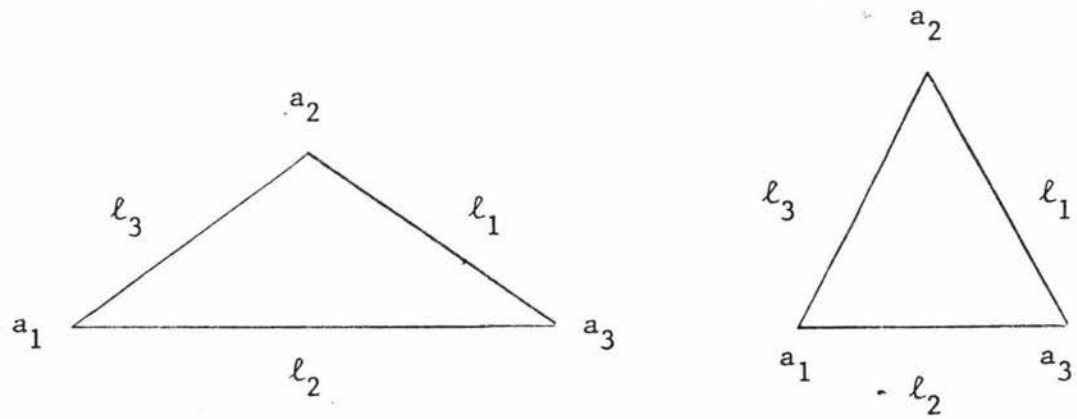
The Steiner Problem in Graphs is so named due to its similarity to an existing problem - the Steiner Problem (in geometry). This latter problem can be stated as follows. Given a set of points in a plane, construct a tree (or trees) which contain all the points and, optionally, any additional points, such that the total length of all lines in the tree is a minimum over all such trees. The interested reader is referred to Cockayne [1970] for further details of the Steiner problem.

A solution to this problem was proposed by Melzak [1961]. Gilbert and Pollak [1968] discussed the classification of various overall problem areas, the restrictions imposed in the particular areas leading to a large reduction in possible solutions. Melzak's method was further developed by Cockayne [1970]. However, the problem can be guaranteed to be solved for a relatively small number of points.

The Melzak solution to the problem involving three points follows. There are two and only two possible solution topologies as in Figure 2.1.

(a) For this solution, the tree required by the problem is $T = \{(a_1,a_2,a_3), (\ell_1,\ell_3)\}$.

(b) For this solution, a point, p, inside the triangle must be determined using the following procedure:



(a) one angle $> \frac{2\pi}{3}$ (b) all angles $\leq \frac{2\pi}{3}$

Figure 2.1

The Three-Point Topologies

(i) Create an equilateral triangle outside the triangle (ℓ_1,ℓ_2,ℓ_3), with one side being ℓ_2.

(ii) Find the circle that passes through points a_1,a_3 and the unnamed point of the equilateral triangle.

(iii) Draw a line from a_2 to the unnamed point of the equilateral triangle, and form the point, p, where this line intersects the circle inside the triangle. Connect this point with a_1,a_2,a_3. We now have Figure 2.2.

The point, p, is connected to points a_1,a_2,a_3 by lines which form three $2\pi/3$ angles, say ℓ_4,ℓ_5,ℓ_6. Then the solution tree to the problem is $T = \{(a_1,a_2,a_3,p), (\ell_4,\ell_5,\ell_6)\}$.

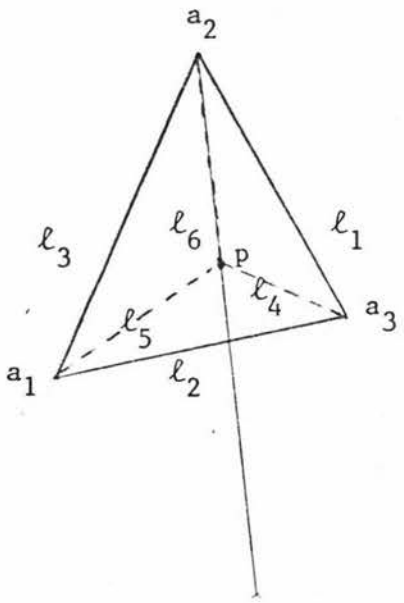


Figure 2.2

The Melzak Solution

For problems with more than three points, Melzak proposes:

(a) Generate all S-trees for the problem, where an S-tree is defined as having the properties:-

(i) It has vertices $a_1 \ldots a_n$ $s_1 \ldots s_k$, $0 \leq k \leq n-2$.

(ii) It is non self-intersecting.

(iii) Each point $s_1 \ldots s_k$ connects to three and only three other points.

(iv) Each point $a_1 \ldots a_n$ connects to up to three other points.

(b) The solution is the minimum cost S-tree.

This involves the examination of

$$\sum_{k=0}^{n-2} N(n,k)(n+k)^{n+k-2} \qquad \text{S-trees}$$

$$\text{where } N(n,k) \leq \frac{[(n+k-2)!\ (n+k-3)!]\ (\frac{n+k-1}{3})}{(n-1)!\ (n-2)!}$$

Cockayne has developed this procedure into a feasible strategy for $n < 30$. His method is as follows:

(a) Form a set of all divisions of the original (necessary) points.

(b) Discard divisions that do not correspond to a Steiner Minimising Tree over the set of all points.

(c) For each division left, construct the set of all trees that are produced by forming the union of all full Steiner Trees of the component divisions.

(d) The solution is the entry in this set with the minimum total length.

2.2 The Steiner Problem in Graphs

A graph G is an ordered pair (P,L), where P is a non empty set of elements called points, and L is a set of unordered pairs of points called lines. Each line $\ell_k \in L$ is expressed as $\{p_i,p_j\}$, and is said to be incident with p_i and p_j. Both p_i and p_j are said to be adjacent to the line $\{p_i,p_j\}$. A graph may be weighted, meaning that each line has a weight associated with it, by the function $w:L \to R$, where R is the set of real numbers. For brevity the weight of the line $\{p_i,p_j\}$ will from now on be denoted w_{ij}. A path between points p_i and p_j in a graph G=(P,L) is a sequence of the form

$$<p_i,\{p_i,p_{k_1}\},p_{k_1},\{p_{k_1},p_{k_2}\},p_{k_2},\ldots\ldots p_{k_n},\{p_{k_n},p_j\},p_j>$$

where p_{k_q} ,q=1...n are all distinct points in P, and the pairs are all distinct lines in L.

A subset $S \subseteq P$ of the points in the graph is said to be connected if there exists at least one path between every pair of points in S. If, in a graph G=(P,L), P is connected, then the graph G is said to be connected.

Now we can state the Steiner Problem in Graphs (SPG).

Given a connected, weighted graph G=(P,L) and a non empty subset S of P, we must find a subset T of L that has the following properties:

(a) The points in S are connected by paths comprising only lines in T.

(b) Σw_{ij} is a minimum among all sets T satisfying (a).

For the purposes of this problem, we will henceforth consider weighted graphs to have non negative weights.

It can be seen that this problem is similar to the original Steiner problem and, as will be seen, there are similar solution methods.

During the rest of the SPG discussion, the elements of set S will be called the essential points, and the elements of set P/S the inessential points. The subset T which produces the minimum total weight, which is the solution to the problem, will be called the Steiner Minimal Spanning Tree (SMST). There are two special cases of the SPG which are as follows:

(a) S consists of only two points from P.

The problem is reduced to that of finding the shortest path between two

points in a graph. An efficient algorithm for this problem is given by Dijkstra [1959].

(b) S consists of all the points in P.

The problem now becomes that of finding a Minimal Spanning Tree (MST) for a graph. Algorithms for this problem have been given (Kruskal [1956], Prim [1957], Dijkstra [1959]). Kevin and Whitney [1971] give an efficient implementation of the Dijkstra algorithm. A description of this implementation is given later when discussing the Hakimi method (see section 2.3).

The general SPG has been studied by Hakimi [1971], producing a solution similar in concept to the Melzak method for the original Steiner problem. Levin [1971] and Dreyfus and Wagner [1972] developed similar methods, both resembling the Cockayne method for solving the original problem. Also a new method is discussed which uses a branch and bound technique. This is the major work of this chapter of the thesis.

2.3 The Hakimi Method

Hakimi proposed that a minimal spanning tree be calculated for each of the possible subsets of points, from just the set of essential points through to the complete set of points. The Kevin and Whitney implementation of Dijkstra's algorithm performs the individual spanning tree calculations. For a graph $G=(P,L)$ with $S \subset P$, and $n = |P|$, $m = |s|$, then there will be $\sum_{k=0}^{m} \binom{m}{k}$ MST calculations The Hakimi algorithm is as follows:

```
Begin
     min_t := ∞
     For all D*⊆P/S do
     Begin
          cost := KEVANDWHIT(SUD*);
          If cost < min_t then
          Begin
               min_t := cost;
               solutiontree := TREE;
          End
     End
End;
```

So this method of solving the SPG examines a number of graphs (i.e. the graph containing only points $p_1 \ldots p_m$, the graph containing points $p_1 \ldots p_m$ and one p_j where j=m+1...n, the graph containing the points $p_1 \ldots p_m$ and two p_j where p_j=m+1...n, the graph ..... containing all points in P). Then the graph which yields the least cost MST is selected, and its associated MST becomes the SMST.

The Dijkstra MST algorithm grows a MST by successively adjoining the nearest remaining point to a partially formed tree until all points of the graph are included in the tree. The Kevin and Whitney implementation is particularly efficient due to its method of dynamically keeping track of the minimum distance to the current subtree from all points outside the tree.

The data structures used in the Kevin and Whitney algorithm are as follows:

n	number of points in the graph.
DM	distance matrix between points.
TREE	array holding lines currently in partial MST.
NIT	array of points not in the tree.
nitp	number of entries in NIT.

JI — array holding the point in the tree closest to the point with the same array index in NIT.

UI — length of the line defined by NIT and JI.

kp — point last added to the tree.

cost — cost of TREE so far.

The algorithm is as follows:

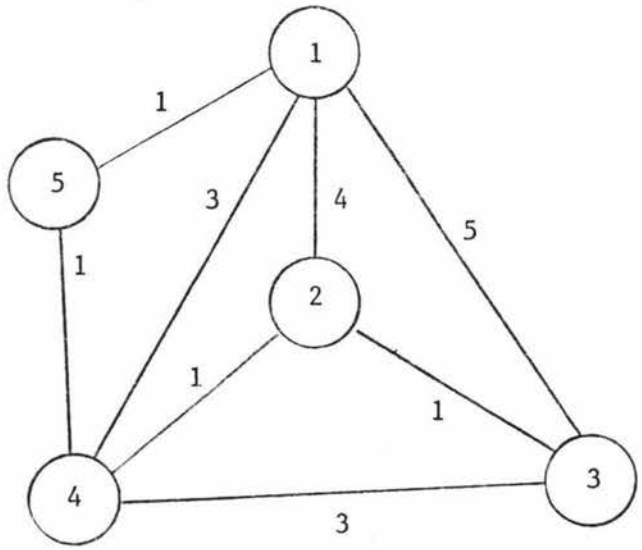
```
Begin
     initialize;
     while ∃ point ∉ tree do
     Begin
          for i := 1 step 1 until nitp do
          if DM(NIT(i),kp)<UI(i) then UPDATE-UI;
          min := ∞;
          for i :=  1 step 1 until nitp do
          if UI(i)<min then
          Begin
               min := UI(i);
               j := i;
          End;
          PUTINTOTREE(NIT(j));
          UPDATEPOINTSTATUS;
          cost := cost+UI(j);
          KP := NIT(j);
          nitp := nitp-1;
     End
End;
```

The expected time for solving a given problem by Hakimis method will be related to the expression

$$0\left(\sum_{k=0}^{m}\binom{m}{k}(n-m+k)^2\right)$$

As (n-m) increases, the method becomes very unwieldy.

Numerical example of the Hakimi Method



$$DM = \begin{bmatrix} 0 & 4 & 5 & 3 & 1 \\ 4 & 0 & 1 & 1 & \infty \\ 5 & 1 & 0 & 3 & \infty \\ 3 & 1 & 3 & 0 & 1 \\ 1 & \infty & \infty & 1 & 0 \end{bmatrix}$$

3 essential points

5 total points

Step 1

$D^* := \{1\ 2\ 3\}$, $min_t := \infty$

step 1.1

kp=3, NIT=[1 2], UI=[5 1], JI=[3 3]
min := ∞
UI(1)=5 < ∞, so min := 5
j := 1
UI(2)=1 < 5, so min := 1
j := 2
TREE(1,1) := 2, TREE(2,1) := 3
cost := 1

step 1.2

kp=2, NIT=[1], UI=[5], JI=[3]
DM(1,2) < UI(1), so UI := [4]
JI := [2]
min := ∞
UI(1)=4 < ∞, so min := 4
j := 1
TREE(1,2) := 1, TREE(2,2) := 1
cost := 1+4=5

Step 2

cost = 5 < ∞ so min_t := 5
solutiontree := [{2,3}, {1,2}]
D^* = [1 2 3 4]

step 2.1

kp=4, NIT=[1 2 3] UI=[3 1 3] JI=[4 4 4]
min := ∞
UI(1)=3 < ∞ so min := 3
j := 1
UI(2)=1 < 3 so min := 1
j := 2
UI(3)=3 ≮ 1
TREE(1,1) := 2 TREE(1,2) := 4
cost := 1

step 2.2

kp=2 NIT=[1 3] UI=[3 3] JI=[4 4]
DM(1,2) $\not<$ UI(1)
DM(3,2)=1 < UI(2)=3 so UI(2) := 1
JI(2) := 2
min := ∞
UI(1)=3 < ∞ so min := 3
j := 1
UI(2)=1 < 3 so min := 1
j := 2
TREE(1,2) := 3 TREE(2,2) := 2
cost := 1+1=2

step 2.3

kp=3 NIT=[1] UI=[3] JI=[4]
DM(1,3)=5 $\not<$ UI(1)=3
min := ∞
UI(1)=3 < ∞ so min := 3
j := 1
TREE(1,3) := 1 TREE(2,3) := 4
cost := 2+3=5

Step 3

cost = 5 $\not<$ 5
D^* = [1 2 3 5]

step 3.1

kp=5 NIT=[1 2 3] UI=[1 ∞ ∞] JI=[5 5 5]
min := ∞
UI(1)=1 < ∞ min := 1
j := 1
UI(2) = ∞ $\not<$ 1
UI(3) = ∞ $\not<$ 1
TREE(1,1) := 1 TREE(2,1) := 5
cost := 1

step 3.2

kp=1 NIT=[3 2] UI=[∞ ∞] JI=[5 5]
DM(1,3)=5 < UI(1)=∞ so UI(1) := 5
JI(1) := 1
DM(1,2)=4 < UI(2)=∞ so UI(2) := 4
JI(2) := 1
min := ∞
UI(1)=5 < ∞ so min := 5
j := 1
UI(2)=4 < 5 so min := 4
j := 2
TREE(1,2) := 2 TREE(2,2) := 1
cost := 1+4=5

step 3.3

kp=2 NIT=[3] UI=[5] JI=[3]
DM(2,3)=1 < UI(1)=5 so UI(1)=1
JI(1)=2
min := ∞
UI(1)=1 < ∞ so min := 1
j := 1
TREE(1,3) := 3 TREE(2,3) := 2
cost := 5+1=6

Step 4

cost = 6 $\not<$ 5

D^* = [1 2 3 4 5]

step 4.1

kp=5 NIT=[1 2 3 4] UI=[1 ∞ ∞] JI=[5 5 5 5]

min := ∞

UI(1)=1 < ∞ so min := 1
j := 1

UI(2) = ∞ $\not<$ 1

UI(3) = ∞ $\not<$ 1

UI(4) = ∞ $\not<$ 1

TREE(1,1) := 1 TREE(2,1) := 5

cost := 1

step 4.2

kp=1 NIT=[4 2 3] UI=[1 ∞ ∞] JI=[5 5 5]

DM(1,4) = 3 $\not<$ 1

DM(1,2)=4 < ∞ so UI(2) := 4
JI(2) := 1

DM(1,3)=5 < ∞ so UI(3) := 5
JI(3) := 1

min := ∞

UI(1)=1 < ∞ so min := 1
j := 1

UI(2) = 4 $\not<$ 1

UI(3) = 5 $\not<$ 1

TREE(1,2) := 4 TREE(2,2) := 5

cost := 1+1=2

step 4.3

kp=4 NIT=[3 2] UI=[5 4] JI=[1 1]

DM(4,3)=3 < 5 so UI(1) := 3
JI(1) := 4

DM(4,2)=1 < 4 so UI(2) := 1
JI(2) := 4

min := ∞

UI(1)=3 < ∞ so min := 3
j := 1

UI(2)=1 < 3 so min := 1
j := 2

TREE(1,3) := 2 TREE(2,3) := 4

cost := 2+1=3

step 4.4

kp=2 NIT=[3] UI=[3] JI=[4]

DM(2,3)=1 < 3 so UI(1) := 1
JI(1) := 1

min := ∞

UI(1)=1 < ∞ so min := 1
j := 1

TREE(1,4) := 3 TREE(2,4) := 2

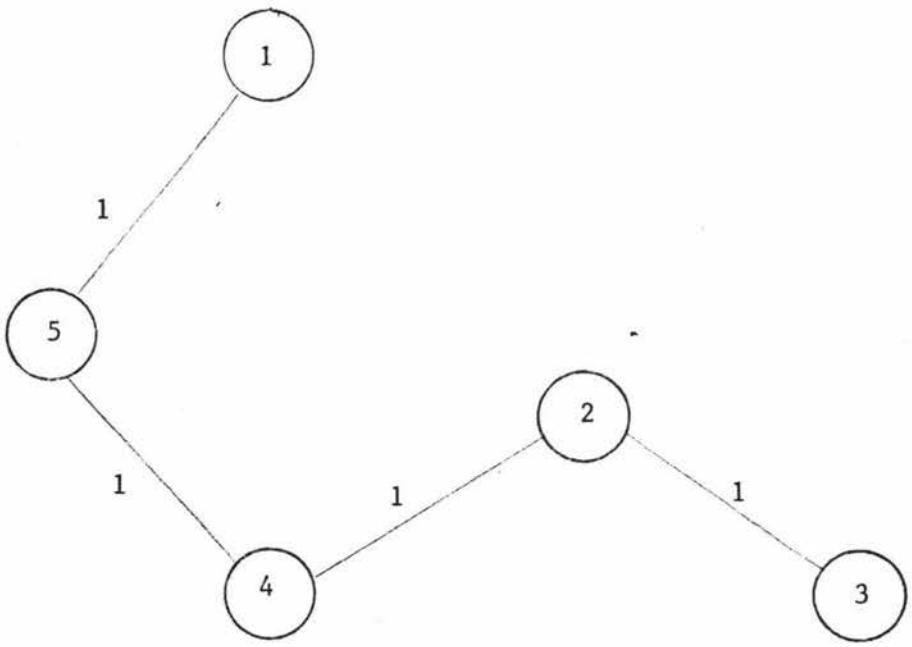
cost := 3+1=4

Step 5

cost = 4 < 5 so min_t := 4

solutiontree := [{1,5}, {4,5}, {2,4}, {3,2}]

The algorithm then terminates, with the SMST as the MST in step 4,



with a cost of 4.

2.4 The Dreyfus and Wagner Method

The Dreyfus and Wagner solution method breaks the problem down into subproblems, and each of these subproblems themselves into subproblems etc., until the subproblem can be solved directly from a matrix of shortest paths between points. Consider a typical SMST as in Figure 2.3.

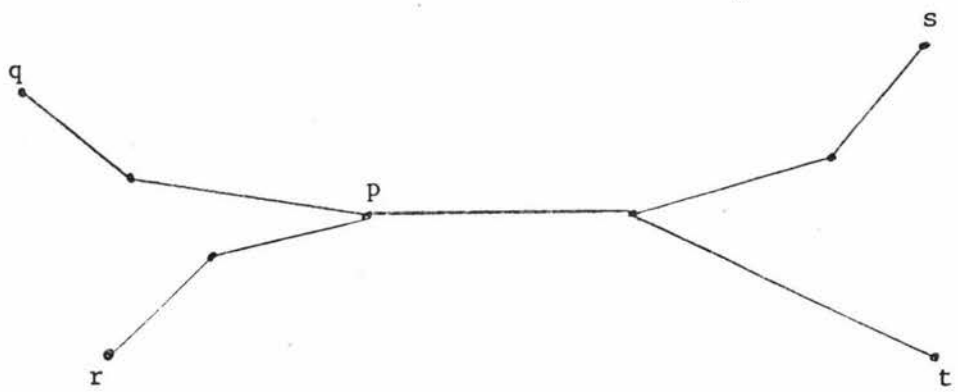


Figure 2.3

A Typical SMST

The solution can be divided into three subproblems, whose union forms this SMST, by the following method:

(a) Choose a point, say q

(b) Find a junction point, here p

(c) Split the tree into three subtrees from this point p, one subtree being the path q to p

So here we have Figure 2.4.

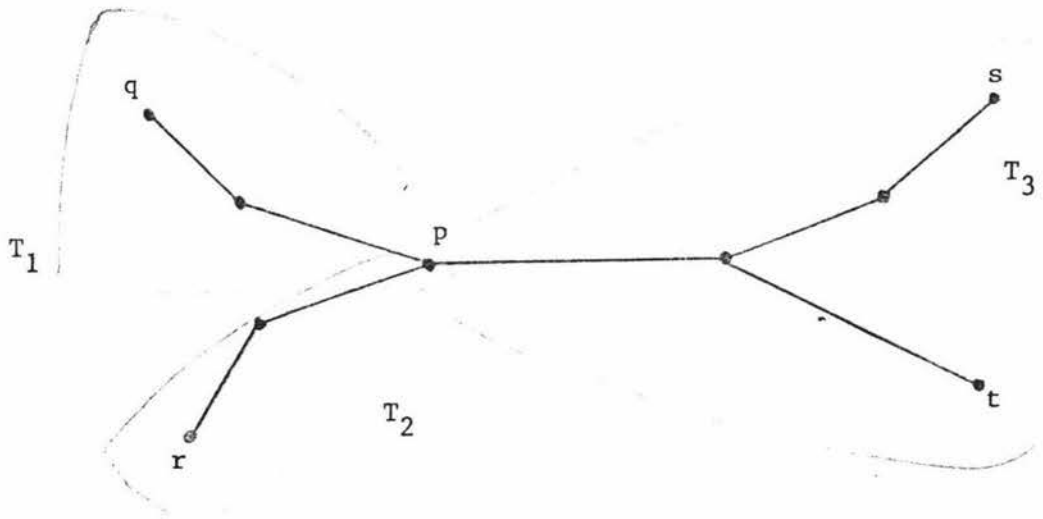


Figure 2.4

Three Subtrees

We have two subproblems T_2, T_3 and a shortest path T_1. Now each subproblem is similarly subdivided as in Figure 2.5.

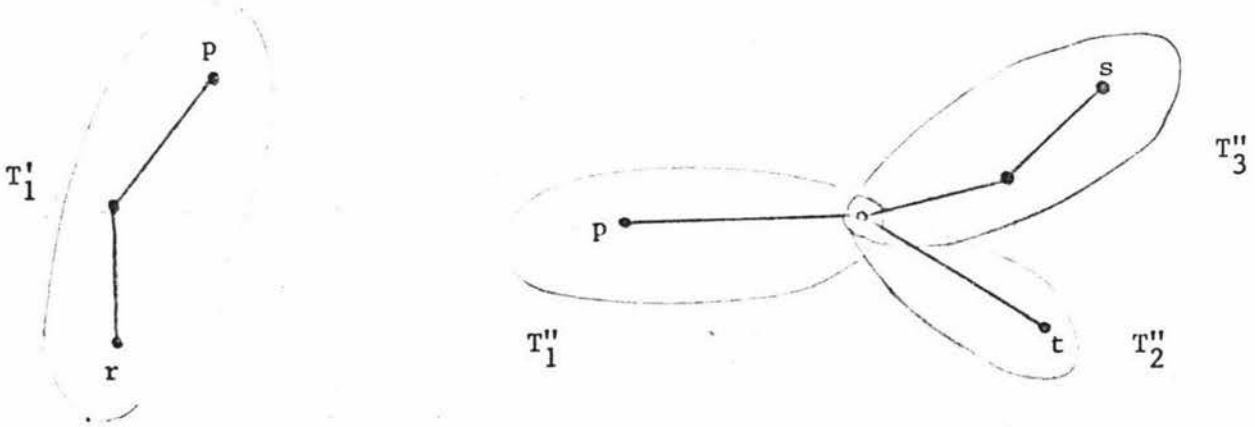


Figure 2.5

Subproblem Subdivision

and here we have all subproblems as shortest paths.

The above division of the problem into subproblems is called the Optimal Decomposition Property, and the method of division can be formally stated:-

Let $G = (P,L)$

$S \subset P$, $T \subset L$ such that T is the SMST

$q \in S$

For $|S| \geq 3$ $\exists$ $p \in P$ and $D \subset T$ such that

D is a proper subset of $T-\{pq\}$, and is non empty

T consists of 3 disjoint subsets T_1 T_2 T_3

T_1 connects p q

T_2 connects pUD

T_3 connects $pU(T-D-T_1)$

and furthermore T_1 T_2 T_3 are all MSTs over their respective sets of points.

The Dreyfus and Wagner method calculates the cost of the SMST by taking the minimum cost of all possible assignments of points to T_1 T_2 T_3, at each level of decomposition.

A formal description of the Dreyfus and Wagner algorithm is as follows:

A array of shortest path weights between all points

n $|P|$

m $|S|$

D (2,3,...,m)

```
Begin
     cost := DECOMPOSE(1,D)
End;
```

```
INTEGER PROCEDURE DECOMPOSE(q,D);
Begin
     if |D| = 1 then min := A [q,D[1]]
     else
     Begin
          min := ∞;
          for p := 1 step 1 until n do
          Begin
               junctioncost := JUNCTION(p,D);
               if A [q,p] + junctioncost < min then min :=
                                              A [q,p] + junctioncost
          End
     End;
     DECOMPOSE := min;
End;

INTEGER PROCEDURE JUNCTION(p,D);
Begin
     if |D| = 2 then min := A [p,D[1]] + A [p,D[2]]
     else
     Begin
          i := STOREINDEX(p,D);
          if DECOMPVAL[i] # 0 then min := DECOMVAL[i]
          else
          Begin
               min := ∞;
               for all D* ⊂ D do
               Begin
                    T2 := DECOMPOSE(p,D*);
                    T3 := DECOMPOSE(p,D/D*);
                    if T2 + T3 < min then min := T2 + T3
               End
          End;
          if DECOMPVAL[i] = 0 then DECOMPVAL[i] := min
     End;
     JUNCTION := min
End;
```

STOREINDEX is some hashing algorithm which, for a point p and subset D, defines the entry in array DECOMPVAL which corresponds to the cost of the decomposition of this subset.

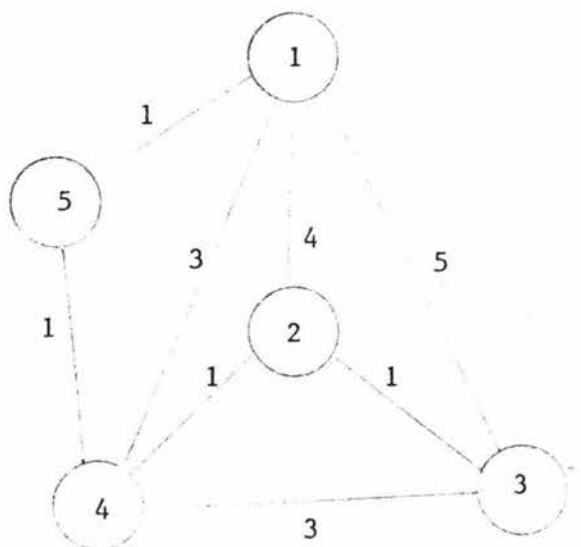
So far, then, we have deduced the cost of the SMST. In order to determine the tree itself, Dreyfus and Wagner propose that the values of p and D that minimise DECOMPOSE be stored, and the tree constructed by processing this information in the reverse order of that of the cost algorithm.

The given timing formula for the Dreyfus and Wagner method is, for a graph G = (P,L), S ⊂ P being a set of essential points, |P| = n, |S| = m :

$$\frac{n^3}{2} + n^2(2^{m-1}-m-1) + \frac{n(3^{m-1}-2^m+3)}{2}$$

and so it can be seen that the algorithm is dependent to a degree on n and, to a far greater degree, on m. Also, in order to maintain the array DECOMPVAL, and thus to be able to use previously calculated results, requires $O(n2^{m-1})$ storage elements. Hence there is serious limit on the value of m.

Numerical example of the Dreyfus and Wagner Method



$$A \quad \begin{bmatrix} 0 & 3 & 4 & 2 & 1 \\ 3 & 0 & 1 & 1 & 2 \\ 4 & 1 & 0 & 2 & 3 \\ 2 & 1 & 2 & 0 & 1 \\ 1 & 2 & 3 & 1 & 0 \end{bmatrix} \quad \text{(shortest paths)}$$

4 essential points

5 total points

DECOMPOSE(1,(2 3 4))
|D| > 1
min := ∞
p := 1
JUNTION(1,(2 3 4))
no stored index so calculate
a) D* = [2], D/D* = [3 4]
T_2 := DECOMPOSE(1,(2))
A[1,2] = 3
T_3 := DECOMPOSE(1,(3 4))
|D| > 1
min := ∞
p := 1
JUNCTION(1,(3 4))
A[1,3] + A[1,4] = 6
min := 6 + A[1,1] = 6
p := 2

JUNCTION(2,(3 4))
A[2,3] + A[2,4] = 2
2 + A[1,2] = 5 < 6, so min := 5
p := 3
JUNCTION(3,(3 4))
A[3,3] + A[3,4] = 2
2 + A[1,3] = 6 ≮ min
p := 4
JUNCTION(4,(3 4))
A[4,3] + A[4,4] = 2
2 + A[1,4] = 4 < min, so min := 4
p := 5
JUNCTION(5,(3 4))
A[5,3] + A[5,4] = 4
4 + A[1,5] = 5 > min
now store DECOMPVAL(1,(3 4)) = 4

So T_3 := 4

$T_2 + T_3 = 7 <$ min, so min := 7

b) D* = [2 3], D/D* = [4]

T_2 := DECOMPOSE(1,(2 3))
|D| > 1
min := ∞
p := 1
JUNCTION(1,(2 3))
A[1,2] + A[1,3] = 7
7 + A[1,1] = 7 < min, so min := 7
p := 2
JUNCTION(2,(2 3))
A[2,2] + A[2,3] = 1
1 + A[1,2] = 4 < min, so min := 4
p := 3
JUNCTION(3(2 3))
A[3,2] + A[3,3] = 1
1 + A[1,3] = 5 > min
p := 4
JUNCTION(4,(2 3))
A[4,2] + A[4,3] = 3
3 + A[1,4] = 5 > min
p := 5
JUNCTION(5,(2 3))
A[5,2] + A[5,3] = 5
5 + A[1,5] = 6 > min
so store DECOMPVAL(1,(2 3)) = 4

Hence T_2 := 4

T_3 := DECOMPOSE(1,(4))
|D| = 1
A[1,4] = 2

$T_2 + T_3 = 6 <$ min, so min := 6

c) D* = [2 4] D/D* = [3]

T_2 := DECOMPOSE(1,(2 4))
|D| > 1
min := ∞
p := 1
JUNCTION(1,(2 4))
A[1,2] + A[1,4] = 5
5 + A[1,1] = 5 < min, so min := 5
p := 2

JUNCTION(2,(2 4))
A[2,2] + A[2,4] = 1
1 + A[1,2] = 4 < min, so min := 4
p := 3
JUNCTION(3,(2 4))
A[3,2] + A[3,4] = 3
3 + A[1,3] = 7 > min
p := 4
JUNCTION(4,(2 4))
A[4,2] + A[4,4] = 1
1 + A[1,4] = 3 < min, so min := 3
p := 5
JUNCTION(5,(2 4))
A[5,2] + A[5,4] = 3
3 + A[1,5] = 4 > min
so store DECOMPVAL(1,(2 4)) = 3

So T_2 := 3

T_3 := DECOMPOSE(1,(3))
|D| = 1
A[1,3] = 4

$T_2 + T_3 = 7 > \min$

So JUNCTION(1,(2 3 4)) = 6
6 + A[1,1] = 6 < min, so min := 6
p := 6
JUNCTION(2,(2 3 4))

and so on

p := 4
JUNCTION(4,(2 3 4))
no stored index, so calculate
a) D* = [2] D/D* = [3 4]

T_2 := DECOMPOSE(4,(2))
|D| = 1
A[4,2] = 1

T_3 := DECOMPOSE(4,(3 4))

So T_3 := 2

$T_2 + T_3 = 3 < \min$, so min := 3

b) D* = [2 3] D/D* = [4]

T_2 := DECOMPOSE(4,(2 3))

So T_2 := 2

T_3 := DECOMPOSE(4,(4))
|D| = 1
A[4,4] = 0

$T_2 + T_3 = 2 < \min$, so min := 2

c) D* = [2 4] D/D* = [3]

T_2 := DECOMPOSE(4,(2 4))

So T_2 := 1

T_3 := DECOMPOSE(4,(3))
|D| = 1
A[4,3] = 2

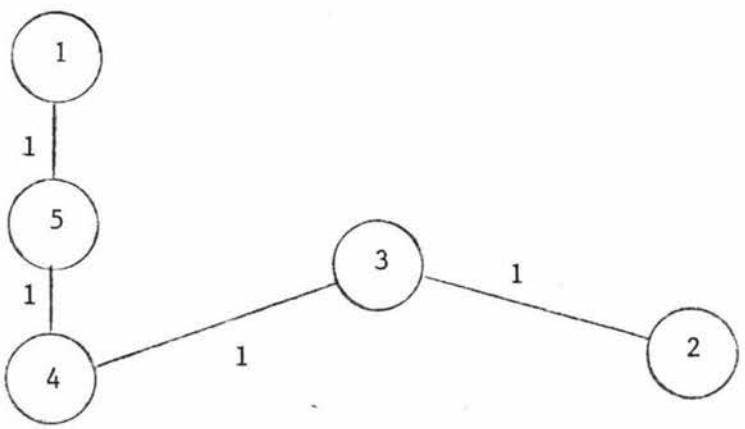
$T_2 + T_3 = 3 > \min$

So JUNCTION(4,(2 3 4)) = 2
2 + A[1,4] = 4 < min, so min := 4
p := 5
JUNCTION(5,(2 3 4))

So JUNCTION(5,(2 3 4)) = 5
5 + A[1,5] = 6 > min

Hence min = 4, and so the cost of the SMST is 4.

This occurred at JUNCTION(4,(2 3 4)), the set (1 2 3 4) therefore being optimally decomposed into the three subsets (1 4), (4 3 2) and (4). The shortest path from point 1 to point 4 is < 1, {1,5}, 5, {5,4}, 4 >. So the tree is



2.5 A new approach using Branch and Bound techniques

2.5.1 The Branch and Bound Methodology

The branch and bound methodology was first suggested by Land and Doig [1960] as a method of solving problems of mixed integer programming. Beale [1968] states that, properly organised on a powerful computer, the method has proved to be very effective in solving many problems.

The method is as follows. The original problem is divided into a number of subproblems by the use of some constraint(s) on a problem variable. Then a subproblem is selected, the criteria for this selection depending on the problem, and examined to see whether it represents a feasible solution to the original problem. If so, this feasible solution is compared with the best feasible solution to date (this is called the incumbent solution), and if it represents a better solution, then it replaces the incumbent solution. If not, then there is no point in further dividing this subproblem (any further subproblem will retain the characteristics of this feasible solution and will not bring about a lower cost feasible solution), and so it has been completely examined, and we say the subproblem is fathomed. If this subproblem does not represent a feasible solution to the original problem, then a lower bound on the value of any feasible solution cost that may be discovered within this subproblem is calculated. If this lower bound is less than the cost of the incumbent solution, then a feasible solution of lower cost than the incumbent may possibly be produced from this subproblem, and so it is divided into further subproblems. If the lower bound on this subproblem is greater than or equal to the cost of the incumbent, then an improved incumbent solution will not be discovered in this subproblem, and so the subproblem is considered fathomed, and discarded. Now a new subproblem is selected from the set of current unfathomed subproblems and the process repeats until all subproblems have been examined. The incumbent solution then represents the optimal solution to the original problem.

This method, in the worst case, examines all combinations of variables that could occur in a solution to the problem; its usefulness lies in being able to discard large subsets of solutions without having to specifically examine any solution in the subset.

For use in the SPG, the branch and bound method uses the inclusion/exclusion of a line in the SMST as its problem variable for dividing a (sub)problem into

further subproblems. The lower bound on a subproblem represents a value less than or equal to the cost of any of the feasible solutions available in that subproblem, the value being the cost of a SMST. This means, then, that the branch and bound solution to the SPG may be represented as a binary tree, with each subordinate node in the tree representing its superior solution but with a line either included in or excluded from the subset of solutions that it represents. For an example see Figure 2.6.

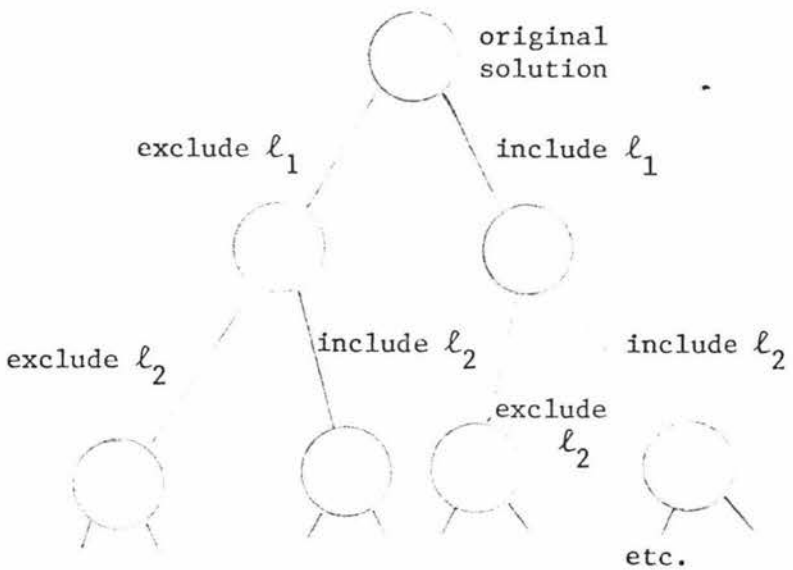


Figure 2.6

A Branch and Bound Decision Tree

Obviously, a pendant node in this representation of the solution to the SPG is fathomed, and denotes either a feasible solution or a class of solutions which cannot produce an optimal solution and have hence been discarded.

2.5.2 Solving the SPG using Branch and Bound

2.5.2.1 Introduction

The optimal (least weight) solution to any SPG will be a tree. By definition, a solution must be connected, and hence any solution that is not a tree must contain a cycle(s). Now a line in this cycle may be removed without disconnecting the solution, and a new solution of lower weight (i.e. less the weight of the cycle line) is produced - but the optimal solution was least weight, and hence this would be a contradiction. Branch and bound enumeration can be used to identify this optimal solution tree. The procedure, in common with all branch and bound techniques, examines in a systematic way a series of partial solutions to the problem. Each such solution is analysed by the use of lower and upper bounds, to discover whether it can be part of a minimal solution.

Let G = (P,L) be a graph in which P is the set of points and L is the set of lines connecting points in P. Also let $S \subseteq P$ be the set of essential points. Let $|P| = n$, $|S| = m$, and refer to the points in S as $p_1 \ldots p_m$ and those in P/S as $p_{m+1} \ldots p_n$. This ordering of points may be done without any loss of generality. A matrix of line weights, $W = \{w_{ij}\}_{n \times n}$ is formed, where $w_{ij} = \infty$ if $i = j$ or $\{p_i p_j\} \notin L$. At times during the course of the procedure a line $\{p_i p_j\}$ may be temporarily excluded from consideration. In this case temporarily set $w_{ij} = w_{ji} = \infty$.

Each partial solution examined is characterised by having a specified set of included lines, and another set of excluded lines, where either set may be empty. The set of included lines for a partial solution forms a set of connected components; a component being an essential component if it contains at least one essential point, and an inessential component otherwise. For a partial solution to be feasible, there must be only one essential component, i.e. all points in S being connected by the set of included lines. As a line $\{p_i p_j\}$ is

added to the set of included lines, the components containing p_i and p_j will be merged to form one component. When a line $\{p_i p_j\}$ is excluded, the components remain unchanged. However, the w_{ij} entry in the weight matrix is set to ∞.

2.5.2.2 Node selection in the branch and bound decision tree

The "quickest feasible solution" or "branch to the right" strategy was adopted. As will be seen later, the branching strategy generates a binary tree. Of the two nodes created at each decision point, one corresponds to excluding a line from consideration, the other to adding that line to the set of included lines for a partial solution. The latter node is always selected first. This leads to an initial examination of successive partial solutions of accepted lines which is eventually fathomed. Backtracking then occurs, to either uncover the optimal solution (which may be the one initially fathomed), or prove no such solution exists (which could occur if G was disconnected).

2.5.2.3 The branching method

A node is fathomed as follows. A line is selected and two further partial solutions are produced. With each line is associated a penalty for not including it in the set of included lines. The line with the largest penalty is selected for branching. In the case where there is more than one line with the largest penalty, then the line that was selected as the penalty line for the essential point with the minimum point index is chosen. This can be formally stated as follows:

(a) Calculate a penalty vector $T = [t_1 \ldots t_m]$ by:

(i) $w_i^* = \min_{1 \le j \le n} \{w_{ij}\}$, and k_i the value of j producing w_i^*

(ii) $w_i^+ = \min_{\substack{1 \le j \le n \\ j \neq k_i}} \{w_{ij}\}$

(iii) $t_i = w_i^+ - w_i^*$

(b) Now let $t_r = \max_{1 \le i \le m} \{t_i\}$

Then the line to branch on is $\{p_r \; p_{k_r}\}$.

As stated previously, two new nodes are created, emanating from this current node (say N). One represents the partial solution as for N, but with the added line $\{p_r \; p_{k_r}\}$, and the two components containing p_r and p_{k_r} merged; the other partial solution is as for N except that $\{p_r \; p_{k_r}\}$ is excluded from consideration.

A numerical example of this branching process follows.

$$W = \begin{bmatrix} \infty & 1 & 2 & 1 & 3 \\ 1 & \infty & 3 & 2 & 3 \\ 2 & 3 & \infty & 1 & 4 \\ 1 & 2 & 1 & \infty & 5 \\ 3 & 3 & 4 & 5 & \infty \end{bmatrix}$$

T = [(1-1),(2-1),(2-1),(1-1),(3-3)]

= [0,1,1,0,0]

and K = [2 1 4 1 1]

Now $t_r = t_2 = 1$, and so we branch on $\{p_r \; p_{k_r}\} = \{p_2 \; p_1\}$

So two new nodes are formed with one adding $\{p_2 \; p_1\}$ to the set of included lines, the other adding $\{p_2 \; p_1\}$ to the set of excluded lines.

2.5.2.4 The bounding method

In order to produce a bound for a new partial solution, the weight matrix W must be adjusted.

(a) If the new partial solution was produced by adding a line $\{p_x\ p_y\}$ to the set of excluded lines, then temporarily set $w_{xy} = w_{yx} = \infty$

(b) If the new partial solution was produced by adding a line to the set of included lines, then components C_x and C_y to which the points adjacent to the line $\{p_x\ p_y\}$ belong are combined. W is then transformed to become a square matrix W' with one less row and one less column than the square matrix W, and in which each row/column corresponds once again to a unique component.

So for $W' = \{w'_{ij}\}_{n \times n}$

$$w'_{ij} = w_{ij} \quad \text{if } i \neq x,y \quad j \neq x,y$$

if $x \leq m$, $w'_{ix} = w'_{xi} = \min \{w_{xi}, w_{yi}\}$ $i = 1....n,\ i \neq y$

else $w'_{iy} = w'_{yi} = \min \{w_{yi}, w_{xi}\}$ $i = 1....n,\ i \neq x$

It is necessary to reduce by one the subscript of entries, in rows below, and columns to the right of, max(x,y).

Note that in the actual implementation of the algorithm, the weight matrix W is not reduced in this fashion, because:

(a) This would lead to a proliferation of matrices, and hence to storage problems.

(b) It would be difficult to unravel solutions as the original point labels would have been lost in component merging.

Instead, one copy of the original weight matrix is retained, and the components are defined in a component vector $C = \{c_i\}$ $1 \times n$ where c_i is the number of the component which holds the point p_i.

Let us now return to the problem of calculating a lower bound on a partial solution. Given a graph $G = (P,L)$, let z^* be the cost of the minimal solution, and let

$$b = \sum_{i=1}^{m} \min_{j=1\ldots n} \{w_{ij}\}$$

$$c = \left(\sum_{i=1}^{m} \min_{j=1\ldots m} \{w_{ij}\} \right) - \left(\min_{\substack{i=1\ldots m \\ j=1\ldots m}} \{w_{ij}\} \right)$$

Then we have:

Theorem 2.1

$$z^* \geq \min(b,c)$$

Proof

Consider a minimal tree T^* with length z^* spanning S. Suppose $T^* = (P^*,L^*)$ where $S \subseteq P^* \subseteq P$ and $L^* \subseteq L$. Given any $p_t \in P^*$, T^* can be represented as the ordered triple (P_t,L^*,p_t), where $P_t \cup p_t = P^*$, and there is a one-to-one correspondence

$h:P_t \rightarrow L^*$ such that p_i is incident with $h(p_i)$ for all $p_i \in P_t$. Now either $P^*/S \neq 0$ or $P^*=S$.

Case I

$P^*/S \neq 0$, i.e. $\exists\ k,\ m < k \leq n \ ||\ p_k \in P^*/S$

Let $p_t = p_k$ so $S \subseteq P_t$ as $p_k \in S$

Thus $\{h(p_i):p_i \in S\} \subseteq L^*$

$$\therefore w_{h(p_i)} \geq \min_{\{p_i p_j\} \in L}(w_{ij})$$

where $w_{h(p_i)}$ = weight of line $h(p_i)$

$$\therefore z^* \geq \sum_{p_i \in P_t} w_{h(p_i)} \geq \sum_{\substack{p_i \in P_t \\ \{p_i p_j\} \in L}} \min(w_{ij})$$

$$\geq \sum_{\substack{p_i \in S \\ \{p_i p_j\} \in L}} \min(w_{ij})$$

$$= \sum_{i=1}^{m} \min_{j=1\ldots n}(w_{ij}) = b$$

Case II

$P^* = S$

Given any $p_t \in P^*$, $\{h(p_i):p_i \in P_t\} = L^*$

Let $w_{gd} = \min_{\substack{i=1\ldots m \\ j=1\ldots m}}(w_{ij})$

Let $p_t = p_g$

Now $w_{h(p_i)} \geq \min(w_{ij})$ for all $p_i \in P_t$
$\{p_i p_j\} \in L$

$$\therefore z^* \geq \sum_{p_i \in P_t} w_{h(p_i)} \geq \sum_{\substack{p_i \in P_t \\ \{p_i p_j\} \in L}} \min(w_{ij})$$

$$= \sum_{p_i \in S} \min(w_{ij}) - w_{gd}$$

$$\geq \sum_{i=1}^{m} \min_{j=i\ldots m}(w_{ij}) - \min_{\substack{i=1\ldots m \\ j=1\ldots m}}(w_{ij})$$

$$= c$$

Thus $z^* \geq b$ or $z^* \geq c$, and so the theorem is proved.

Two numerical examples of the bounding process follow.

Example I

$$W = \begin{bmatrix} \infty & 5 & \infty & 5 & 4 \\ 5 & \infty & 5 & \infty & 4 \\ \infty & 5 & \infty & 5 & 4 \\ 5 & \infty & 5 & \infty & 4 \\ 4 & 4 & 4 & 4 & \infty \end{bmatrix}$$

$n = 5,\ m = 4$

then $b = 4 + 4 + 4 + 4$

$= 16$

$c = 5 + 5 + 5 + 5 - 5$

$= 15$

and in fact here $z^* = 15$.

Example II

$$W = \begin{bmatrix} \infty & 3 & \infty & 3 & 2 \\ 3 & \infty & 3 & \infty & 2 \\ \infty & 3 & \infty & 3 & 2 \\ 3 & \infty & 3 & \infty & 2 \\ 2 & 2 & 2 & 2 & \infty \end{bmatrix}$$

$n = 5,\ m = 4$

then $b = 2 + 2 + 2 + 2$

$= 8$

$c = 3 + 3 + 3 + 3 - 3$

$= 9$

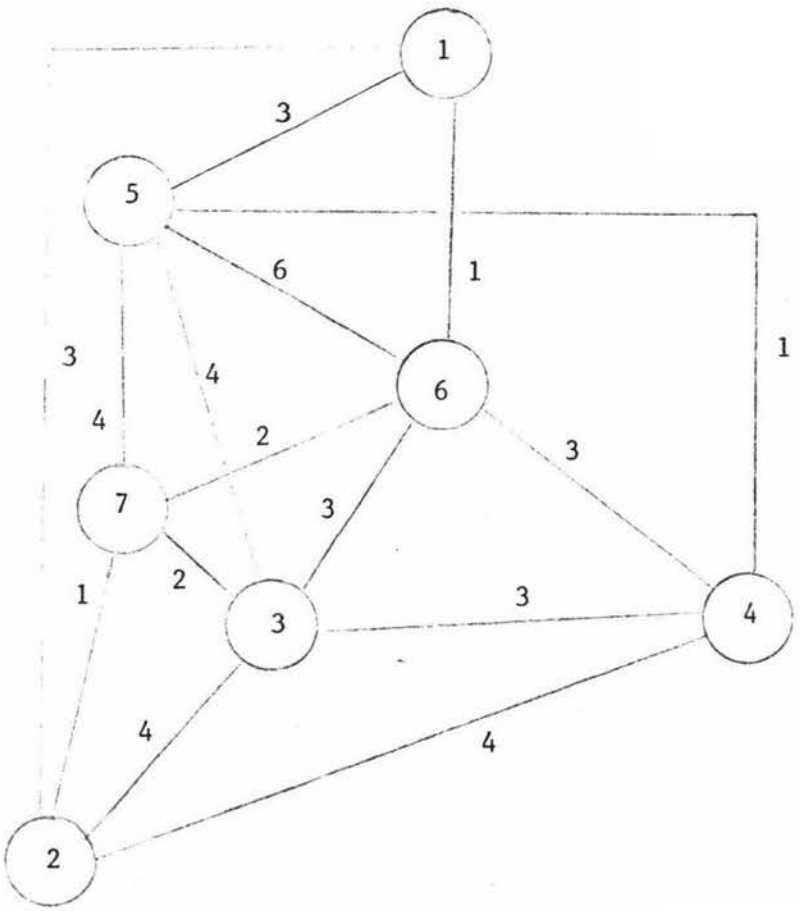
thus $z^* \geq \min(8,9)$

$= 8$

and in fact here $z^* = 8$.

2.5.2.5 A numerical example of the branch and bound method

Figure 2.7 shows the decision tree for the branch and bound process, and each node shows its lower bound in parentheses. Let z' be the cost of the current incumbent solution. Initially we do not have an incumbent solution, and hence set $z' = \infty$.



Node 0

$$W = \begin{bmatrix} \infty & 3 & \infty & \infty & 3 & 1 & \infty \\ 3 & \infty & 4 & 4 & \infty & \infty & 1 \\ \infty & 4 & \infty & 3 & 4 & 3 & 2 \\ \infty & 4 & 3 & \infty & 1 & 3 & \infty \\ 3 & \infty & 4 & 1 & \infty & 6 & 4 \\ 1 & \infty & 3 & 3 & 6 & \infty & 2 \\ \infty & 1 & 2 & \infty & 4 & 2 & \infty \end{bmatrix}$$

$n = 7, m = 4$

The component vector C = {(1),(2),(3),(4)}

Lower bound = min(5,9) = 5

Partial solution is unfathomed

T = penalty vector = {(3-1),(4-1),(3-2),(3-1)} = (2,3,1,2)

Hence we branch on line $\{p_2, p_7\}$

Node 1 $\{p_2, p_7\}$ included

$$W = \begin{bmatrix} \infty & 3 & \infty & \infty & 3 & 1 \\ 3 & \infty & 2 & 4 & 4 & 2 \\ \infty & 2 & \infty & 3 & 4 & 3 \\ \infty & 4 & 3 & \infty & 1 & 3 \\ 3 & 4 & 4 & 1 & \infty & 6 \\ 1 & 2 & 3 & 3 & 6 & \infty \end{bmatrix}$$

C = {(1),(2,7),(3),(4)}

Lower bound = min(6,8) + 1 = 7

Partial solution is unfathomed

T = {(3-1),(2-2),(3,2),(3-1)} = (2,0,1,2)

Hence we branch on one $\{p_1, p_6\}$

Node 2 $\{p_1, p_6\}$ included

$$W = \begin{bmatrix} \infty & 2 & 3 & 3 & 3 \\ 2 & \infty & 2 & 4 & 4 \\ 3 & 2 & \infty & 3 & 4 \\ 3 & 4 & 3 & \infty & 1 \\ 3 & 4 & 4 & 1 & \infty \end{bmatrix}$$

C = {(1,6),(2,7),(3),(4)}

Lower bound = min(7,7) + 2 = 9

Partial solution is unfathomed

T = {(3-2),(2-2),(3-2),(3-1)} = (1,0,1,2)

Hence we branch on line $\{p_4, p_5\}$

Node 3 $\{p_4, p_5\}$ included

$$W = \begin{bmatrix} \infty & 2 & 3 & 3 \\ 2 & \infty & 2 & 4 \\ 3 & 2 & \infty & 3 \\ 3 & 4 & 3 & \infty \end{bmatrix}$$

C = {(1,6),(2,7),(3),(4,5)}

Lower bound = min(9,7) + 3 = 10

Partial solution is unfathomed

T = {(3-2),(2-2),(3-2),(3-3)} = (1,0,1,0)

Hence we branch on $\{p_6, p_7\}$

Node 4 $\{p_6, p_7\}$ included

$$W = \begin{bmatrix} \infty & 2 & 3 \\ 2 & \infty & 3 \\ 3 & 3 & \infty \end{bmatrix}$$

C = {(1,2,6,7),(3),(4,5)}

Lower bound = min(7,5) + 5 = 10

Partial solution is unfathomed

T = {(3-2),(3-2),(3-3)} = (1,1,0)

Hence we branch on $\{p_3, p_7\}$

Node 5 $\{p_3, p_7\}$ included

$$W = \begin{bmatrix} \infty & 3 \\ 3 & \infty \end{bmatrix}$$

C = {(1,2,3,6,7),(4,5)}

Lower bound = min(6,3) + 7 = 10

Partial solution is unfathomed

T = {(∞-3),(∞-3)} = (∞,∞)

Hence we branch on $\{p_4, p_6\}$

Node 6 $\{p_4, p_6\}$ included

The set of included lines forms a feasible solution, hence this partial solution is fathomed. The cost of this feasible solution is less than the current incumbent, and so a new incumbent is set.

z' = 10

Node 7 $\{p_4, p_6\}$ excluded

Backtrack to node 5

$$W = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$$

C = {(1,2,3,6,7),(4,5)}

Lower bound = min(∞,∞) + 7 = ∞

Lower bound ≥ z', and hence the partial solution is fathomed as not containing a better solution and is discarded.

Node 8 $\{p_3, p_7\}$ excluded

Backtrack to node 4

$$W = \begin{bmatrix} \infty & \infty & 3 \\ \infty & \infty & 3 \\ 3 & 3 & \infty \end{bmatrix}$$

C = {(1,2,6,7),(3),(4,5)}

Lower bound = min(9,6) + 5 = 11

Lower bound ≥ z', hence discard.

Node 9 $\{p_6, p_7\}$ excluded

Backtrack to node 3

$$W = \begin{bmatrix} \infty & \infty & 3 & 3 \\ \infty & \infty & 2 & 4 \\ 3 & 2 & \infty & 3 \\ 3 & 4 & 3 & \infty \end{bmatrix}$$

C = {(1,6),(2,7),(3),(4,5)}

Lower bound = min(10,8) + 3 = 11

Lower bound ≥ z', hence discard this partial solution.

Node 10 $\{p_4, p_5\}$ excluded

Backtrack to node 2.

$$W = \begin{bmatrix} \infty & 2 & 3 & 3 & 3 \\ 2 & \infty & 2 & 4 & 4 \\ 3 & 2 & \infty & 3 & 4 \\ 3 & 4 & 3 & \infty & \infty \\ 3 & 4 & 4 & \infty & \infty \end{bmatrix}$$

C = {(1,6),(2,7),(3),(4)}

Lower bound = min(9,7) + 2 = 9

The partial solution is unfathomed.

T = {(3-2),(2-2),(3-2),(3-3)} = (1,0,1,0)

Hence we branch on $\{p_6, p_7\}$

Node 11 $\{p_6, p_7\}$ included

$$W = \begin{bmatrix} \infty & 2 & 3 & 3 \\ 2 & \infty & 3 & 4 \\ 3 & 3 & \infty & \infty \\ 3 & 4 & \infty & \infty \end{bmatrix}$$

C = {(1,2,6,7),(3),(4)}

Lower bound = min(7,5) + 4 = 9

The partial solution is unfathomed.

T = {(3-2),(3-2),(3-3)} = (1,1,0)

Hence we branch on $\{p_3, p_7\}$

Node 12 $\{p_3, p_7\}$ included

$$W = \begin{bmatrix} \infty & 3 & 3 \\ 3 & \infty & \infty \\ 3 & \infty & \infty \end{bmatrix}$$

C = {(1,2,3,6,7),(4)}

Lower bound = min(6,3) + 3 = 9

The partial solution is unfathomed.

T = {(3-3),(∞-3)} = (0,∞)

Hence we branch on line $\{p_4,p_6\}$

Node 13 $\{p_4,p_6\}$ included

The set of included lines forms a feasible solution, hence this partial solution is fathomed. The cost of this feasible solution is less than the current incumbent, and so a new incumbent is set.

z' = 9

Node 14 $\{p_4,p_6\}$ excluded

Backtrack to node 12.

$$W = \begin{bmatrix} \infty & 3 \\ 3 & \infty \end{bmatrix}$$

C = {(1,2,3,6,7),(4)}

Lower bound = min(∞,∞) + 6 = ∞

Lower bound ≥ z', and hence discard this partial solution.

Node 15 $\{p_3,p_7\}$ excluded

Backtrack to node 11

$$W = \begin{bmatrix} \infty & \infty & 3 & 3 \\ \infty & \infty & 3 & 4 \\ 3 & 3 & \infty & \infty \\ 3 & 4 & \infty & \infty \end{bmatrix}$$

C = {(1,2,6,7),(3),(4)}

Lower bound = min(9,6) + 4 = 10

Lower bound ≥ z', and hence discard this partial solution.

Node 16 $\{p_3, p_7\}$ excluded

Backtrack to node 10.

$$W = \begin{bmatrix} \infty & \infty & 3 & 3 & 3 \\ \infty & \infty & 2 & 4 & 4 \\ 3 & 2 & \infty & 3 & 4 \\ 3 & 4 & 3 & \infty & \infty \\ 3 & 4 & 4 & \infty & \infty \end{bmatrix}$$

C = {(1,6),(2,7),(3),(4)}

Lower bound = min(10,8) + 2 = 10

Lower bound ≥ z', and hence discard this partial solution.

Node 17 $\{p_1, p_6\}$ excluded

Backtrack to node 1.

$$W = \begin{bmatrix} \infty & 3 & \infty & \infty & 3 & \infty \\ 3 & \infty & 2 & 4 & 4 & 2 \\ \infty & 2 & \infty & 3 & 4 & 3 \\ \infty & 4 & 3 & \infty & 1 & 3 \\ 3 & 4 & 4 & 1 & \infty & 6 \\ \infty & 2 & 3 & 3 & 6 & \infty \end{bmatrix}$$

C = {(1),(2,7),(3),(4)}

Lower bound = min(8,8) + 1 = 9

Lower bound ≥ z', and hence discard this partial solution.

Node 18 $\{p_2, p_7\}$ included

Backtrack to node 0.

$$W = \begin{bmatrix} \infty & 3 & \infty & \infty & 3 & 1 & \infty \\ 3 & \infty & 4 & 4 & \infty & \infty & \infty \\ \infty & 4 & \infty & 3 & 4 & 3 & 2 \\ \infty & 4 & 3 & \infty & 1 & 3 & \infty \\ 3 & \infty & 4 & 1 & \infty & 6 & 4 \\ 1 & \infty & 3 & 3 & 6 & \infty & 2 \\ \infty & \infty & 2 & \infty & 4 & 2 & \infty \end{bmatrix}$$

C = {(1),(2),(3),(4)}

Lower bound = min(7,9) = 7

This partial solution is unfathomed.

T = {(3-1),(4-3),(3-2),(3-1)} = (2,1,1,2)

Hence we branch on line $\{p_1, p_6\}$.

Node 19 $\{p_1, p_6\}$ included

$$W = \begin{bmatrix} \infty & 3 & 3 & 3 & 3 & 2 \\ 3 & \infty & 4 & 4 & \infty & \infty \\ 3 & 4 & \infty & 3 & 4 & 2 \\ 3 & 4 & 3 & 1 & 1 & \infty \\ 3 & \infty & 4 & 1 & \infty & 4 \\ 2 & \infty & 2 & \infty & 4 & \infty \end{bmatrix}$$

C = {(1,6),(2),(3),(4)}

Lower bound = min(8,9) + 1 = 9

Lower bound $\geq z'$, and hence discard this partial solution.

Node 20 $\{p_1, p_6\}$ excluded

Backtrack to node 18.

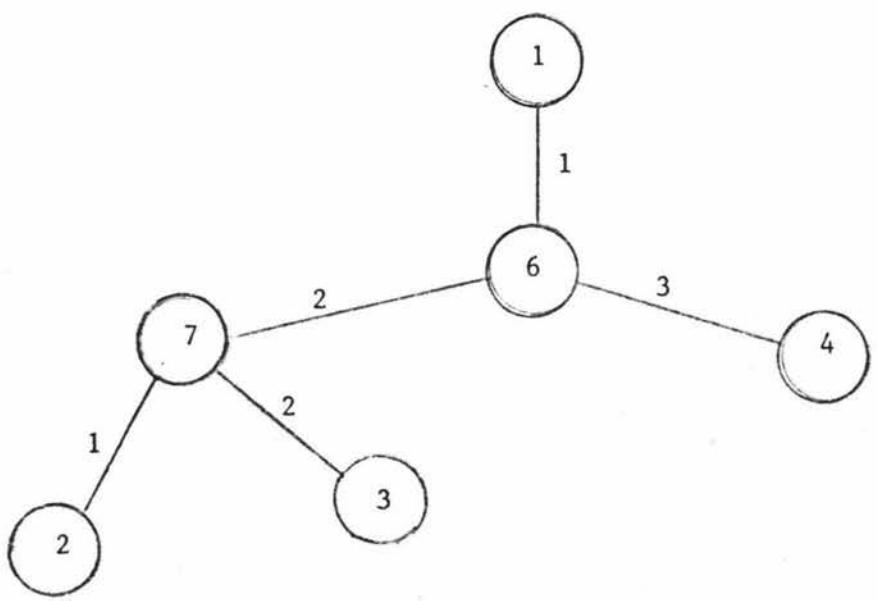
$$W = \begin{bmatrix} \infty & 3 & \infty & \infty & 3 & \infty & \infty \\ 3 & \infty & 4 & 4 & \infty & \infty & \infty \\ \infty & 4 & \infty & 3 & 4 & 3 & 2 \\ \infty & 4 & 3 & \infty & 1 & 3 & \infty \\ 3 & \infty & 4 & 1 & \infty & 6 & 4 \\ \infty & \infty & 3 & 3 & 6 & \infty & 2 \\ \infty & \infty & 2 & \infty & 4 & 2 & \infty \end{bmatrix}$$

$C = \{(1),(2),(3),(4)\}$

Lower bound = $\min(9,9) = 9$

Lower bound $\geq z'$, and hence discard this partial solution.

Thus we end the algorithm, and the incumbent solution becomes the SMST. This was set at node 13, and the set of included lines at that time was $\{(p_2,p_7),(p_1,p_6),(p_6,p_7),(p_3,p_7),(p_4,p_6)\}$. So the SMST looks like:



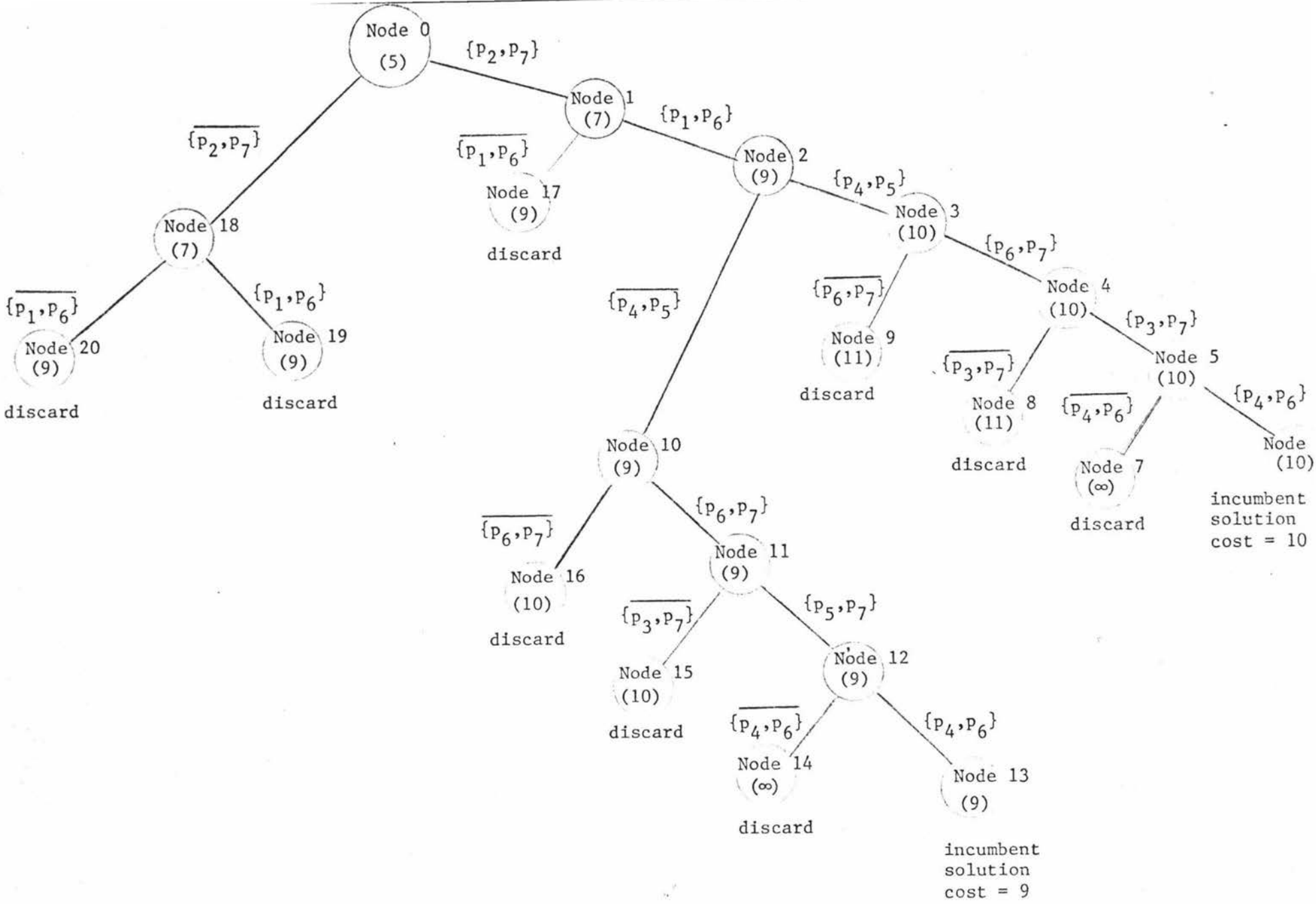


Figure 2.7 The decision tree for the branch and bound numerical example

2.5.3 Other Considerations

Although no initial upper bound was used in the algorithm, one could be generated as follows. Consider a graph G = (P,L) with $S \subset P$ the set of essential points, and $T \subset L$ the set of lines connecting the points in S. Then the Kevin and Whitney implementation of Dijkstra's algorithm for the MST could be used to generate a MST for the graph G* = (S,T). This would then be an upper bound on the SMST for G. However, it was deemed that the branch and bound algorithm as implemented found an incumbent solution (thus giving a very good upper bound on the SMST) rapidly enough to warrant the non inclusion of an initial upper bound.

An extension to the upper bounding was considered - that of including as a lower bound a value

$$a = \max_{\substack{i = 1...m \\ j = 1...m}} (\text{shortest path } \langle i....j \rangle)$$

thus giving

$$z^* \geq \min(a,b,c)$$

where b and c are as previously defined for the lower bound.

This is not of much use when a line is added to the set of included lines, but becomes significant when a line is excluded from consideration. This shortest path calculation can be performed in one of two ways - either by complete recalculation (see Floyd [1961]) or, using incremental techniques, by adjustment of the shortest path matrix.

In the first case, due to the way the branch and bound method was implemented,

the weight matrix must effectively be condensed to a weight matrix between components, either by actually calculating a new matrix, or by amending the shortest path algorithm to reflect the component nature of the problem.

In the second case, an incremental approach would need a calculation time of less than $0(n^3)$ to be worthwhile (as the total recalculation is $0(n^3)$). However the incremental approach would need to reflect all paths affected by the removal of any line and this would involve some method of storing $\frac{n(n-1)}{2}$ shortest paths. As a path could contain up to n-1 lines, the storage of this shortest path matrix would involve $0(n^3)$ items.

These two methods of continually producing a shortest path matrix at each exclusion of a line have not been completely examined, although it is thought that their implementation would not significantly affect the time taken to solve the larger (and hence more time consuming) problems. The use of this maximum shortest path as an initial adjunct to the lower bounding of node 0 was tested, however the results indicated that there was little to be gained by this approach, and in fact in the large proportion of the test cases, the cost of performing this shortest path calculation was greater than the time saved by having the value included in the lower bound. Hence this use of the shortest path matrix was discarded.

2.5.4 Timings

Figure 2.8 shows, in graphical form, the time taken to solve a number of randomly generated graphs, using the branch and bound method. The time is mill seconds taken, and the method was implemented in Algol on a Burroughs B6700 computer.

The method of generating a random graph is as follows:

(a) Randomly generate the total number of points in the graph.

(b) Randomly generate the number of essential points, and if this is greater than the total number of points, swap them.

(c) Randomly generate a tree connecting all essential points.

(d) Generate a random number of lines connecting any two random points in the graph.

(e) Randomly assign weights to each line in the graph.

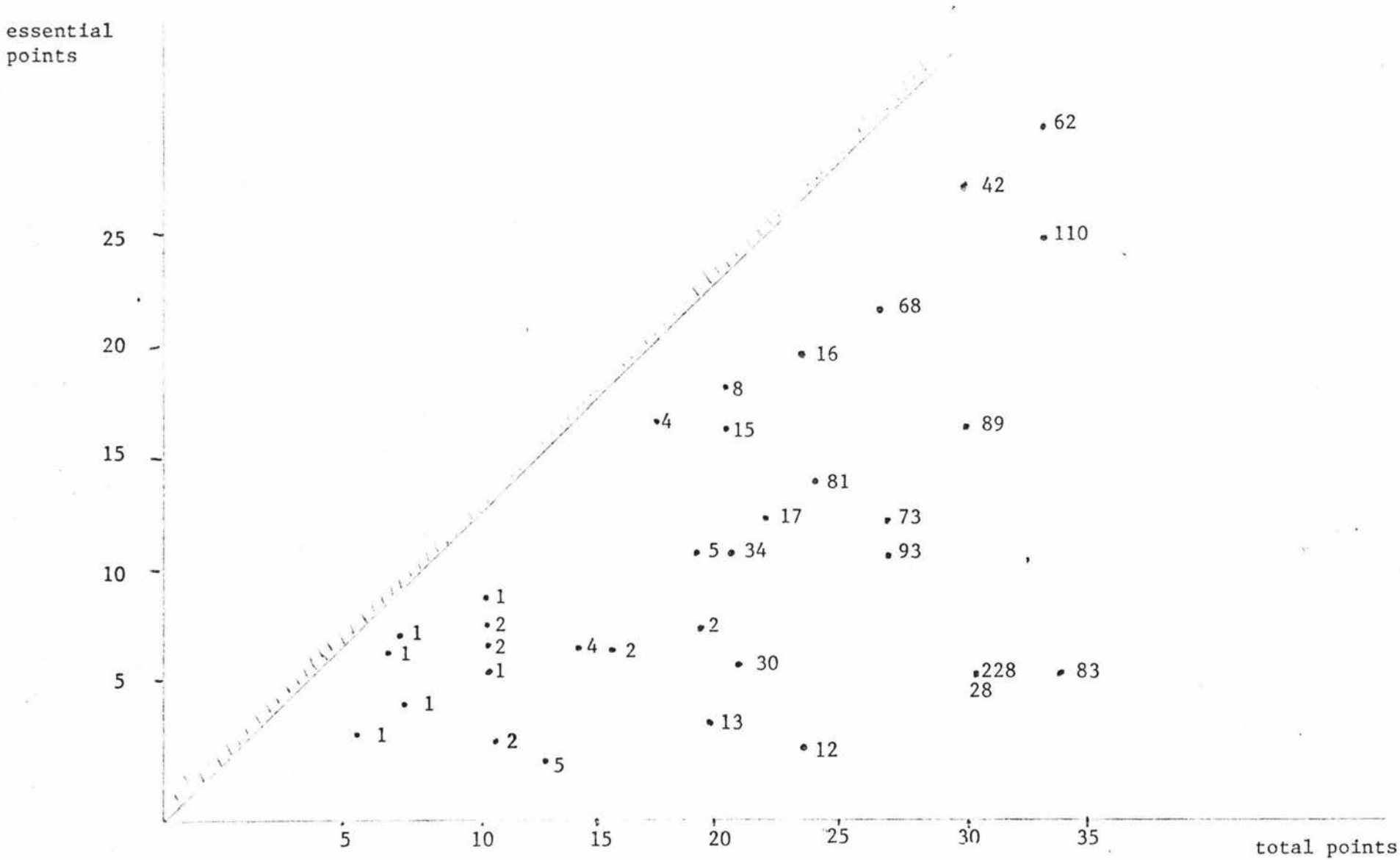


Figure 2.8 CPU time (mill seconds) for the branch-and-bound method

2.5.5 Timing Comparisons with Other SPG Methods

A number of complete graphs were randomly generated, for a fixed number of total and essential nodes, and used as benchmarks for all three methods described in this chapter for solving the SPG, i.e.

(a) Hakimi

(b) Dreyfus and Wagner

(c) Branch and Bound

Unfortunately it was not possible to obtain a copy of the Dreyfus and Wagner code for their solution method, and a great deal of effort went into developing the best implementation of the algorithm as possible for this method.

			Average time (secs)			Range of time (secs)		
Runs	m	n	Hakimi	Dreyfus	Branch and-Bound	Hakimi	Dreyfus	Branch and-Bound
5	3	10	5	<1	3	-	-	1-4
5	5	10	1	10	2	1-2	5-6	0-3
5	8	10	<1	100+	1	-	-	1-2
5	5	20	100+	21	1	-	20-22	9-77
5	10	20	88	100+	21	82-95	-	12-29
5	15	20	3	100+	22	-	-	9-51
5	5	30	100+	45	100+	-	45-46	72-100+
5	15	30	100+	100+	90	-	-	56-100+
5	25	30	6	100+	80	6-7	-	53-100+

Table 2.1

m number of essential points

n number of total points

time in mill seconds on a Burroughs B6700, coded in Algol

The results of the comparison are shown graphically, as in Figure 2.9, and in tabular form in Table 2.1. It is interesting to note the domains over which the various methods are superior; it can be seen that for

(a) $n \gg m$, the Dreyfus and Wagner solution is superior.

(b) $n \simeq m$, the Hakimi solution is superior.

(c) $\frac{n}{2} \simeq m$, the branch and bound solution is superior.

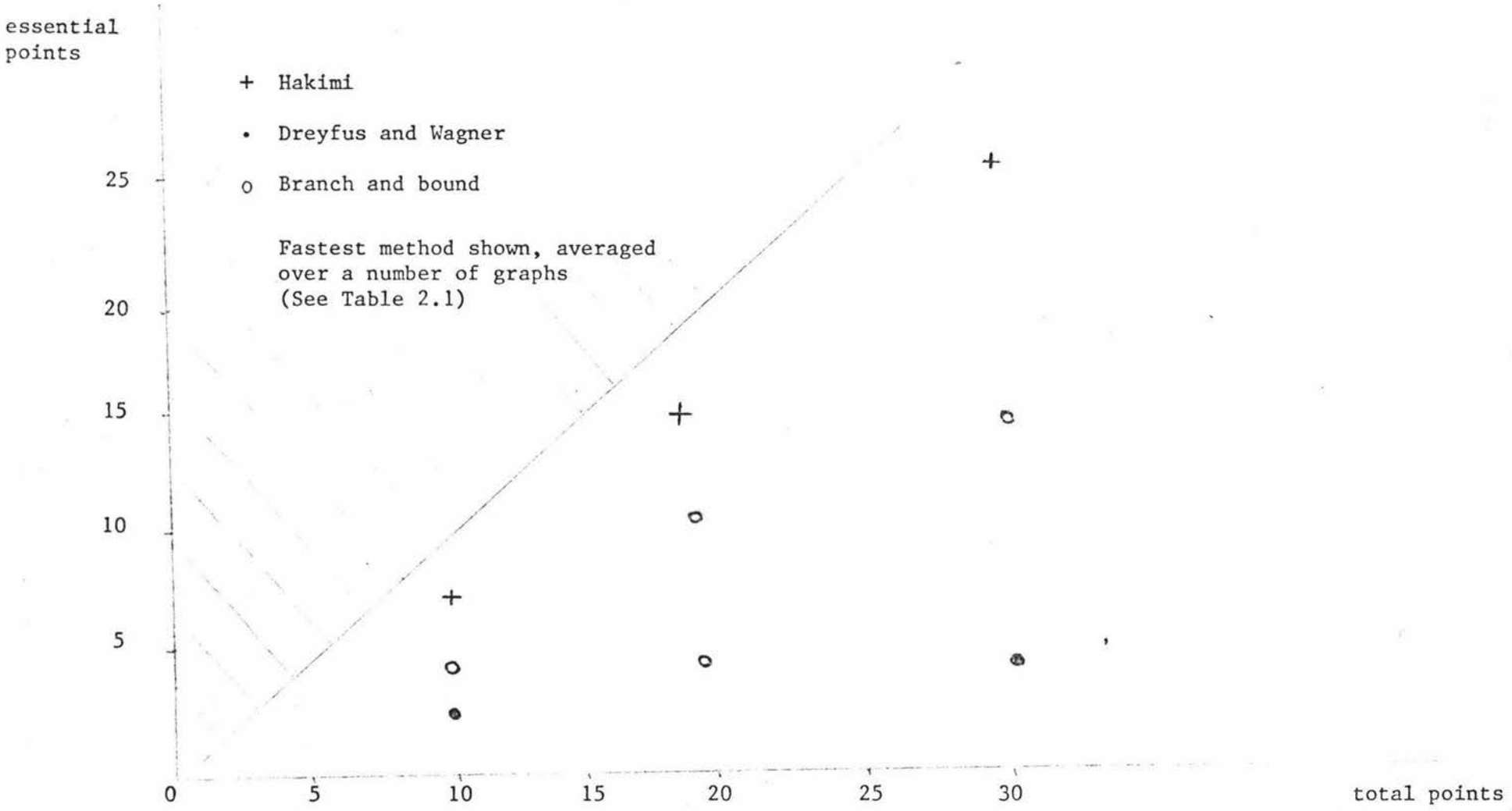


Figure 2.9 SPG timing comparisons

2.5.6 Conclusions

It can be seen from the preceding results that the branch and bound technique provides a good method of solving the SPG, filling in a gap that existed between the Hakimi method and the Dreyfus and Wagner method. However, the time taken to solve a problem using the branch and bound method is somewhat unpredictable; the test graphs generated showed a large range in times taken to solve the randomly generated problems. The graphs also do not show memory requirements of the various methods. It was found that the Hakimi method required less memory than the branch and bound, which in its turn required less than the Dreyfus and Wagner method. Again, for the branch and bound method this memory requirement is unpredictable, as it depends to a large extent on the depth of recursion that the program has to go to, which is itself dependent upon the type of graph being analysed. In the test graph runs, the memory requirements for the branch and bound method varied between 3490 and 7870 bytes.

CHAPTER 3

PHYLOGENY

3.1 Introduction

We now turn our attention to a somewhat different problem to that dealt with in the last chapter. This new problem is one concerning evolution, and the links that have existed in the past between existing species. No two members of a species are exactly the same (except identical twins!); each has slight modifications from their parents. As environmental conditions change, nature will favour that branch of a species with some particular modification; as time goes on another mutation of the basic stock will become dominant. In this way, all the species are continually evolving, this evolution occurring in a number of ways at the same time, some to die out and some to become new species in their own right. We can describe this linking in a diagram which we call a phylogeny, or phylogenetic tree. For example,

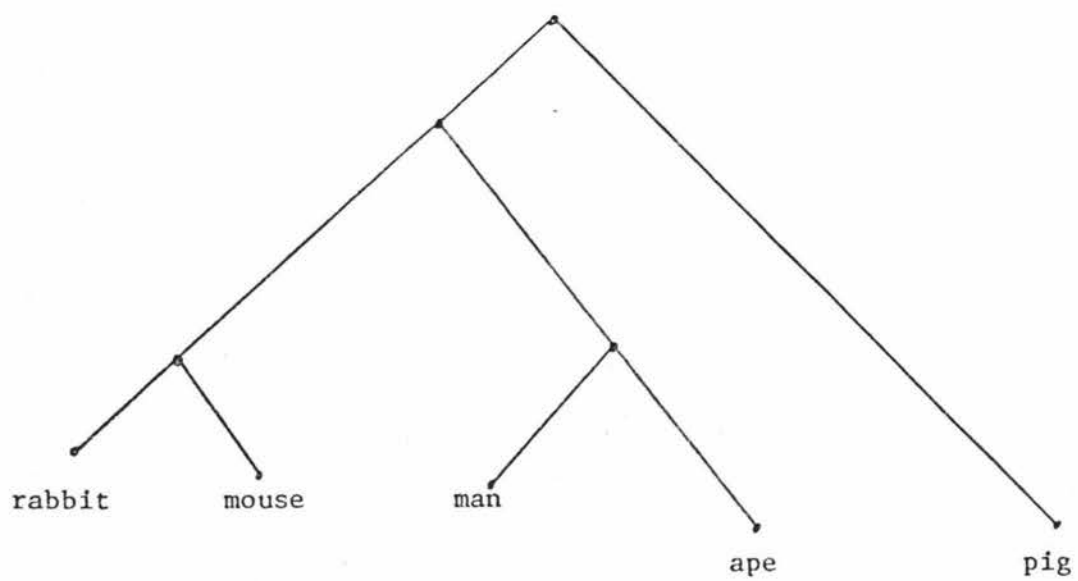


Figure 3.1

A phylogeny

There have been many phylogenies suggested to link both existing species and organisms in the fossil record. With some groups, particularly the vertebrates, the information gleaned from fossil records together with comparative information from existing species allow phylogenies to be determined for which there is a fair degree of agreement among researchers. But for most species the fossil record is either inadequate or non-existant and the evolution of the group must be determined from a knowledge of existing species. There are a number of methods used to determine phylogenies by objective methods, and these will be discussed later. Penny [1976] subdivided the process of determining phylogenies into six steps as follows:

(a) Collecting data.
(b) Selecting a biological model.
(c) Deciding on optimality criteria.
(d) Generating an optimal network.
(e) Deciding ancestral states.
(f) Determining which point in the tree represents the root.

We are interested only in the second, third and fourth steps in this process. The biological model chosen is one in which species are represented by some protein common to the sets of species under study, and the links between the species being the changes that occur in the proteins between those species. Although two species may have a common protein, say cytochrome-c, a respiratory protein, the detailed composition of that protein may differ. In fact two members of the same species will have a protein which, while on the whole representing the species, may contain minor differences distinguishing the members. The criterion we shall use to decide whether a tree is optimal is the minimisation of these differences between species over the whole tree. In order to simplify the problem a little, when we consider proteins we only use the parts of the

protein common to the species, and hence ignore these changes among members of one species.

Nucleotides

A protein is a sequence of amino acids, each acid being coded for by a sequence of three nucleic acid bases, or nucleotides. There are four possible nucleotide codes - ACGU, thus giving rise to 4^3 or 64 possible amino acid codings. However only 20 amino acids appear to exist in nature, and hence we find that an amino acid may be coded for by more than one nucleotide sequence. For further details on the biochemical aspects, see Watson [1975].

The nucleotides are ordered within the protein, and the relative position they occupy is called their site. The change of one nucleotide to another, at a particular site, is called a base change, and the conversion of a protein in one species to the corresponding protein in another species will involve a number of these base changes. Researchers cannot always define the exact nucleotide that occurs at a particular site in a species protein, and so we will use the following table to map codes given to actual nucleotides:

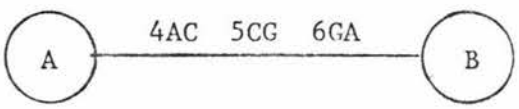
coding	A	G	C	U	R	Y	M
possible nucleotides	A	G	C	U	A G	U C	A G C U

Table 3.1

Nucleotide Codings

A phylogenetic tree, or phylogeny, is shown as a tree consisting of points which represent species, and lines between points on which we denote the base

changes in the form r α β, where r is the site number, α is the nucleotide coding for species 1, and β is the nucleotide coding for species 2. Obviously, as the tree lines have no direction, the choice of order for the two nucleotide codings is to an extent arbitrary. As an example of this method of describing a phylogeny, consider two species, coded as AAGACG and AAGCGA, linked together. This would be represented as



Phylogeny

	Site					
	1	2	3	4	5	6
species A	A	A	G	A	C	G
B	A	A	G	C	G	A

Species Table

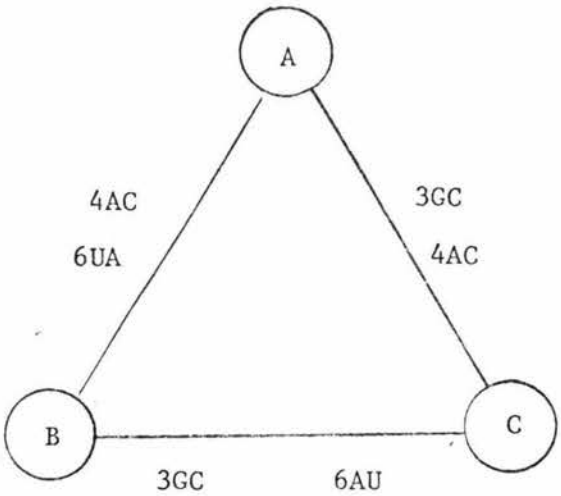
Figure 3.2

Representing Phylogenies

We have taken the criterion for optimality of a phylogeny as to minimise these base changes, and so our problems becomes that of producing a tree with the minimum number of base changes.

Generating the tree

The problem of generating a phylogeny is not, however, merely a Minimal Spanning Tree problem. This can be shown with the following example. Consider three species coded, for some part of a protein, as AAGACU, AAGCCA, AACCCU. The network joining these species is

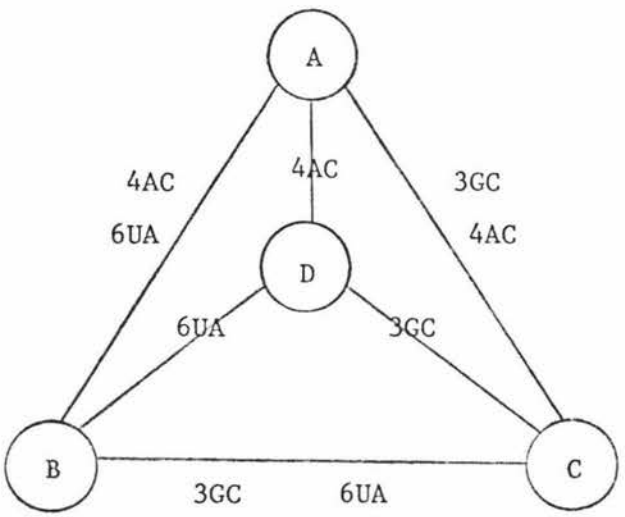


	Sites					
	1	2	3	4	5	6
species A	A	A	G	A	C	U
B	A	A	G	C	C	A
C	A	A	C	C	C	U

Figure 3.3. (a)

An initial network

Thus any tree linking A, B and C would involve 4 base changes. However, now consider an intermediate species D, coded AAGCCU.



	Sites					
	1	2	3	4	5	6
species A	A	A	G	A	C	U
B	A	A	G	C	C	A
C	A	A	C	C	C	U
D	A	A	G	C	C	U

Figure 3.3 (b)

Generating a Steiner Point

Now a tree is possible which links A, B, C and D with a total cost of three base changes, i.e.

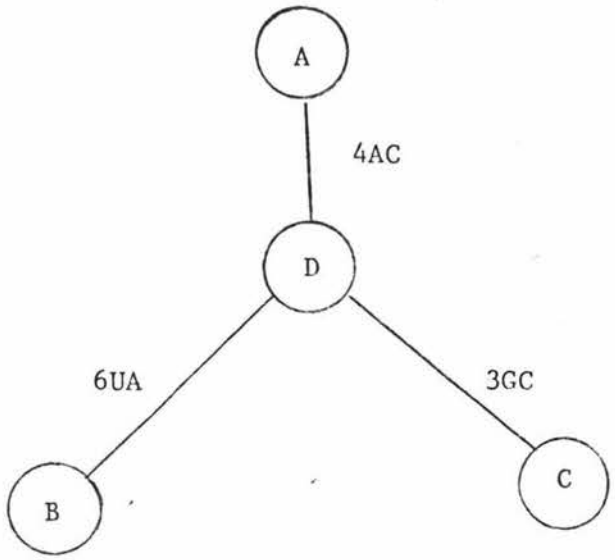


Figure 3.3 (c)

The Final Phylogeny

This process is called coalescing, and the species D is called a Steiner point, as it is not essential to the problem, but may be generated in order to minimise the number of base changes in the tree. The species A, B and C we call original species. So the problem then is one of linking together original species, with the possible formation of Steiner points, to produce a spanning tree that is of minimal total cost. Thus the problem can be considered as the Steiner Problem in Graphs, with two important differences:

(a) The Steiner points are not given, but must be generated from the original data.

(b) The triangular inequality holds - i.e. in a triangle ABC of species, each link ({A,B}, {A,C}, {B,C}) is less than or equal to the sum of the other two.

When generating a Steiner point, we have to decide what the nucleotide coding is to be for each site. Now this point when generated will connect to three species already existing in the tree, and so the coding at any particular

site in the Steiner point will be the same as the coding at that site for one or more of the original species. If at a site, a code appears in more than one of the original species, then that code is used for the Steiner point at that site. If the three codes for the original species at that site are distinct, then the code that appears most often throughout all the species in the data set at that site is used.

Cycles

Another problem encountered when generating a phylogeny is that of the existence of cycles. If we consider a typical species graph, as in Figure 3.4, we can define a path as an alternating sequence of points representing species, and lines connecting points.

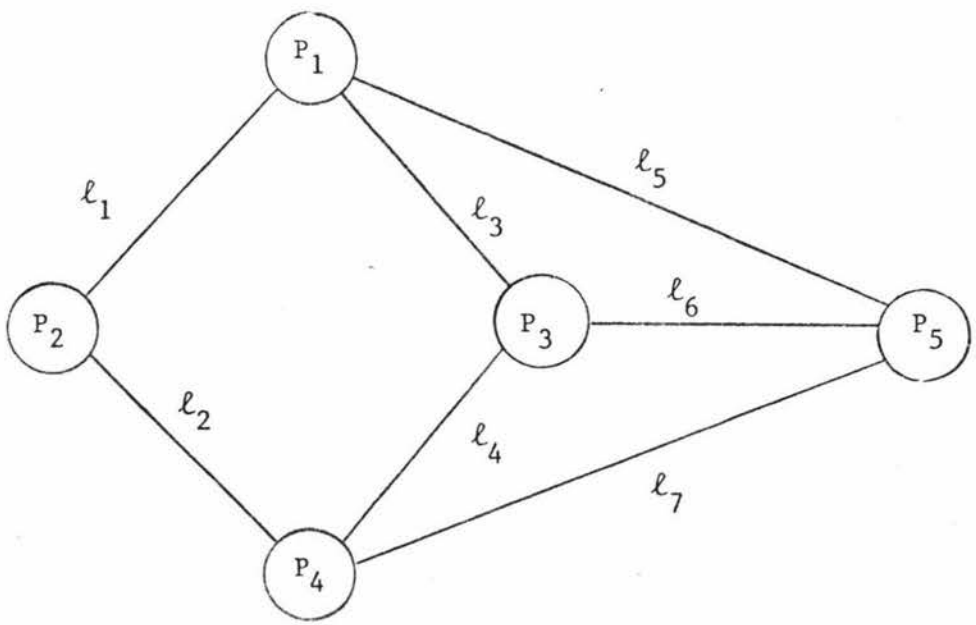


Figure 3.4

A typical species graph

For example, $\langle p_1, \ell_1, p_2, \ell_2, p_4, \ell_4, p_3 \rangle$ is a path. Now if we have a path $\langle p_i, \ldots, p_j \rangle$ such that $p_i = p_j$, then we call that path a cycle.

In the case of a phylogeny, it is easy to see why a cycle is undesirable - it would cause an increase in cost. For example

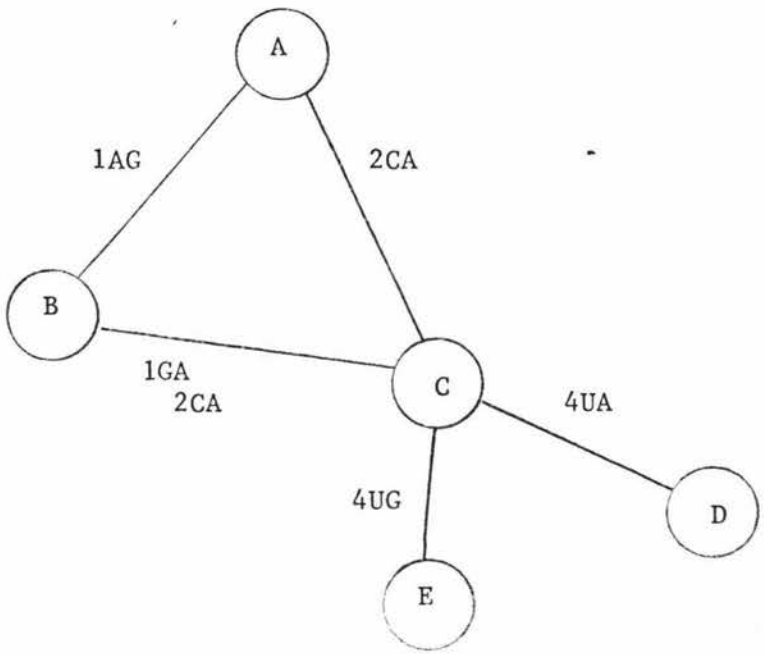


Figure 3.5 (a)

Phylogeny cost with a cycle

This graph contains the cycle {(A B), (B C), (C A)} and is of weight 6 in total. We can remove one of the cycle lines, say (B C) and produce a tree of weight 4 in total as below:

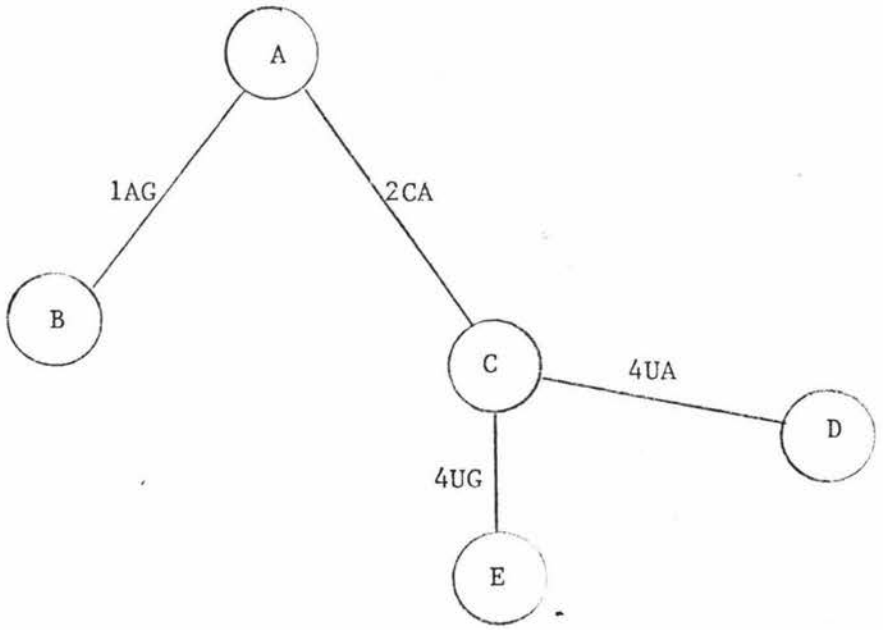


Figure 3.5 (b)

Phylogeny cost without the cycle

Connectivity

The problem of checking for connectivity of the tree at intermediate stages during tree building was looked at in depth and the method most suited to this application was found to be that of component merging. This method involves considering each original point in the tree as belonging to a connected component (i.e. a subset of points which are connected by lines in the tree).

The tree is thus disconnected whilst there exists more than one component, and vice versa when there is only one component then the tree must be connected. Other methods of connectivity checking looked at were breadth first search, depth first search and the method of transitive closure.

Methods of solving the problem

Camin and Sokal [1965] assign a variable (e.g. number of limbs, for animals) to each species, and for each variable an ordered set of values. They assume that mutation takes place only from a lower value to a higher value. Estabrook [1968] extends this structure on the values of the variable to be a partial order with tree structure. In both cases the final tree topology is given. Cavalli-Sforza and Edwards [1967] consider the problem using continuous variables which, for a given tree, is equivalent to the SPG. Farris [1970] considers the final tree as binary, as does Fitch [1971]. However Fitch also specifies rules for constructing a general tree, and proves that the minimal tree does occur. Hartigan [1973] gives the mathematical proof of the Fitch [1971] method. Fitch [1977] presents procedures for discovering the tree with minimum changes (he calls this the most parsimonious tree) by minimising the number of discordancies (similar changes occurring in parallel or a change that is changed back later). He does this by a process based on discordancy diagrams. An alternative approach is also presented; constructing a spanning tree of unavoidable distances. A relationship between the minimal spanning tree and the phylogenetic tree is also stated. This method has been tested on small sets of data, and has been shown to produce an optimal tree for up to eight species.

Foulds, Hendy and Penny [1978, 1979] have developed a simple heuristic process for constructing phylogenetic trees, and a method for proving a phylogenetic tree optimal. This approach will be described in detail in the following sections of this chapter. Also presented in the rest of this chapter are various computer programs which automatically generate phylogenies, and the comparison of their phylogenies to optimal phylogenies discussed. Also an extension to the method of proving phylogenies optimal is discussed.

3.2 Building phylogenetic trees

3.2.1 Introduction

In section 3.2 we will look at the Foulds, Hendy and Penny [1978] original method of tree building, at three computer programs using heuristic tree building methods, and finally at the current Penny, Hendy and Foulds [1979] interactive method of generating phylogenies. Appendix II contains data sets representing the partial nucleotide codings for ten species (man, ape, monkey, rabbit, mouse, dog, horse, pig, cow and ewe) for a number of proteins. A large number of the sites have been excluded in these data sets, either because the species have a common nucleotide coding at that site, or because the nucleotide at that site cannot be determined with enough confidence to be worthy of inclusion in the data set. The sites have then been renumbered so the sites in the data set are numbered in sequence starting at 1. The appendix then goes on to list the best tree obtained by the interactive tree building method, and the three trees obtained from the computer programs. After the description of all the tree building methods mentioned above, there is a section discussing the computer programs and what we can deduce from the results of the tree building comparison.

3.2.2 The original Foulds, Hendy and Penny method

In this original method of tree construction, Foulds et. al. suggest the use of a procedure based on the Kruskal [1956] method of solving the Minimal Spanning Tree problem. This method involves entering into the tree being constructed each line joining two points, in non-decreasing order of line weight (in the phylogeny problem this is number of base changes between species), with the restriction that a line is not entered if it would then make a cycle with other lines in the tree. The method would terminate when all points are connected by n-1 lines, for n points. However Foulds et. al. extend this method to cater

for the possibility of coalescing to form Steiner points when adjacent lines contain common base changes for a given site.

The method is as follows:

Step 1 ..... Create a table of difference between species, where an entry d(i,j) in this table is the weight of the line connecting species i and j, i.e. the number of base changes between the species.

Step 2 ..... Order the entries in this table into non-decreasing order of weight.

Step 3 ..... Perform steps 3a to 3c until all species are connected in a single tree.

Step 3a ..... Enter the next (starting with the one with least weight) of the ordered set of lines into the tree, if so doing does not create a cycle.

Step 3b ..... Check whether it is possible to coalesce this new line with an adjacent line already in the tree, to form a Steiner point.

Step 3c ..... If a Steiner point was created at step 3b, enter the Steiner point into the set of species, and enter the lines from this new species to all other species (bar the three lines already in the tree due to coalescing) into the ordered set of lines not yet entered into the tree.

A characteristic of the Kruskal method of building a tree is that during the process a number of intermediate, unconnected trees may exist, which will be connected together at a later stage in the building process.

Numerical example of the original Foulds et. al. method

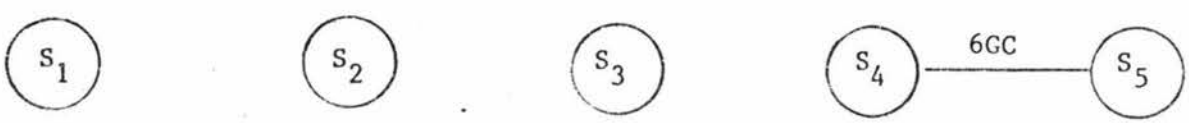
species	Sites 1	2	3	4	5	6
S_1	A	G	C	G	C	G
S_2	G	C	C	G	C	G
S_3	G	G	A	G	C	G
S_4	G	G	G	C	A	G
S_5	G	G	G	C	A	C

Species Table

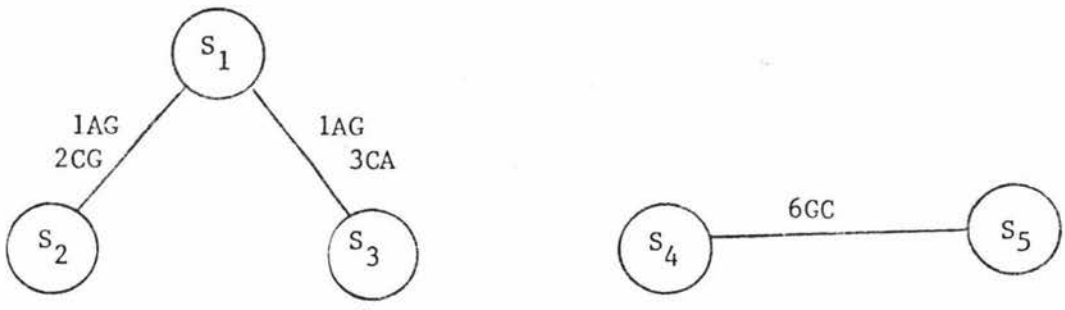
	S_1	S_2	S_3	S_4	S_5
S_1	0	2	2	4	5
S_2	2	0	2	4	5
S_3	2	2	0	3	4
S_4	4	4	3	0	1
S_5	5	5	4	1	0

Difference Table

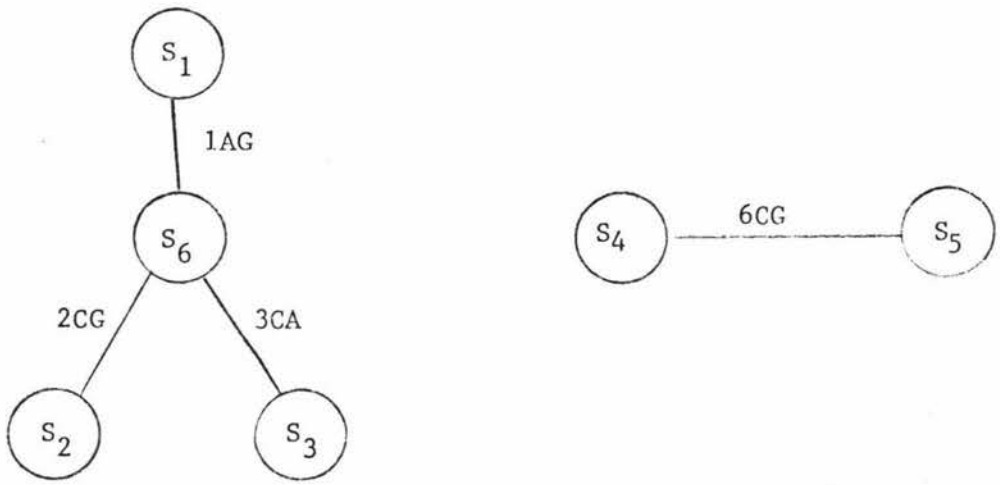
The first line to be entered into the tree is $\{S_4S_5\}$. No coalescing is possible at this point, and the tree looks like:



Next the lines $\{S_1S_2\}$ and $\{S_1S_3\}$ are entered, giving



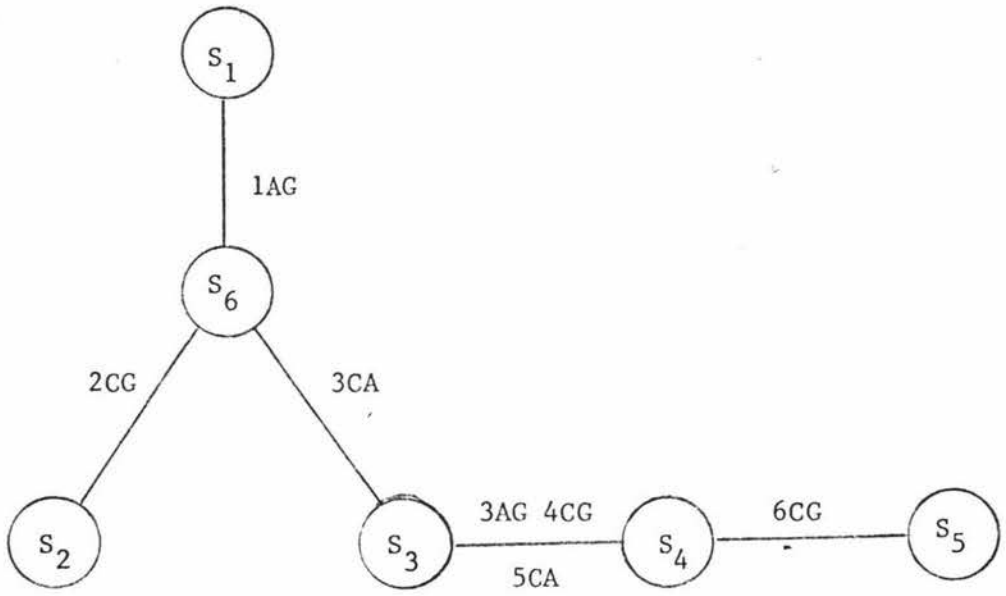
and the two lines just entered can be coalesced, creating point S_6.



Now the difference table is updated with the S_6 entry, i.e.

	S_1	S_2	S_3	S_4	S_5	S_6
S_6	1	1	1	3	4	0

The next line to consider is $\{S_2S_3\}$. However, entering this line would cause a cycle and hence we leave it out. So the next line to be entered is $\{S_3S_4\}$. The tree then is



The graph now connects all original points, and so the algorithm terminates with a PST cost of 7.

3.2.3 PST1

This program is based on the original Foulds, Hendy and Penny algorithm for tree construction, which uses, as stated before, a Kruskal type approach. The actual algorithm implemented follows their suggested method closely, with a few minor procedural alterations. The algorithm is as follows:

Step 1 ..... Generate the difference table between all species.

Step 2 ..... The following steps are performed until the generated tree is one connected component, incrementing a link length inclusion variable at each iteration.

Step 2a ..... Enter all links (not previously entered into the tree) into the tree if their associated weight is less than or equal to

the current value of the link length inclusion variable.

Step 2b ..... Check the tree for cycles. This involves following every path in the tree, maintaining a list of points found on that path to date, and at each new point in the path checking whether that point already exists in this list. If a cycle is found, then the weights of all lines making up the cycle are examined, and the uniquely highest weight line removed. If there is no unique greatest weight line, then the cycle is left in the tree for the moment.

Step 2c ..... Check each adjacent pair of links in the tree for common changes, and on the first occurrence coalesce to form a Steiner point, and enter the Steiner point into the set of species and the difference table.

Step 2d ..... If coalescing took place in step 2c then there may now be the possibility of a hitherto unbreakable cycle becoming breakable (this would occur if the coalescement used one of the highest weight lines in the cycle), and so go on to step 2b. If no coalescement took place go back to step 2 for the next increment of the link length inclusion variable.

Now the resulting graph has been produced as far as the program is concerned, and it is printed. This graph may contain cycles which cannot be broken by the program, and they are left in for the user to break using some other biological knowledge.

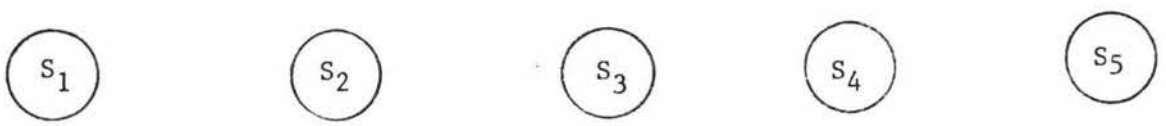
Numerical example of the PST1 method

		Sites					
		1	2	3	4	5	6
species	S_1	A	G	C	G	C	G
	S_2	G	C	C	G	C	G
	S_3	G	G	A	G	C	G
	S_4	G	G	G	C	A	G
	S_5	G	G	G	C	A	C

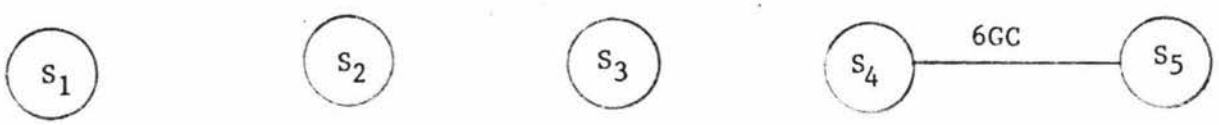
The difference table is calculated as

	S_1	S_2	S_3	S_4	S_5
S_1	0	2	2	4	5
S_2	2	0	2	4	5
S_3	2	2	0	3	4
S_4	4	4	3	0	1
S_5	5	5	4	1	0

The graph is initially five disconnected points.

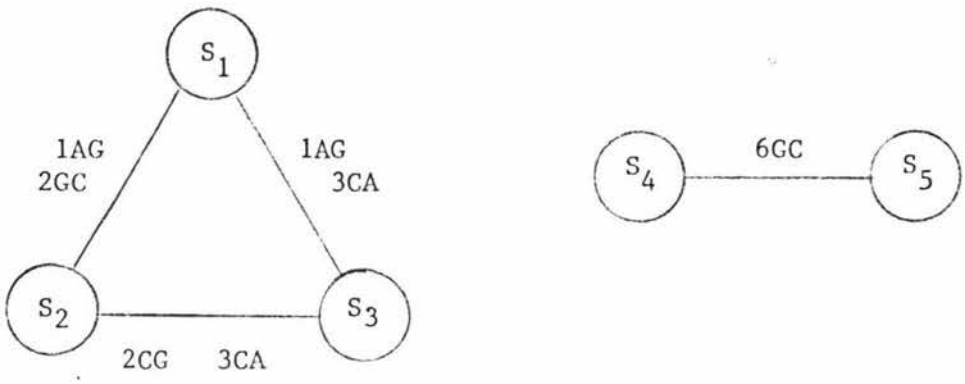


Set the link length inclusion variable, ℓ, to 1. Then by including in the graph all lines with a weight of less than or equal to 1, the graph becomes



There are no cycles and no coalescing is possible. The component vector is (1 2 3 4 4) and so the graph is disconnected.

Now set $\ell = 2$. By including all lines with a weight ≤ 2, the graph becomes

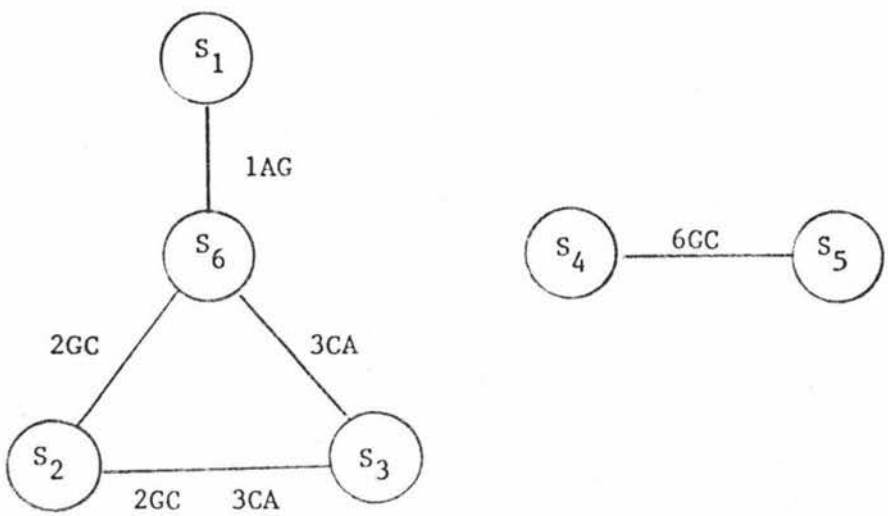


Now there is a cycle in the graph, but it has no uniquely greatest weight line, and so it is left as it is. However, the lines $\{S_1S_2\}$ $\{S_1S_3\}$ have a change in common - 1AG. Thus we coalesce.

The sites for the Steiner point are decided as follows

Site	S_1	S_2	S_3	Steiner	
1	A	G	G	G	most common code
2	G	C	G	G	most common code
3	C	C	A	C	most common code
4	G	G	G	G	most common code
5	C	C	C	C	most common code
6	G	G	G	G	most common code

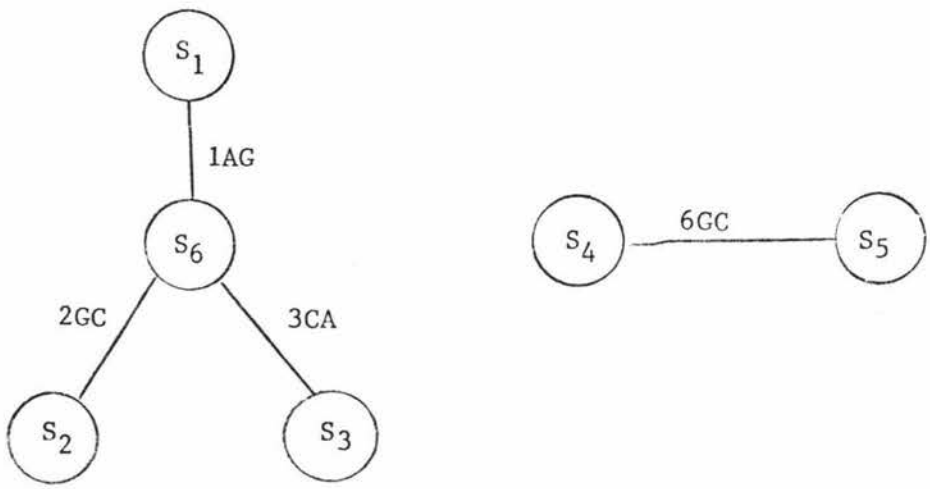
and so the graph becomes



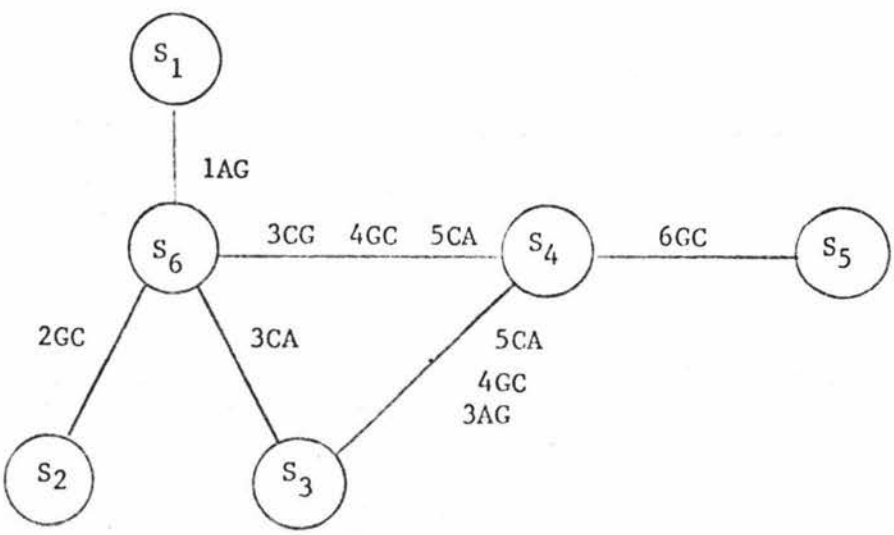
The Steiner point S_6 is included in the species table as GGCGCG and in the difference table as

	S_1	S_2	S_3	S_4	S_5	S_6
S_6	1	1	1	3	4	0

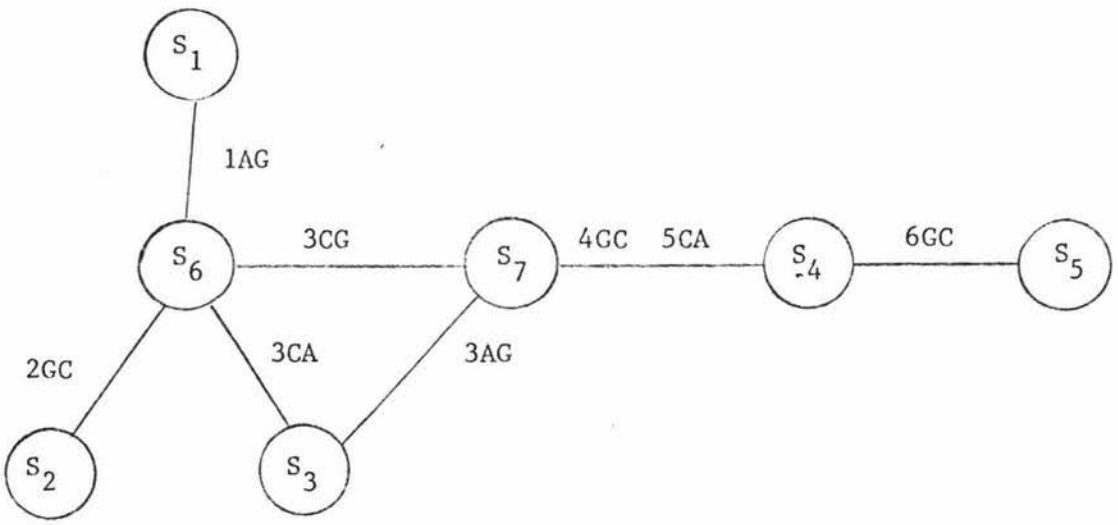
Now we must check for cycles again. One is found when considering the path $<\{S_1S_6\}, \{S_6S_2\}, \{S_2S_3\}, \{S_3S_6\}>$, and the line $\{S_2S_3\}$ is found to be of uniquely greatest weight. Thus it is removed, and the graph becomes



No more coalescing is possible, and the component vector is (1 1 1 4 4). Hence the graph is disconnected and processing continues. So set $\ell = 3$. The graph becomes



The cycle $<S_6,S_3,S_4,S_6>$ is found, but as there is no uniquely greatest weight line in the cycle, so it is left unbroken. Coalescing occurs on the lines $\{S_4S_6\}$ and $\{S_4S_3\}$. The graph becomes



Checking for cycles again, the cycle $<S_6,S_3,S_7,S_6>$ is found, but cannot be broken. No more coalescing is possible, and the component vector is (1 1 1 1 1). Hence the program terminates. Three distinct PSTs are possible, and the choice between them must be made by the program user.

3.2.4 PST2

This program uses similar techniques to those used in PST1 but processes in somewhat of a reverse fashion. Initially, all possible lines are included in the graph, and the program examines the graph repeatedly, searching for and breaking those cycles with a uniquely highest cost line, and coalescing adjacent lines that have common changes. The algorithm is as follows:

Step 1 ..... Create the difference table between species.

Step 2 ..... Set up the graph to contain lines from every point to every

other point.

Step 3 ..... Examine the graph for cycles, removing (if there exists one) the uniquely highest cost line.

Step 4 ..... Examine the graph to find a pair of lines that are adjacent and also have a common change. If such a pair of lines is found, then the program will coalesce to form a Steiner point, and enter lines from this point to all other points in the graph. Then the program proceeds to step 3. If no pair of lines suitable for coalescing were found, then the program terminates.

The graph that results from this program will probably contain one or more cycles, and again some other specific knowledge of the data set processed must be used to determine which lines should be removed.

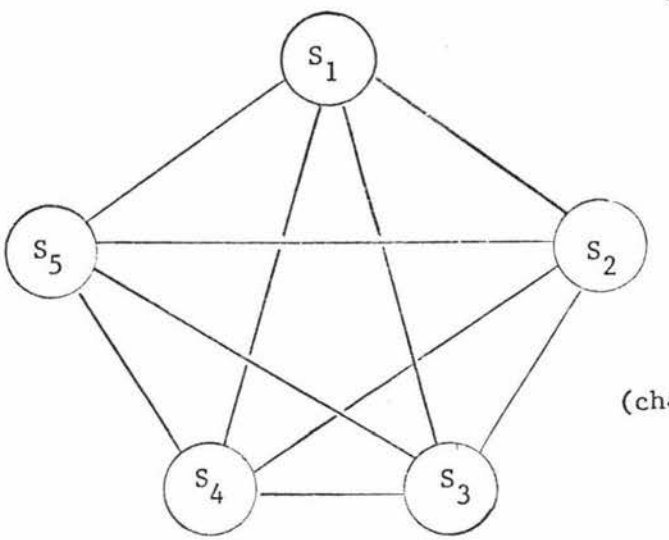
Numerical example of the PST2 method

		Sites					
		1	2	3	4	5	6
	S_1	A	G	C	G	C	G
	S_2	G	C	C	G	C	G
species	S_3	G	G	A	G	C	G
	S_4	G	G	G	C	A	G
	S_5	G	G	G	C	A	C

and we calculate the difference table to be:

	S_1	S_2	S_3	S_4	S_5
S_1	0	2	2	4	5
S_2	2	0	2	4	5
S_3	2	2	0	3	4
S_4	4	4	3	0	1
S_5	5	5	4	1	0

The graph initially is set to:

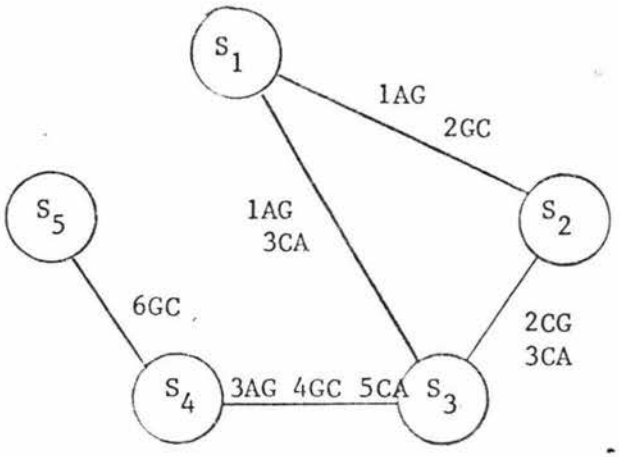


(changes not shown on lines)

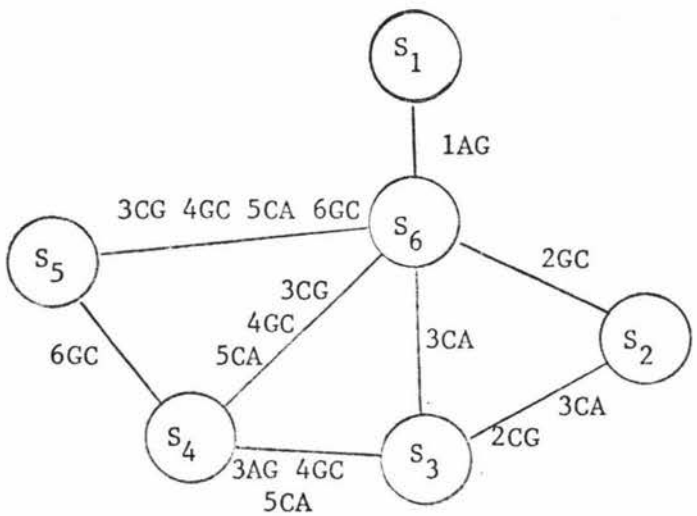
Now the graph is examined and all possible lines within cycles that may be removed are.

$<S_1,S_2,S_3,S_1>$	cannot break
$<S_1,S_2,S_3,S_4,S_1>$	cannot break
$<S_1,S_2,S_3,S_4,S_2>$	remove $\{S_4S_2\}$
$<S_1,S_2,S_3,S_4,S_5,S_1>$	remove $\{S_5S_1\}$
$<S_1,S_2,S_3,S_4,S_5,S_2>$	remove $\{S_5S_2\}$
$<S_1,S_2,S_3,S_4,S_5,S_3>$	remove $\{S_5S_3\}$
$<S_1,S_3,S_2,S_1>$	cannot break
$<S_1,S_3,S_4,S_1>$	remove $\{S_4S_1\}$

That completes all possible cycle breaking, and the graph now looks like:



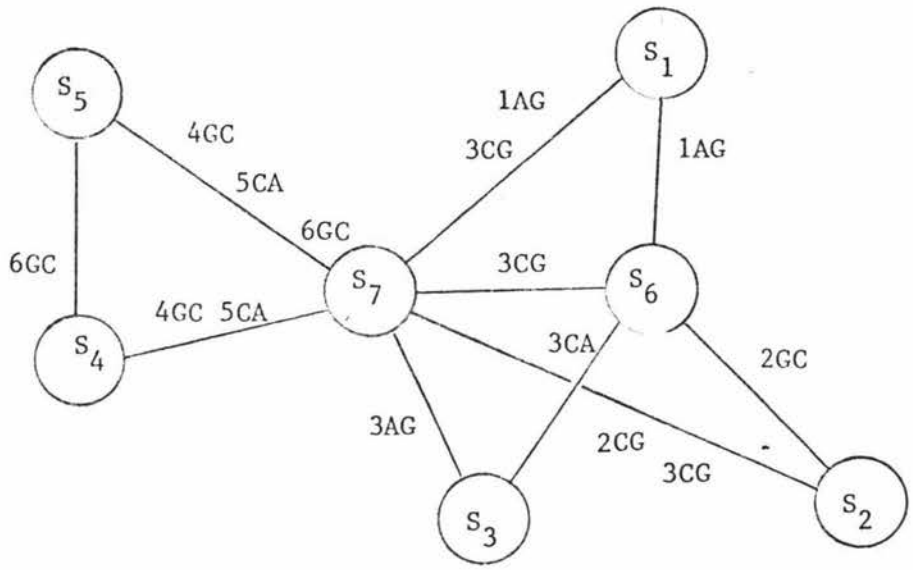
Now we coalesce. Lines $\{S_1S_3\}$ and $\{S_1S_2\}$ have the change 1AG in common. Thus we generate a Steiner point (as in PST1) and link the point to all other points, removing the two lines used for coalescing. Thus our graph looks like:



Now we go back and examine this graph for cycles

$<S_1,S_6,S_2,S_3,S_6>$	remove $\{S_2S_3\}$
$<S_1,S_6,S_4,S_5,S_6>$	remove $\{S_5S_6\}$
$<S_1,S_6,S_3,S_4,S_6>$	cannot break

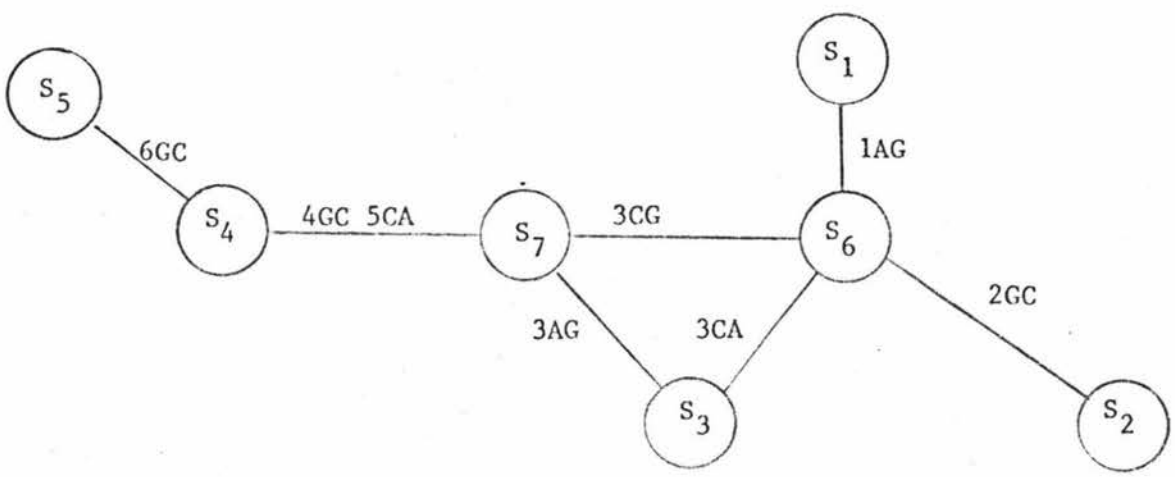
No more cycle removal is possible. Hence we coalesce again in this case using lines $\{S_4S_6\}$ and $\{S_4S_3\}$. Hence we get



Again we go back to the cycle checking stage, and find

$<S_1,S_6,S_2,S_7,S_6>$	remove $\{S_2S_7\}$
$<S_1,S_6,S_3,S_7,S_6>$	cannot break
$<S_1,S_6,S_7,S_1>$	remove $\{S_1S_7\}$
$<S_1,S_7,S_4,S_5,S_7>$	remove $\{S_5S_7\}$

We then check for any more lines to coalesce, and find none. Hence the program terminates, printing the graph:



One of the three lines $\{S_6S_7\}$, $\{S_6S_3\}$, $\{S_7S_3\}$ must be removed by the user.

Thus the program has produced three possible MPSTs each with the optimal total weight of 7.

3.2.5 PST3

This program uses a tree building method similar to that of the Prim [1957] method of generating MSTs, an implementation of the tree building part of the interactive method described in the next section. The method basically chooses a line to enter into the tree initially, and from then on chooses a point from the set of points not yet connected to the tree, and links that point into the tree by the smallest weight line possible. In general, this means that the method will link a point onto an existing line rather than an existing point (the latter is a special case of linking onto a line) and this means that it will generate a Steiner point at the position in the tree line that the external point is linked.

The algorithm is as follows:

Step 1 ..... Generate the difference table between species.

Step 2 ..... By scanning this table, find the line with the greatest weight, and enter that line into the tree initially.

Step 3 ..... Repeat the following steps while there are points not connected into the tree:

- Step 3a ..... Calculate the distance to the tree from every point not yet in the tree. The method of doing this is described below.
- Step 3b ..... Select the nearest of these distances (i.e. the one with lease base changes) and enter that species into the tree.

If required, generate a Steiner point and update the species and difference tables.

Step 3c ..... Go back to step 3 in order to put the next point into the tree.

In order to calculate the number of changes necessary to link an external point into the tree, (loosely referred to as the distance of the point from the tree), the program considers the point in conjunction with every line in the tree. For each of these lines, the two points incident at that line are compared, site by site, with the external point. The number of changes required to link the external point onto that line will then be equal to the number of sites at which the nucleotide for the external point is different from both other points codings.

For example, consider linking the point AGCUA onto the line which connects points GUAUA and GAUUG. Now at the first three sites the external point has differing codes from either of the two connected points. Hence the cost of linking this point onto that line is 3 changes, and this would be:

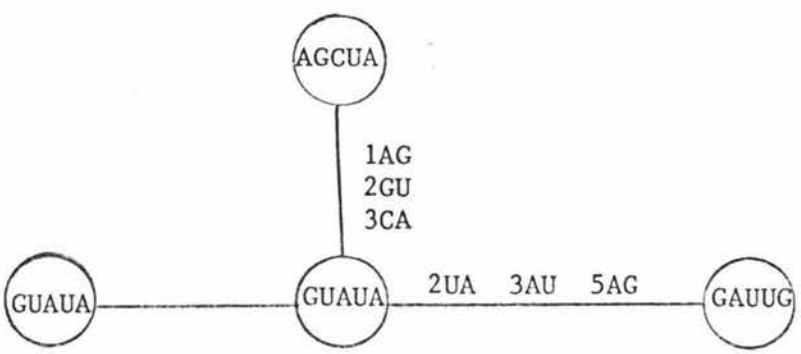


Figure 3.6

Linking onto a line

In this case, the point would link onto the existing tree point GUAUA.

So, to reiterate, this calculation is performed for each line in the tree, and the least number of changes obtained represents the distance to the tree from this external point.

Numerical example of the PST3 method

		Sites					
		1	2	3	4	5	6
	S_1	A	G	C	G	C	G
	S_2	G	C	C	G	C	G
species	S_3	G	G	A	G	C	G
	S_4	G	G	G	C	A	G
	S_5	G	G	G	C	A	C

and the difference table is calculated as

	S_1	S_2	S_3	S_4	S_5
S_1	0	2	2	4	5
S_2	2	0	2	4	5
S_3	2	2	0	3	4
S_4	4	4	3	0	1
S_5	5	5	4	1	0

Initially the tree is:

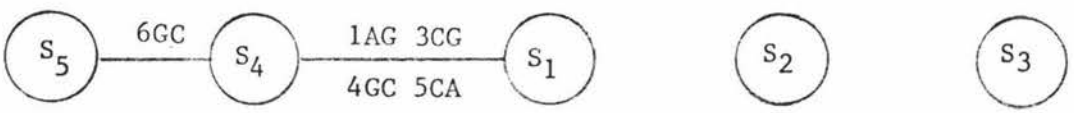
The cost of linking S_2,S_3 and S_4 into the tree need only be calculated for the line $\{S_1S_5\}$, and the distances are:

S_2 ... 1

S_3 ... 1

S_4 ... 0

and hence we choose S_4 as the nearest to the tree, and enter it.



Now we again calculate the distances to the tree for all external points, in this case S_2 and S_3.

The cost of linking both points onto the line $\{S_1S_4\}$ is as follows:

S_2 ... 1

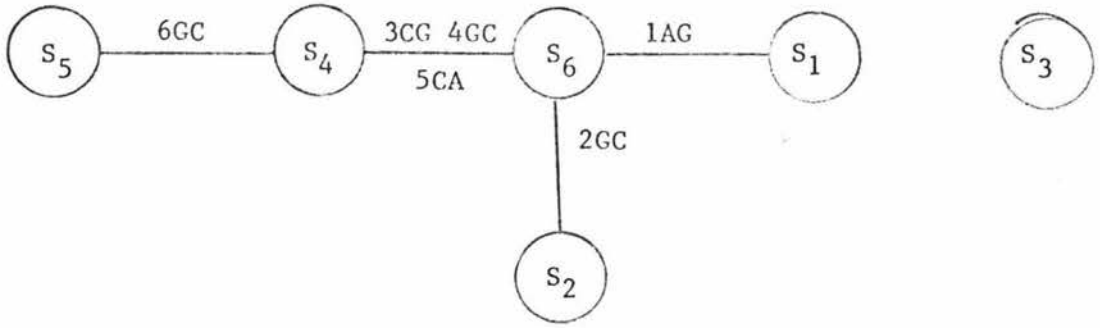
S_3 ... 1

The cost of linking the two points onto the line $\{S_4S_5\}$ is as follows:

S_2 ... 4

S_3 ... 3

Hence we choose S_2 to be entered into the tree on line $\{S_1S_4\}$. The tree then becomes:



Now we look at the possible lines that the remaining point S_3 can be linked to, and calculate the distances involved.

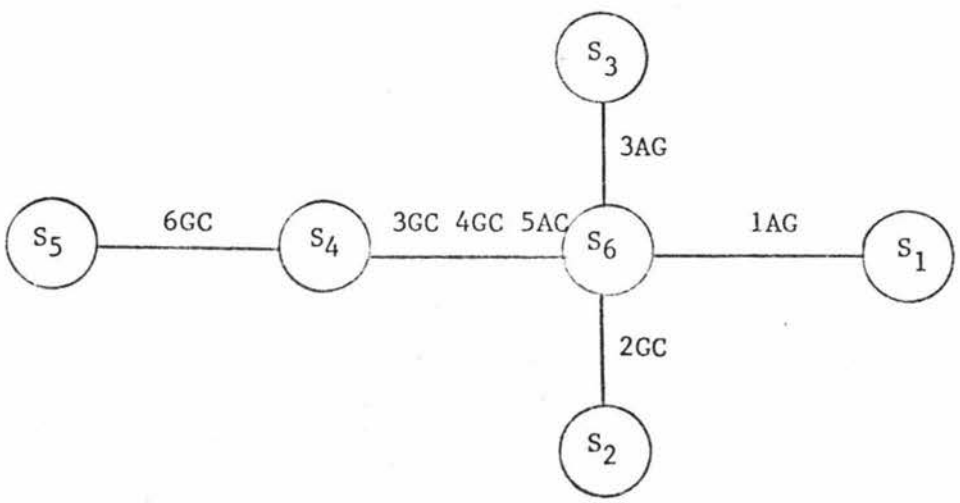
For line $\{S_5S_4\}$, the cost of linking is 3

For line $\{S_4S_6\}$, the cost of linking is 1

For line $\{S_2S_6\}$, the cost of linking is 1

For line $\{S_1S_6\}$, the cost of linking is 1

So we link point S_3 to line $\{S_4S_6\}$ giving the final tree:



and this MPST has a total weight of 7.

3.2.6 An interactive method of tree building

Penny, Hendy and Foulds [1979] have described a method they currently use to generate phylogenies, using a suite of computer programs interactively. They have called the method the π_i approach to generating MPSTs. One of the greatest problems they encountered when generating phylogenies was the analysis of the constructed tree in order to prove minimality. In order to ease this problem, they developed a strategy of growing the tree, one species at a time, and at each growth stage proving that tree is minimal before continuing on to the next stage. The overall algorithm for their method is as follows:

Step 1 ..... Choose a pair of species, and enter the line connecting them into the tree.

Step 2 ..... Select a species not yet in the tree, and link it into the tree (this will be described in detail below).

Step 3 ..... Using the results of the previous iteration, now prove that this tree is optimal.

Step 4 ..... If there are more species to connect to the tree go to step 2; if there are none, then terminate.

They use a naming convention for each of the intermediate trees, and the associated subgroups of sites used to prove that tree minimal (to be described later). At the stage where the tree contains i species, the tree connecting these species is called T_i and the subgroups of sites π_i, hence the naming of the strategy.

We shall leave the analysis of the tree (for proof of optimality) until section 3.3, and continue now to describe in detail the tree construction algorithm. This algorithm is based on the Prim [1957] method of constructing Minimal Spanning Trees for graphs.

The algorithm is as follows:

Step 1 ..... Select the line most likely to represent the central path (see below) in the final phylogeny, and use this line as the initial line in the tree.

Step 2 ..... While there are species outside the tree, repeat the following steps:

Step 2a ..... For each species not in the tree (S_i) calculate the increase in cost of the tree resulting from linking S_i into the tree. This calculation is done by repeating the following steps for each species in the tree (S_j):

(i) Add the line $\{S_iS_j\}$ into the tree.

(ii) Coalesce if possible.

(iii) Record the increase in tree cost.

(iv) Remove the line (or new lines resulting from any coalescing as in step (iii)).

Step 2b ..... Permanently add into the tree the species S_i that gave rise to the least increase in tree cost.

Step 2c ..... Coalesce if possible.

Thus a tree is produced which may now be analysed to test optimality.

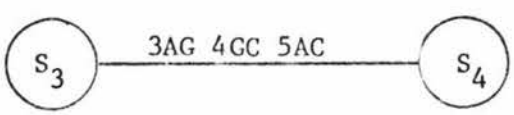
The problem of selecting the initial 'central path' to enter as the first line is not a well defined procedure, but relies heavily on intuition and examination of phylogenies for the same set of species but a different protein.

Numerical example of the interactive method

		Sites					
		1	2	3	4	5	6
species	S_1	A	G	C	G	C	G
	S_2	G	C	C	G	C	G
	S_3	G	G	A	G	C	G
	S_4	G	G	G	C	A	G
	S_5	G	G	G	C	A	C

	S_1	S_2	S_3	S_4	S_5
S_1	0	2	2	4	5
S_2	2	0	2	4	5
S_3	2	2	0	3	4
S_4	4	4	3	0	1
S_5	5	5	4	1	0

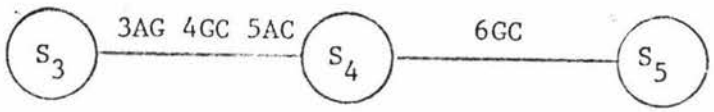
We must select a line we think will be most central. Here, in the absence of any data to help us choose, we arbitrarily select $\{S_3, S_4\}$.



By looking at each of the species S_1, S_2 and S_5, we find the increase in costs of linking these species into the tree are

S_1 : 2
S_2 : 2
S_5 : 1

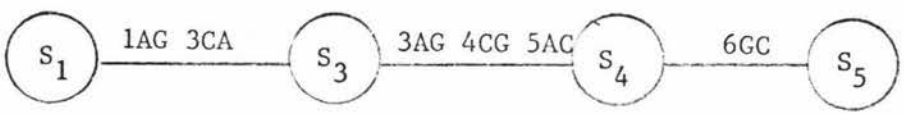
Thus S_5 is entered, giving



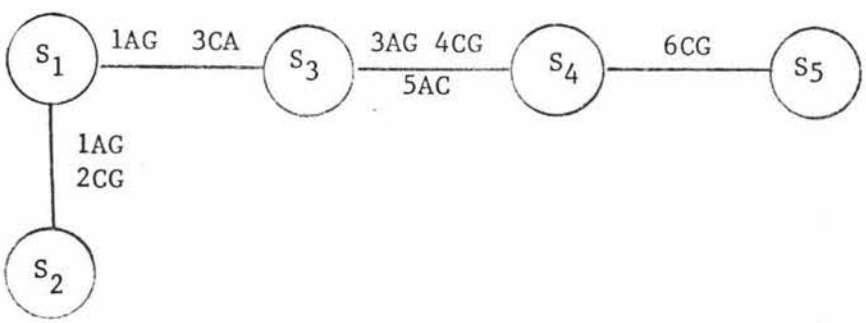
Again, the cost increase incurred in linking the two species S_1 and S_2 into the tree is calculated to be:

$$S_1 : 2$$
$$S_2 : 2$$

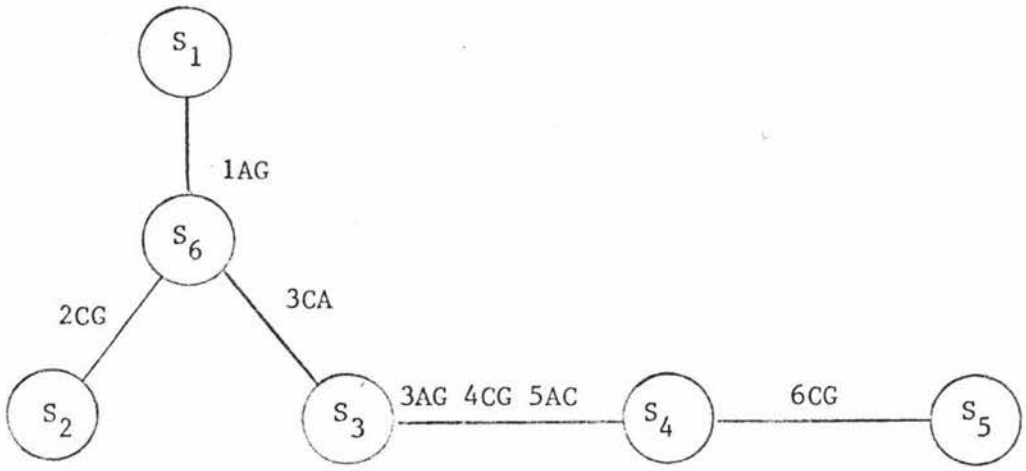
So we choose S_1. The tree then becomes:



Then, finally, the species S_2 is entered, giving



For the first time, coalescing is possible, and a new point is created, linking the species S_1, S_2 and S_3. The tree then is



and the algorithm terminates with a PST of cost 7.

3.2.7 Conclusions

The three programs developed - PST1, PST2 and PST3 - were run on two sets of live data, for which there were PST's available from the Foulds et. al. interactive tree building method. The results are summarised in the following table.

Data	Tree Costs			
	PST1	PST2	PST3	Foulds et. al.
Haemoglobin-α	84	91	80	78
Fibrinopeptide-β	32	33	34	32

Table 3.2

Comparative Phylogeny Costs

As can be seen, the PST's produced by each of the four methods are somewhat inconsistent, in both total cost and topology. This latter inconsistency, especially, seems to be a serious drawback to any practical use of the programs,

even though the PST's produced, although not minimal, are of relatively low cost. However, this is not to say that the methods used in the programs are not viable - they are merely incomplete. For example, in PST2 a MPST may potentially be produced, as all interspecies links exist initially. That one was not produced indicates that the choice of lines to coalesce at each stage, and the lines to delete, is not as simple as currently programmed, and a method of correctly ordering these operations must be produced. There are similar arbitrary decisions in the other methods modelled by PST1 and PST3.

To sum up, then, the programs do not generate MPST's, and further development of the programs to produce MPST's will require a more complete theoretical base for their algorithms.

3.3 Proving the phylogeny optimal

The method used to prove the tree optimal in effect involves finding a lower bound on the tree which is equal to the total weight of the built tree. Then, by the definition of a lower bound, the tree must be optimal. However, using this approach it is interesting to note that the tree cannot be proved non-optimal, as a lower bound less than the cost of the tree will mean one of two things - either there is a tree with less total weight than the one currently under examination, or that the lower bound itself is less than the cost of the least weight tree. An initial lower bound can be found as follows. For each site in the data set, count the number of different nucleotide codings that appear. The lower bound is then the sum of these figures, less the number of sites. This lower bound represents the cost of an ideal phylogeny, in which there are no duplicated changes, and this can be shown in the following example.

		Site		
		1	2	3
	S_1	A	G	U
species	S_2	G	G	U
	S_3	G	U	U
	S_4	A	U	U
		A	G	U
		C	U	

(codings at each site)

Table 3.3

An initial lower bound for a phylogeny

Hence the lower bound produced by this initial approach is

$$\text{lower bound} = 2 + 2 + 1 - 3 = 2$$

However we cannot produce a phylogeny for this data set with a cost of 2, and the least cost phylogeny possible is shown below, with a cost of 3.

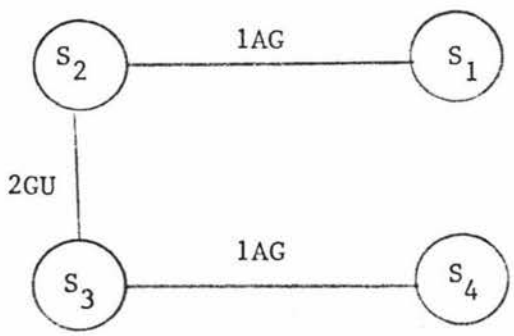


Figure 3.7

The actual phylogeny

The assumption that there will be no duplicated changes in the phylogeny is valid only for the simplest of data sets, and for the purposes of proving phylogenies optimal we need a more sophisticated approach which will take this condition into account.

3.3.1 Partitioning

Hence, a sophisticated lower bounding method has been developed by Hendy, Foulds and Penny [1979], which develops very good bounds, and we now look at this method in detail.

We have been using a table to represent the nucleotide codings at each site, for each species. Let us formally define P as a set of nucleotide sequences, where a nucleotide sequence defines a species, and is an ordered set of nucleotide codes. So

$$P = \{Q\} \qquad \text{where } Q_i \text{ is a nucleotide sequence.}$$

Now let I be the index set for the nucleotide sequences P, i.e. $I = \{1,2, \ldots t\}$ where $|Q| = t$.

A Phylogentic Spanning Tree (PST) of P is any tree $T \subset G = (P,L)$ which spans P. We can define the length of this tree as ℓ, and is the sum of the costs of each $\ell_i \in L$. If we construct all possible trees T, then the tree(s) with minimum length among all spanning trees will be called a Minimal Phylogenetic Spanning Tree (MPST), and the length of this MPST is obviously a function of the set P. This will be referred to as m(P); the measure of P.

The Partition Theorem

Suppose the index set of sites I is partitioned into r subsets $I_1, I_2 \ldots I_r$ with cardinalities $|I_i| = t_i$ so that $\sum_{i=1}^{r} t_i = t$. Then $m(P) \geq \sum_{i=1}^{r} m(P^i)$ where P^i is the subset P consisting of sites only t_i.

This theorem, then, states that a given set of species P can be partitioned into a number of subsets, each containing an entry for each species but having only a disjoint subset of the set of sites, and then the cost of an MPST on the set P will have a lower bound given by the sum of the costs of the MPSTs of the subsets. The proof for this theorem may be found in Hendy et. al. [1979]. This theorem can be shown in the following example.

Let P = {{ACAG}, {CCAU}, {CCUG}} = $\{p_1, p_2, p_3\}$. Then the MPST will be

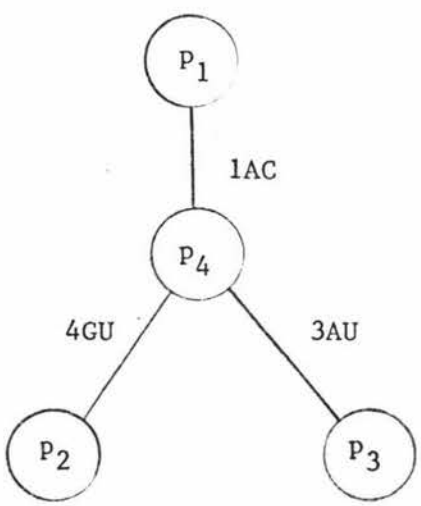
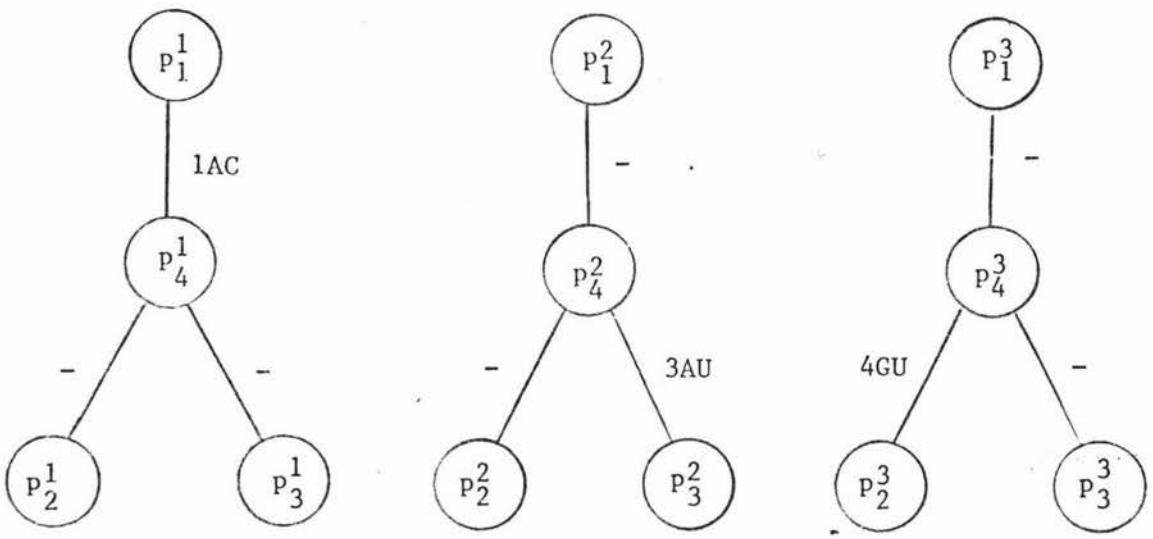


Figure 3.8

A MPST

Now I = {1,2,3,4}, let I_1 = {1,2}, I_2 = {3}, I_3 = {4}. Then the homomorphic copies of the MPST of P will be:



$P^1 = \{\{AC\}, \{CC\}, \{CC\}\}$ $P^2 = \{\{A\}, \{A\}, \{U\}\}$ $P^3 = \{\{G\}, \{I\}, \{G\}$

and so $\ell_1 + \ell_2 + \ell_3 = 1 + 1 + 1 = 3 = \ell$

Figure 3.9

Homomorphic copies of a MPST

Of special interest is the complete partition, where $r = t, I_i = i \ \forall \ i$. In this case we obtain the following lemma.

Lemma 3.1

If there are n_j different nucleotide codes at site j among the $P^i \subset P$, then

$$m(P) \geq \sum_{j=1}^{t} n_j - 1$$

This is the initial bound stated informally at the beginning of this section.

When using this theorem in practice, we will not generally know the topology of the MPST and so will need to use the values $m(P^1)$, $m(P^2)$ etc. So we need to be able to find the cost of the MPST of a set of species P^i. These costs are now classified for up to five species. Further classification is possible, but with an obvious proliferation of possible tree topologies.

Case I, $|P| = 1$

Trivially, one species requires no linking, and so $m(P) = 0$.

Case II, $|P| = 2$

Let $P = \{X,Y\}$. Then $m(P) = d(X,Y)$ where $d(X,Y)$ is the cost of the line connecting X and Y.

Case III, $|P| = 3$

Any MPST will be a homomorphic copy of the following tree (where A is a Steiner point), for $P = \{X,Y,Z\}$.

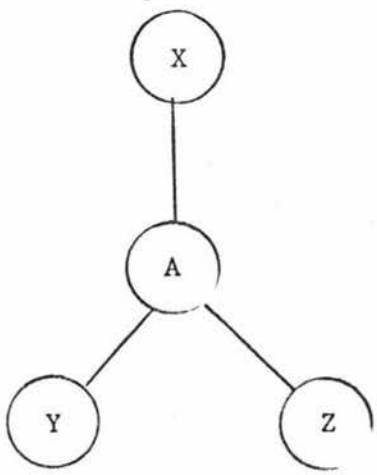


Figure 3.10

The 3 species MPST

One of these lines may have a cost of zero, (i.e. A=X or Y or Z). Suppose at t_1 sites the nucleotide of X, Y and Z are all distinct.

t_2 sites there are two distinct nucleotides for X Y Z

t_3 sites all nucleotides are equal.

So $t_1 + t_2 + t_3 = t$. By lemma 3.2.3.1 $m(P) \geq 2t_1 + t_2$. We can construct a simple PST T, for P, selecting the nucleotides for A so that $A_i = X_i$ when $Y_i \neq Z_i$, and $A_i = Y_i$ otherwise. Thus T will have length $d(A,X) + d(A,Y) + d(A,Z) = 2t_1 + t_2 = \ell \geq m(P)$. Hence $m(P) = 2t_1 + t_2$.

If we let $a = d(X,Y)$, $b = d(X,Z)$, $c = d(Y,Z)$ then

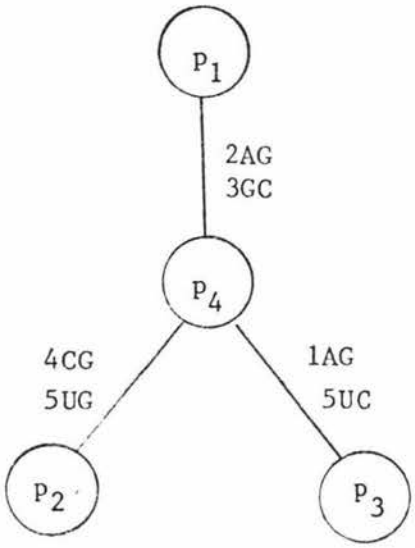
$$d(A,X) = \tfrac{1}{2}(a + b - c - t_1)$$
$$d(A,Y) = \tfrac{1}{2}(a - b + c + t_1)$$
$$d(A,Z) = \tfrac{1}{2}(-a + b + c + t_1)$$
$$\ell = \tfrac{1}{2}(a + b + c + t_1)$$

Example I

$P = \{AAGCU,\ AGCGG,\ GGCCC\} = \{P_1, P_2, P_3\}$

There are five sites:

$$t_1 = 1$$
$$t_2 = 4$$
$$m(P) = 2 + 4 = 6$$



$P_4 = \{AGCCU\}$

Also the difference table is:

	P_1	P_2	P_3
P_1	0	4	4
P_2	4	0	3
P_3	4	3	0

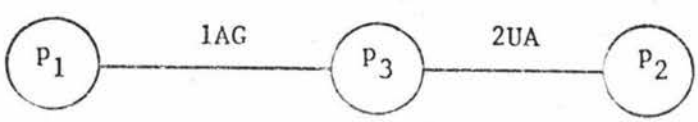
$$m(P) = \tfrac{1}{2}(4 + 4 + 3 + 1) = 6$$

Example II

$$P = \{AUC, GAC, GUC\} = \{P_1, P_2, P_3\}$$

There are three sites:

$$t_1 = 0$$
$$t_2 = 2$$
$$t_3 = 1$$
$$m(P) = 0 + 2 = 2$$



Case IV, $|P| = 4$

Let P = {W,X,Y,Z}. Then any MPST of P will be a homomorphic copy of one of the following trees:

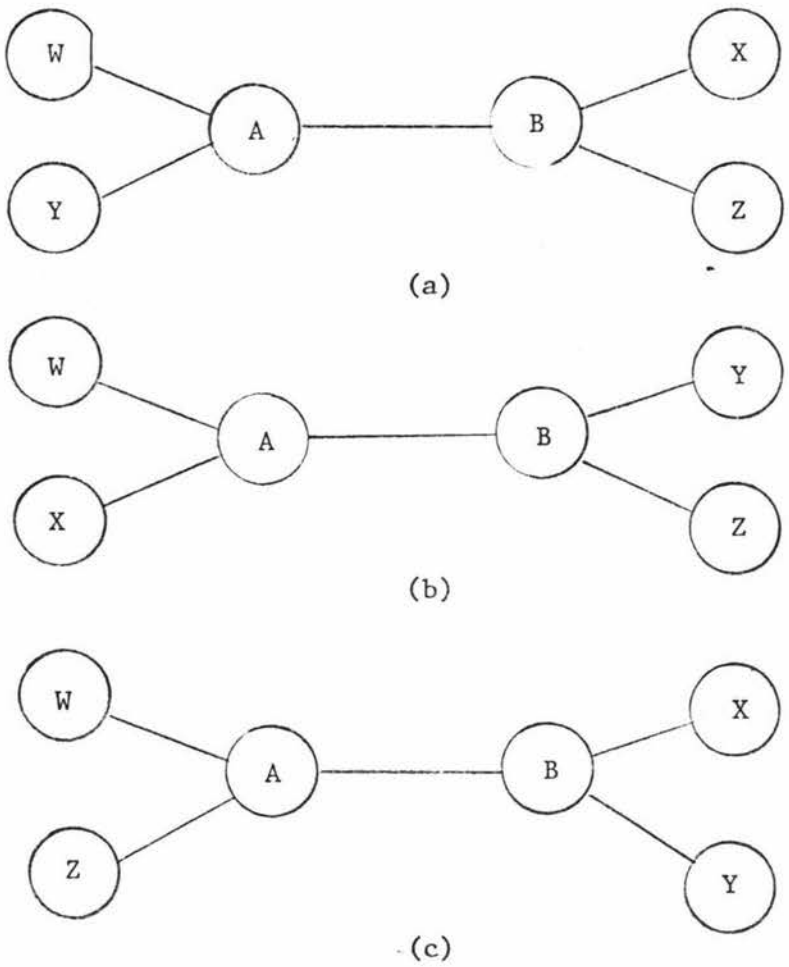


Figure 3.11

The three 4 species MPST's

Suppose that the nucleotide codes at

t_1 sites, $w_i\ x_i\ y_i\ z_i$ are all distinct. (Type I)

t_2 sites, two of $w_i\ x_i\ y_i\ z_i$ are equal, the other two distinct. (Type II)

t_3 sites, $w_i = x_i\ y_i = z_i$. (Type III)

t_4 sites, $w_i = y_i\ x_i = z_i$. (Type IV)

t_5 sites, $w_i = z_i\ x_i = y_i$. (Type V)

t_6 sites, three of w_i x_i y_i z_i are equal. (Type VI)

t_7 sites, all four codes are equal. (Type VII)

By lemma 3.1, $m(P) \geq 3t_1 + 2t_2 + t_3 + t_4 + t_5 + t_6$.

This can be improved. Order W X Y Z so that $t_3 \leq t_4 \leq t_5$. If we collect together the first code of type III, with the first of type IV and the first of type V then we obtain a set of sequences

L' = {W' X' Y' Z} with

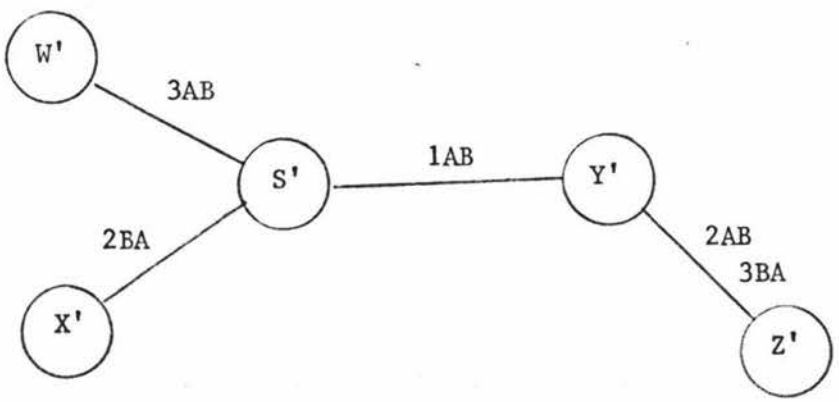
W' = AAA

X' = ABB

Y' = BAB

Z' = BBA

There are t_3 such partitions, with a MPST as follows:



giving m(P') = 5.

Figure 3.12

t_3 type Partitions

For the t_4-t_3 partitions of type IV and type V, we can form
P'' = {W'' X'' Y'' Z''} = {AA AB BA BB}.

This obviously can be spanned by an MPST of cost 3. So we can now find that:

each of the t_1 sites of type I have an MPST of length 3.

each of the t_2 sites of type II have an MPST of length 2.

each of the t_3 sites of type III, IV, V have an MPST of length 5.

each of the t_4-t_3 subsets of types IV, V have an MPST of length 3.

each of the t_5-t_4 sites of type V have an MPST of length 1.

each of the t_6 sites of type VI have an MPST of length 1.

each of the t_7 sites of type VII have an MPST of length 0.

Hence by the partition theorem,

$$m(P) \geq 3t_1 + 2t_2 + 5t_3 + 3(t_4 - t_3) + t_5 + t_6$$
$$= 3t_1 + 2(t_2 + t_3 + t_4) + t_5 + t_6$$

We can now construct a PST of P equivalent to that of Figure 3.11 (c) by choosing codes for A, B as follows:

For sites of type I, IV, V $A_i = B_i = W_i$

For sites of types II, VI, VII $A_i = B_i$ = common values

For sites of type III $A_i = W_i$, $B_i = X_i$

This tree has length $3t_1 + 2(t_2 + t_3 + t_4) + t_5 + t_6$.
So that this is an MPST and

$$m(P) = 3t_1 + 2(t_2 + t_3 + t_4) + t_5 + t_6$$

Example

$$P = \{AAGCU\ AGAGU\ GGCAU\ GGUUC\} = \{p_1\ p_2\ p_3\ p_4\}$$

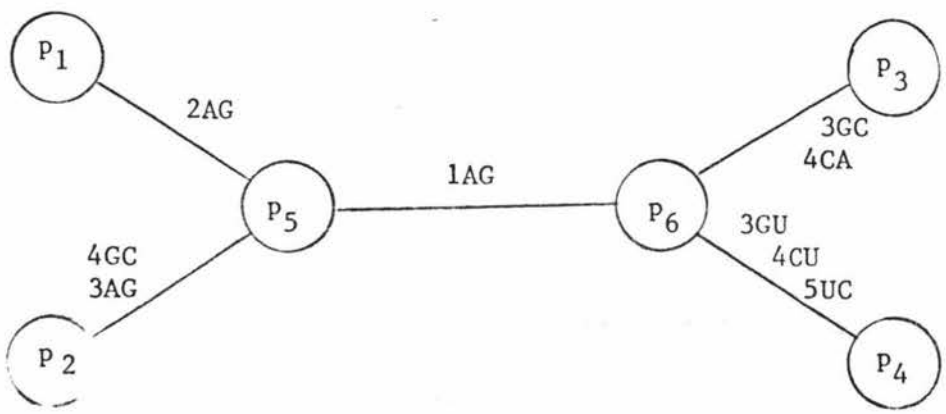
There are five sites and

$$t_1 = 2$$
$$t_2 = 0$$
$$t_3 = 1$$
$$t_4 = 0$$
$$t_5 = 0$$
$$t_6 = 2$$
$$t_7 = 0$$

Reorganising to make $t_3 \leq t_4 \leq t_5$, $P = \{AAGCU, GGCAU, GGUUC, AGAGU\}$

$$m(P) = 3 * 2 + 2(0 + 0 + 0) + 1 + 2 = 9$$

The MPST being



Case V $|P| = 5$

Let $P = \{V,W,X,Y,Z\}$. Then the MPST will be a homomorphic copy of the following tree:

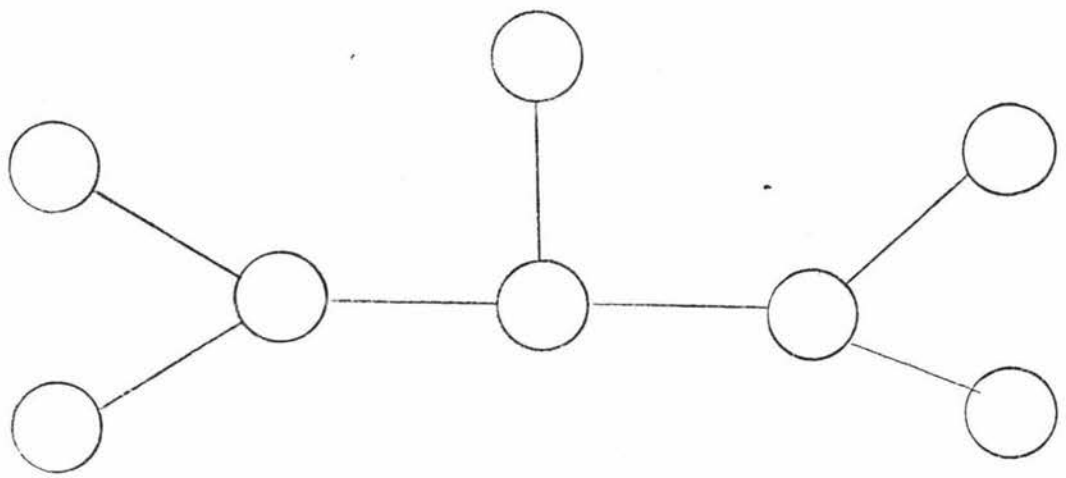


Figure 3.13

The 5 species MPST

For any assignation of species, V, W, X, Y, Z to the five pendant points, we can assign codes to the three Steiner species to minimise the length of the tree. Hence for each of the 15 non-equivalent simple PST's obtained by the different permutations of symbols we can calculate this length. The smallest length will obviously be m(P).

We will use the notation #AABBC to mean that for some permutation of the species V, W, X, Y, Z the codes at the site in question (say site i) are such that $V_i = W_i$, $X_i = Y_i$ and V_i, X_i, Z_i distinct. Then it can be shown that

$$m(L) = 4\ \#ABCDE + 3[\#AABCD + \#AABBC] + 2[\#AAABC + \#AAABB] + \#AAAAB - g_{max}$$

where g_{max} is the largest g, g being defined for each permutation $(C^1, C^2, C^3, C^4, C^5)$ of (V, W, X, Y, Z) as follows:

$g = \#[C_i^1 = C_i^2 \neq C_i^3 = C_i^4]$ for each site i where the pattern of nucleotides can be permuted to AAABB

$+ \#[C_i^1 = C_i^2$ or $C_i^3 = C_i^4]$ for each site i where the pattern of nucleotides can be permuted to AABBC

Example

$P = \{AGGUC\ GGGGC\ CAGGC\ AGUGC\ CGUGA\} = \{p_1\ p_2\ p_3\ p_4\ p_5\}$

* site 1 can be permuted to AACCG

  site 2 can be permuted to GGGGA

* site 3 can be permuted to GGGUU

  site 4 can be permuted to GGGGU

  site 5 can be permuted to CCCCA

Thus $m(P) - 4 * 0 + 3\ [0 + 1] + 2\ [0 + 1] + 3 - g_{max}$

* Now the g values for the combinations of p_1-p_5 are

$\{p_1p_2p_3p_4p_5\} \Rightarrow g = 0 + 0 = 0$

$\{p_1p_2p_3p_5p_4\} \Rightarrow g = 0 + 1 = 1$

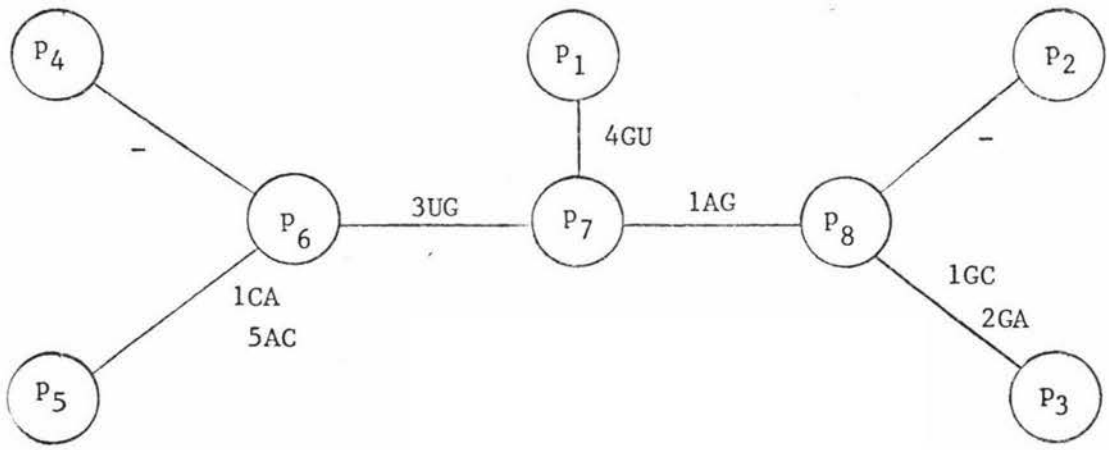
$\{p_1p_2p_4p_3p_5\} \Rightarrow g = 0 + 0 = 0$

$\{p_1p_2p_4p_5p_3\} \Rightarrow g = 1 + 0 = 1$

$\{p_5p_4p_3p_2p_1\} \Rightarrow g = 1 + 0 = 1$

giving a g_{max} of 1, and so $m(P) = 7$

The MPST corresponding to this solution is:



and here $p_4 = p_6$ and $p_2 = p_8$ and p_7 = ACGGC

3.3.2 *ℓ*-clusters

This approach, analysing the possible MPST topologies and calculating a cost formula could be extended further, but with an obvious proliferation of topologically distinct trees and hence terms in the equation for m(P). A method of avoiding this problem has been given by Hendy et. al. [1979], and uses the concept of clustering species together to form a connected component, and then using the partition theorem to connect these clusters.

We define an *ℓ*-cluster as a connected component, connecting those species whose distance apart is $\leq \ell$. For example

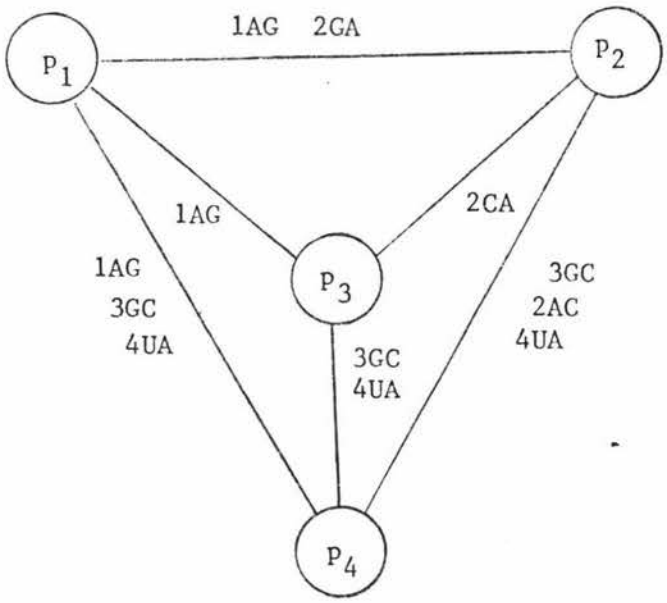


Figure 3.14

A 3-cluster

Then $\{p_1,p_2,p_3\}$ forms a 1-cluster, as the points can be connected into a single connected component using two lines of cost 1, i.e. $\{p_1p_3\}$, $\{p_2p_3\}$.

$\{p_1,p_2,p_3,p_4\}$ forms a 2-cluster, as the points may be connected at best by lines $\{p_1p_3\}$, $\{p_2p_3\}$ and $\{p_3p_4\}$ with line costs all ≤ 2.

This concept of clustering may now be used to calculate the cost of a MPST over larger sets of species. The only cluster of interest is the 1-cluster, as a 1-cluster contains no Steiner points, and can hence be shown to exist in a MPST as the cluster.

The Cluster Theorem

Given a set of nucleotide sequences, C, that can be partitioned into r 1-clusters C_1 .... C_r then there exists a MPST in which all 1-clusters appear.

Proof

Consider two species S_1, S_2 belonging to the same 1-cluster. Now suppose that the MPST does not contain the line $\{S_1S_2\}$. Then S_1, or a subtree whose root is S_1, is connected to the MPST by a line $\{S_1S_i\}$ where S_i does not belong to the same 1-cluster as S_1. But the cost of the line $\{S_1S_i\} \geq$ cost of line $S_1S_2 = 1$. Then the line $\{S_1S_i\}$ may be removed and the line $\{S_1S_2\}$ may be introduced into the MPST. Now by definition, the MPST will still be a tree, and cannot be of greater cost than the original MPST. But now the two species exist as a 1-cluster within the MPST. Obviously this procedure can be repeated for all members of 1-clusters within the MPST.

Using clustering

Any point not belonging to a 1-cluster can itself be considered as a 1-cluster (it is connected to itself by a line of cost 0, which is ≤ 1), and the cost of the MPST calculated by linking together all 1-clusters. This is done as follows. Firstly, consider all ways of linking the clusters without using Steiner points. This will give a cost for a PST of this type. Then consider all ways of linking the points together with the use of just one Steiner point. This will give another cost for the PST, which will replace the previous cost if it is less. Then consider all ways of linking the points together using two Steiner points. Again if this cost is the least found so far, it becomes the current smallest PST cost. This procedure will continue until the all possible tree topologies have been

considered. The following table states the maximum number of Steiner points to consider when looking at a set of 1-clusters.

no of 1-clusters	1	2	3	4	5	6	7	8
max no Steiner points	0	0	1	2	3	4	5	6

Table 3.4

Number of Steiner points

This approach as it has been stated will still cause a large number of possible trees to be considered, and in the actual implementation by Foulds et. al. a lower bounding procedure is used to discard some classes of solutions from consideration, and hence reduce the time taken to produce a cost.

The method can be seen in the following example.

Numerical example

Consider the following set of five species

	Sites				
	1	2	3	4	5
S_1	G	A	U	U	A
S_2	G	A	G	U	R
S_3	A	G	G	U	R
S_4	A	A	G	U	A
S_5	G	G	G	U	R

and the associated difference table

	S_1	S_2	S_3	S_4	S_5
S_1	0	2	4	2	3
S_2	2	0	2	2	1
S_3	4	2	0	2	1
S_4	2	2	2	0	3
S_5	3	1	1	3	0

Thus giving three 1-clusters $\{S_1\}$, $\{S_2S_3S_5\}$, $\{S_4\}$.

These will be denoted C_1, C_2, C_3.

Consider their difference table with the distance between C_i and C_j being defined as:

$$d(C_i,C_j) = \min_{\substack{\forall S_i \in C_i \\ \forall S_j \in C_j}} (d(S_i,S_j))$$

and $d(S_i,S_j)$ being the cost of the line $\{S_iS_j\}$.

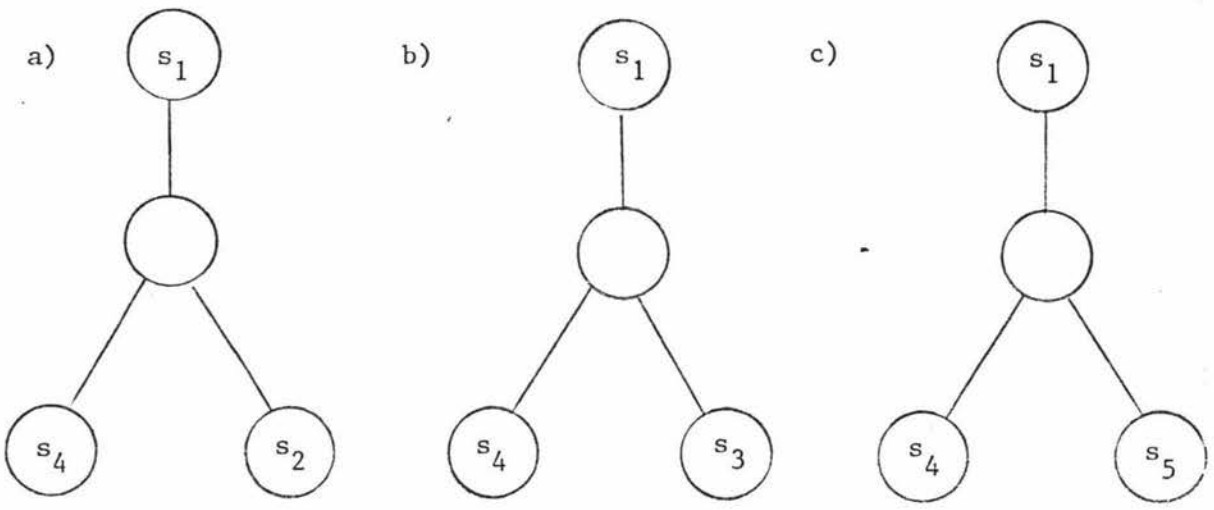
So

	C_1	C_2	C_3
C_1	0	2	2
C_2	2	0	2
C_3	2	2	0

So any connection of these three clusters without a Steiner point will involve at least two lines of cost 2 each, the total PST cost then being the sum of the internal cluster lines plus the sum of the costs of the lines required to link these clusters.

So the cost without Steiner points = 0 + 2 + 0 + 2 + 2 = 6.

Now we consider inserting a Steiner point. There are three possible MPST topologies:



Using the previous partition theorem and subsequent classifications the cost of these three trees can be quickly calculated:

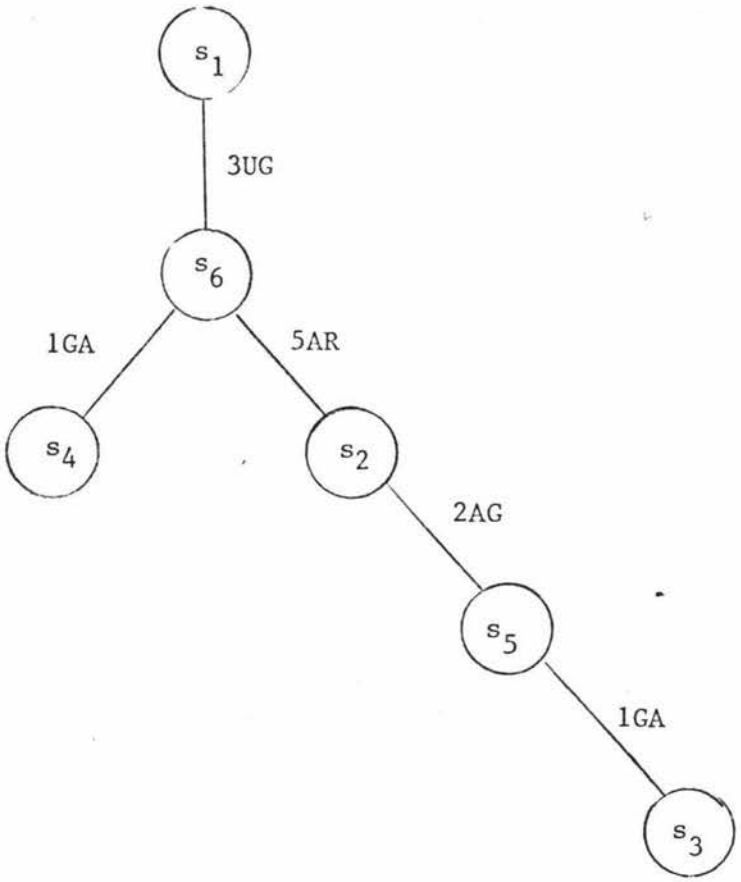
tree (a) cost is 2 * 0 + 3 = 3

(b) cost is 2 * 0 + 4 = 4

(c) cost is 2 * 0 + 4 = 4

Hence the minimal topology using a Steiner point will produce a PST of cost 0 + 2 + 0 + 3 = 5.

For three clusters we do not need to consider any more Steiner points, and so this cost of 5 is the cost of the MPST. The MPST in fact is



with the Steiner point S_6 as GAGUA.

3.3.3 A new approach

The classification of a partition in a data set as described in section 3.3.1 can be extended, with difficulty, to six species. However, as we extend the partition size, the number of possible MPST's that have to be considered by the classification process soars. Researchers at Massey University have found that, using the methods of section 3.3.1, any extension beyond six species in a partition to be out of the question, due to the computing time involved. Hence we now suggest a new approach to this classification process which will allow larger partitions to be handled.

The method, in general terms, examines all possible MPST topologies for the given number of species in the partition, and then generates all possible trees that are homomorphic copies of one of these general trees. Given an

assignment of species to the points in a possible tree, the actual tree cost and Steiner point coding is worked out by considering the tree to be a number of connected minimal subtrees. This makes it relatively easy to build the tree; however, more importantly, we have a good lower bounding facility. For example, suppose in our examination of possible trees we have a tree with a cost of 15 as the best tree found to date, and we are currently examining a tree that is split into two subtrees of costs 8 and 9. Then no matter how we connect these subtrees we cannot produce a final tree with a cost of less than 17. Hence we do not need to bother considering any of the ways of connecting the subtrees.

To give an idea of the work involved in this process, Table 3.5 indicates the number of trees conforming to an MPST topology, for a given number of species in the partition. Obviously, a percentage of these will be discarded by any lower bounding process.

# original points	1	2	3	4	5	6	7	8
# trees	1	1	1	3	15	105	945	10395

Table 3.5 #MPST Candidate Trees

This method is currently being implemented at Massey University by Drs Hendy and Penny, and preliminary results indicate that the method will extend the classification process to ten species. A more complete description of the method and the results obtained is to be produced by Drs Hendy and Penny.

CHAPTER 4

CONCLUSIONS AND FUTURE RESEARCH

4.1 Conclusions

The aims of the thesis were as follows:

(a) To examine the SPG and, if feasible, develop a new solution strategy.

(b) To examine the problem of constructing MPST's and, if feasible, develop a new solution strategy also, using the techniques developed in (a).

The thesis achieved the first aim well, producing a new method of solving the problem that complemented, rather than replaced, existing techniques. The second aim of the thesis was not achieved exactly as initially expected; the emphasis moved from developing an automatic method of constructing trees to an expansion of the method for proving trees optimal. However, as will be appreciated, by increasing the size of a partition the overall effect is to allow larger sets of data to be analysed. Hence progress has been made in both areas considered within the scope of this thesis.

4.2 Future research

Polynomial solution algorithms were not produced for either of the problems considered and the results presented merely push the threshold of practical use of solution methods a little further away. So there is scope for dramatic improvements in both areas, however it is difficult to see exactly how this can be done. One of the main problems with the SPG method as presented is that the lower bounding is tight for only a small number of Steiner points. However, as the problem increases in size, the lower bound increasingly drops below the actual cost of the MST, and thus far more possible solutions have to

fathomed. If this lower bounding could be developed to maintain its tightness as the size of the problem increases, then the branch and bound method could be used on far larger graphs.

As noted before, the Hakimi and Dreyfus and Wagner solutions to the SPG have a resemblance to two of the solutions to the Steiner Problem in Geometry. So, is there a method similar to the Branch and Bound Method developed here that would be applicable to the original Steiner Problem? This, however, is outside the scope of this thesis.

As for the Phylogeny problem, the implicit restrictions on any PST (for example, the triangular inequality holds true) reduce the generality of the problem, and should thus ease the development of a solution. It is not known whether the phylogeny problem is NP-complete, and this knowledge could be useful. Perhaps another approach to the phylogeny problem using matrix operations on the original data could be developed? Thus there is much work to be done in this area still.

GLOSSARY

adjacent	points p_i and p_j are said to be adjacent if the line $\{p_i\ p_j\}$ is present.
binary tree	a tree structure in which a point may have at most two descendant points.
base change	the change in nucleotide coding between two species, at a given site.
branch and bound	a method of solving integer programming problems by dividing the problem into a number of subproblems by the use of constraints on problem variables.
ℓ-cluster	an ℓ-cluster is a connected set of points in which all line costs $\leq \ell$.
coalescing	the act of generating a Steiner point to link three species with the aim of reducing overall cost.
code	the character representing the nucleic acid base for a site.
connected component	a tree connecting a number of points, that may be considered to be a single point in the branch and bound process.
cycle	a path $<p_i \ldots p_j>$ such that $p_i = p_j$ are the only non distinct points.
data set	the nucleotide code sequences for a set of species which forms the data to be used to generate a phylogeny.
decision tree	a graphical representation of the branching decisions made during the branch and bound process.
decomposition	in relation to the Dreyfus and Wagner SPG solution method, the process of breaking a tree into three subtrees.

difference table	a table in which rows and columns represent species, and entries in the table represent the cost of a direct link between the two species.
essential point	a point which must appear in any solution to the SPG for a given graph.
fathom	in relation to the branch and bound process fathoming a subproblem means examining it until a solution is found or it is obvious that no solution better than the current incumbent can be produced from this subproblem.
feasible	a solution is feasible if it contains all essential points in a single component.
graph	a graph G is an ordered pair (P,L), where P is a non empty set of elements called points, and L is a set of unordered pairs of points called lines.
hashing algorithm	a computation on a given data item which will produce an address for storage.
incident	a line is incident to the two points it connects and vice versa.
incumbent	the best feasible solution found to date.
inessential point	a point which may possibly occur in a solution to the SPG.
lower bound	a value less than or equal to the cost of the problem solution.
minimal	in the context of this thesis, the least cost.
MPST	Minimal Phylogenetic Spanning Tree - the least cost phylogenetic spanning tree for the given data set.

MST	Minimal Spanning Tree - the least cost spanning tree for the given graph.
NP-complete	a problem is NP-complete if it (a) can be solved in polynomial time on a non deterministic machine (b) every other NP problem can be polynomially reduced to it.
nucleotide	or nucleic acid base, is the building block for amino acids.
original point	a point representing one of the species which must be spanned by any PST (see essential point).
partial solution	the set of included lines.
partition	a subset of the data set for a phylogeny representing a number of the sites of the data set.
path	the ordered set $\langle p_i, \{p_i,k_1\}, p_{k_1}, \ldots, \{p_{k_n}, p_j\}, p_j \rangle$ of alternating points and lines, where all points are distinct.
pendant point	a point in a graph incident with one and only one line.
phylogeny	a tree in which points represent species and lines represent evolutionary paths.
PST	Phylogenetic Spanning Tree - a phylogeny.
S-tree	a Steiner tree in the plane consisting of four points arranged as three outer points and one internal point, with three lines connecting the internal point to each outer point, these lines being at 120° to each other.
shortest path	the path between two points in which the sum of the line costs in the path is a minimum.

site	the relative position of a nucleotide in a protein. Generally relative to only those other nucleotides under examination.
SMST	the Steiner Minimal Spanning Tree.
SPG	the Steiner Problem in Graphs.
Steiner point	a point generated to represent an intermediate species in the phylogeny, which allows for a lower overall phylogeny cost. Also see inessential point.
tree	a connected graph which contains no cycles.
upper bound	a value greater than or equal to the cost of the minimal solution.
weight	the cost associated with a line.

REFERENCES

Camin, J. H. & Sokal, R. R. A method for deducing branching sequences in phylogeny. Evolution Vol. 19 [1965] 311-326.

Cavalli-Sforza, L. L. & Edwards, A. W. F. Phylogenetic Analysis - models and estimation procedures. Amer. J. Human Genetics Vol. 19, [1967] 233-257.

Chang, S. K. The generation of minimal trees with a Steiner topology. JACM Vol. 19, [1972] 699-711.

Cockayne, E. J. On the efficiency of the algorithm for Steiner Minimal Trees. SIAM, Vol. 18, [1970] 150-159.

Dijkstra, E. W. A note on two problems in connection-with graphs. Numerische Mathematik Vol. 1 [1959], 269-271.

Dreyfus, S. E. An appraisal of some shortest path algorithms. OR Vol. 17 [1969], 221-238.

Dreyfus, S. E. & Wagner, R. A. The Steiner Problem in Graphs. Networks Vol. 1, [1972], 195-207.

Estabrook, G. F. A general solution in partial orders for the Camin-Sokal model in phylogeny. J. Theoret. Biol, Vol. 21, [1968], 421-438.

Farris, J. S. Methods for computing Wagner trees. Systematic Zoology, Vol. 19 [1970], 83-92.

Fitch, W. M. Toward defining the course of evolution. Systematic Zoology, Vol. 20, [1971], 406-416.

Fitch, W. M. On the problem of discovering the most parsimonious tree. American Naturalist, Vol. III, [1977], 223-223

Floyd, R. W. Algorithm 97 - Shortest path. CACM, Vol. 16, [1958], 97-97.

Foulds, L. R., Hendy, M. D. & Penny, D. Solving a problem concerning molecular evolution using the OR approach. NZOR, Vol. 6, [1978], 21-33.

Gilbert, E. N. & Pollak, H. O. Steiner Minimal Trees. SIAM, Vol. 16, [1968], 1-29.

Hakimi, S. L. Steiners Problem in Graphs and its implications. Networks Vol. 1, [1971], 113-133.

Halder, A. K. The method of competing links. [1969], 36-51.

Harary, F. Graph Theory. Published Addison-Wesley, Mass, [1969].

Hartigan, J. A. Minimum mutation fits to a given tree. Biometrics, Vol. 29 [1973], 53-65.

Hendy, M. D., Foulds, L. R. & Penny, D. Proving phylogenetic trees minimal with ℓ-clustering and set partitioning. (Submitted).

Hendy, M. D., Penny, D., & Foulds, L. R. Identification of phylogenetic trees of minimal length. J. Theor. Biol., Vol. 71, [1978], 441-452.

Kevin, V. & Whitney, M. Algorithm 422 - Minimal spanning tree. CACM, Vol. 15, [1972], 273-273.

Kruskal, J. B. On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. Am. Math. Soc., Vol. 7, [1956], 48-50.

Land, A. H. & Doig, A. G. An automatic method of solving discrete programming problems. Econometrica, Vol. 28, [1960], 53-76.

Levin, A. J. Algorithm for the shortest connection of a group of graph vertices. Soviet Math. Dokl., Vol. 12, [1971], 1477-1481.

Loubal, P. S. A network evaluation procedure. Highway Research Record, [1967], 96-109.

Melzak, Z. A. On the problem of Steiner. Can. Math. Bull., Vol. 4, [1961], 143-148.

Penny, D. Criteria for optimizing phylogenetic trees and the problem of determining the root of a tree. J. Mol. Evol., Vol. 8, [1976], 75-116.

Penny, D., Foulds, L. R. & Hendy, M. D. Methods for constructing optimal phylogenetic trees.

Penny, D., Hendy, M. D. & Foulds, L. R. Techniques for the vertification of minimal phylogenetic trees.

Prim, R. C. Shortest connecting networks and some generalizations. Bell Syst. Tech. Jnl., Vol. 36, [1957], 1389-1389.

Shore, M. L., & Foulds, L. R. & Gibbons, P. B. An algorithm for the Steiner Problem in Graphs.

van Oudheusden, D. L. Shortest distances in a Modified transport network.

Weintraub, A. The shortest and the k-shortest routes as assignment problems. Networks, Vol. 3, [1973], 61-73.

APPENDIX I THE BRANCH AND BOUND SPG PROGRAM

(CS0600GIBBONS)CANDE/CODE61 ON PACK

```
 $ SET LIST
%          T H E   S T E I N E R   P R O B L E M   I N   G R A P H S
%
%
%          B R A N C H   A N D   B O U N D   A L G O R I T H M
%
%
%     THIS PROGRAM IS AN IMPLEMENTATION OF THE STEINER PROBLEM IN GRAPHS
%     USING A BRANCH AND BOUND APPROACH.   IT GENERATES ITS OWN GRAPHS
%     RANDOMLY AND SO NEEDS NO GRAPH INPUT. THE RESULTS ARE OUTPUT TO
%     THE LINE PRINTER.  THE BRANCH AND BOUND APPROACH IS IMPLEMENTED AS
%     A DEPTH FIRST STRATEGY, RATHER THAN THE THEORETICAL 'BEST
%     POSSIBILITY' METHOD. THIS IS PURELY TO SAVE MEMORY SPACE.
%     THE PROGRAM USES A COMPONENT VECTOR TECHNIQUE TO KEEP TRACK
%     OF ITS CONNECTED COMPONENTS RATHER THAN PHYSICALLY
%     COALESCING THE POINTS TO FORM A NEW GRAPH.
%
%     THE MAJOR PROGRAM VARIABLES ARE
%
%     M            THE NUMBER OF NECCESSARY POINTS
%     N            THE TOTAL NUMBER OF POINTS
%     D[,]         THE LINE WEIGHT TABLE
%     INCSOL[,]    THE INCUMBENT SOLUTION LINES
%     C[]          THE COMPONENT VECTOR
%     LINKS[,]     THE CURRENT SEARCH ACCEPTED LINES
%     LB           THE LOWER BOUND
%     INC          INCUMBENT SOLUTION TOTAL WEIGHT
%     HIGHLB       LOWER BOUND AT PREVIOUS STEP
%
%
BEGIN

INTEGER M,N;                                                    1
INTEGER K,L;
REAL A;
FILE LP(KIND=PRINTER);

A:=TIME(4)/5;
L:=1;
M:=ENTIER(RANDOM(A)*25+3);
N:=ENTIER(RANDOM(A)*35);
IF M=N THEN N:=N+1;
IF M>N THEN
BEGIN
   K:=M;                                                        2
   M:=N;
   N:=K;
END;
```

```
00000100   000:0000:0
00000200   000:0000:0
00000300   000:0000:0
00000400   000:0000:0
00000500   000:0000:0
00000600   000:0000:0
00000700   000:0000:0
00000800   000:0000:0
00000900   000:0000:0
00001000   000:0000:0
00001100   000:0000:0
00001200   000:0000:0
00001300   000:0000:0
00001400   000:0000:0
00001500   000:0000:0
00001600   000:0000:0
00001700   000:0000:0
00001800   000:0000:0
00001900   000:0000:0
00002000   000:0000:0
00002100   000:0000:0
00002200   000:0000:0
00002300   000:0000:0
00002400   000:0000:0
00002500   000:0000:0
00002600   000:0000:0
00002700   000:0000:0
00002800   000:0000:0
00002900   000:0000:0
00003000   000:0000:0
00003100   000:0000:0
00003200   000:0000:0
B.0000 IS SEGMENT 00003
00003300   003:0000:1
00003400   003:0000:1
00003500   003:0000:1
00003600   003:0000:1
DATA IS 0004 LONG
00003700   003:0000:1
00003800   003:0002:1
00003900   003:0002:5
00004000   003:0006:5
00004100   003:000A:2
00004200   003:000D:0
00004300   003:000D:5
00004400   003:000E:2
00004500   003:000F:2
00004600   003:0010:2
00004700   003:0011:2
```

```
BEGIN                                                                    2    00004900   003:0011:2
INTEGER ARRAY D[1:N,1:N],INCSOL[1:2,1:N],LINKS[1:2,1:N],C[1:N];          2    00005000   003:0011:2
                                                                              B.0001 IS SEGMENT 00006
INTEGER I,J,K,INC,LINES,HIGHLB;                                               00005100   006:000C:3
                                                                              00005200   006:000C:3
%                                                                             00005300   006:000C:3
%                                                                             00005400   006:000C:3
%  SPLIT NODE                                                                 00005500   006:000C:3
%                                                                             00005600   006:000C:3
%  THIS ROUTINE IS GIVEN A GRAPH AND A SET                                    00005700   006:000C:3
%  OF INCLUDED LINES AS PARAMETERS.                                           00005800   006:000C:3
%  IT CHECKS WHETHER THESE LINES FORM A FEASIBLE                              00005900   006:000C:3
%  SOLUTION, AND IF SO WHETHER IT IS A NEW                                    00006000   006:000C:3
%  INCUMBENT   IF NOT IT DECIDES ON WHICH ARC                                 00006100   006:000C:3
%  TO SPLIT THE GRAPH AND RECURSES WITH THE TWO                               00006200   006:000C:3
%  NEW GRAPHS                                                                 00006300   006:000C:3
%                                                                             00006400   006:000C:3
%                                                                             00006500   006:000C:3
                                                                              00006600   006:000C:3
PROCEDURE SPLITNODE(C,HIGHLB);                                                00006700   006:000C:3
VALUE HIGHLB;                                                                 00006800   006:000C:3
INTEGER ARRAY C[*];                                                           00006900   006:000C:3
INTEGER HIGHLB;                                                               00007000   006:000C:3
BEGIN                                                                         00007100   006:000C:3
INTEGER ARRAY ARCS[1:4,1:N],SAVEC[1:N];                                       00007200   006:000C:3
                                                                         SPLITNODE IS SEGMENT 00008
INTEGER I,J,K,MAX,LB,P1,P2,P3;                                           3    00007300   008:0006:0
BOOLEAN FEASIBLE;                                                             00007400   008:0006:0
```

```
(*********************

   CHECK FOR INCUMBENT

   EXAMINES A FEASIBLE SOLUTION AND DECIDES
   WHETHER IT IS OF LOWER COST THAN THE INCUMBENT
   IF SO, IT REPLACES THE INCUMBENT

*********************)

PROCEDURE CHECKFORINCUMBENT;
BEGIN
INTEGER I,J,K,LB;

   LB:=0;
   FOR I:=1 STEP 1 UNTIL LINES DO
   LB:=* + D[LINKS[1,I],LINKS[2,I]];
   IF LB<INC THEN
   BEGIN
        INC:=LB;
        FOR I:=1 STEP 1 UNTIL N-1 DO
        IF I>LINES THEN INCSOL[1,I]:=INCSOL[2,I]:=0 ELSE
        BEGIN
             INCSOL[1,I]:=LINKS[1,I];
             INCSOL[2,I]:=LINKS[2,I];
        END
   END
END;
```

```
                    00007600   008:0006:0
                    00007700   008:0006:0
                    00007800   008:0006:0
                    00007900   008:0006:0
                    00008000   008:0006:0
                    00008100   008:0006:0
                    00008200   008:0006:0
                    00008300   008:0006:0
                    00008400   008:0006:0
                    00008500   008:0006:0
                    00008600   008:0006:0
                    00008700   008:0006:0
                    00008800   008:0006:0
CHECKFORINCUMBENT IS SEGMENT 00009
              4     00008900   009:0000:1
                    00009000   009:0000:5
                    00009100   009:0005:2
                    00009200   009:000A:2
                    00009300   009:000B:1
              5     00009400   009:000B:4
                    00009500   009:000C:4
                    00009600   009:0011:3
                    00009700   009:0016:3
              6     00009800   009:0017:0
                    00009900   009:001A:0
                    00010000   009:001D:0
              6     00010100   009:001D:0
              5     00010200   009:001D:3
CHECKFORINCUMBENT(009) IS 0020 LONG
```

```
%
%
%  LOWER BOUND
%
%  EVALUATES A LOWER BOUND TO A GIVEN GRAPH
%  USING THE FOLLOWING PROCEDURE
%
%    LOWER BOUND = MINIMUM OF
%
%      A)   SUM OF LEAST WEIGHTS BETWEEN EACH
%           NECESSARY POINT AND ALL OTHER POINTS
%
%       B)  SUM OF LEAST WEIGHTS FROM EACH NECESSARY
%           POINT TO EVERY OTHER NECESSARY POINT, LESS
%           THE LEAST OF THESE
%
%
%
INTEGER PROCEDURE LOWERBOUND;
BEGIN
INTEGER ARRAY ARCS[1:M];

INTEGER I,J,K,MIN,LB1,LB;
   LB:=0;
   IF LINES>0 THEN
   FOR I:=1 STEP 1 UNTIL LINES DO
   LB:=* + D[LINKS[1,I],LINKS[2,I]];
   LB1:=LB;
   FOR I:=1 STEP 1 UNTIL M DO ARCS[I]:=1000;
   FOR I:=1 STEP 1 UNTIL N DO
   FOR J:=1 STEP 1 UNTIL N DO
   IF C[I]<=M AND C[I] NEQ C[J] THEN
   BEGIN
      IF D[I,J]<ARCS[C[I]] THEN ARCS[C[I]]:=D[I,J];
   END;
   FOR I:=1 STEP 1 UNTIL M DO
   IF C[I]=I THEN LB:=* + ARCS[I];
   FOR I:=1 STEP 1 UNTIL N DO ARCS[I]:=1000;
   FOR I:=1 STEP 1 UNTIL N DO
   FOR J:=1 STEP 1 UNTIL N DO
   IF C[I]<=M AND C[J]<=M AND C[I] NEQ C[J] THEN
   BEGIN
      IF D[I,J]<ARCS[C[I]] THEN ARCS[C[I]]:=D[I,J];
   END;
   MIN:=1000;
   FOR I:=1 STEP 1 UNTIL M DO
   IF C[I]=I THEN
   BEGIN
      LB1:=* + ARCS[I];
      IF ARCS[I]<MIN THEN MIN:=ARCS[I];
   END;
   LB1:=* - MIN;
   LOWERBOUND:= IF LB<LB1 THEN LB ELSE LB1;
END;
```

```
4    00010400   008:0006:0
     00010500   008:0006:0
     00010600   008:0006:0
     00010700   008:0006:0
     00010800   008:0006:0
     00010900   008:0006:0
     00011000   008:0006:0
     00011100   008:0006:0
     00011200   008:0006:0
     00011300   008:0006:0
     00011400   008:0006:0
     00011500   008:0006:0
     00011600   008:0006:0
     00011700   008:0006:0
     00011800   008:0006:0
     00011900   008:0006:0
     00012000   008:0006:0
     00012100   008:0006:0
     00012200   008:0006:0
     00012300   008:0006:0
     00012400   008:0006:0
     00012500   008:0006:0
LOWERBOUND IS SEGMENT 000A
4    00012600   00A:0002:5
     00012700   00A:0002:5
     00012800   00A:0003:3
     00012900   00A:0004:1
     00013000   00A:0009:1
     00013100   00A:000E:1
     00013200   00A:000F:1
     00013300   00A:0016:0
     00013400   00A:001A:3
     00013500   00A:001F:0
     00013600   00A:0023:2
5    00013700   00A:0023:5
     00013800   00A:0026:0
5    00013900   00A:0029:0
     00014000   00A:0031:3
     00014100   00A:0036:2
     00014200   00A:003D:1
     00014300   00A:0041:4
     00014400   00A:0046:1
     00014500   00A:0048:2
5    00014600   00A:004C:5
     00014700   00A:0055:0
5    00014800   00A:0056:0
     00014900   00A:0057:0
     00015000   00A:005B:3
     00015100   00A:005D:1
5    00015200   00A:005D:4
     00015300   00A:005F:5
     00015400   00A:0063:3
5    00015500   00A:0064:0
     00015600   00A:0065:3
     00015700   00A:0068:4
LOWERBOUND(00A) IS 006F LONG
```

```
PROCEDURE GETLINE;                                                                                                    4   00015900   008:0006:0
BEGIN                                                                                                                     00016000   008:0006:0
%                                                                                                                         00016100   008:0006:0
%    SOLUTION IS TO BE SPLIT INTO TWO SOLUTIONS                                                                           00016200   008:0006:0
%    GET LINE ON WHICH TO BRANCH(GREATEST PENALTY LINE)                                                                   00016300   008:0006:0
%                                                                                                                         00016400   008:0006:0
%    TO DO THIS WE MUST                                                                                                   00016500   008:0006:0
%       GET THE LEAST WEIGHT LINE FROM POINT                                                                              00016600   008:0006:0
%       GET THE NEXT LEAST COST LINE FROM THE SAME POINT.                                                                 00016700   008:0006:0
%       CALCULATE WEIGHT DIFFERENCE AND STORE IT                                                                          00016800   008:0006:0
%                                                                                                                         00016900   008:0006:0
%     ONCE ALL NECESSARY POINTS HAVE BEEN LOOKED AT, CHOOSE                                                               00017000   008:0006:0
%     CHOOSE THE MAXIMUM PENALTY LINE AS THE LINE WHICH                                                                   00017100   008:0006:0
%     CAUSES THE HIGHEST WEIGHT PENALTY IF IT IS REMOVED                                                                  00017200   008:0006:0
%                                                                                                                         00017300   008:0006:0
                                                                                                                          00017400   008:0006:0
                                                                                                                          00017500   008:0006:0
        FOR I:=1 STEP 1 UNTIL N DO                                                                                    4   00017600   008:0006:0
        FOR J:=1 STEP 1 UNTIL N DO                                                                                        00017700   008:000F:3
        IF C[I] NEQ C[J] AND C[I]<=M THEN                                                                                 00017800   008:0013:5
        BEGIN                                                                                                         5   00017900   008:0014:2
            IF D[I,J]<ARCS[3,C[I]] THEN                                                                                   00018000   008:0018:4
            BEGIN                                                                                                     6   00018100   008:0019:1
                ARCS[1,C[I]]:=I;                                                                                          00018200   008:001C:1
                ARCS[2,C[I]]:=J;                                                                                          00018300   008:001F:1
                ARCS[3,C[I]]:=D[I,J];                                                                                     00018400   008:0023:4
            END;                                                                                                      6   00018500   008:0023:4
        END;                                                                                                          5   00018600   008:0024:4
        FOR I:=1 STEP 1 UNTIL N DO                                                                                        00018700   008:0029:1
        FOR J:=1 STEP 1 UNTIL N DO                                                                                        00018800   008:002D:4
        IF C[I] NEQ C[J] AND C[I]<=M THEN                                                                                 00018900   008:0032:0
        BEGIN                                                                                                         5   00019000   008:0032:3
            IF D[I,J]<ARCS[4,C[I]] THEN                                                                                   00019100   008:0036:5
            BEGIN                                                                                                     6   00019200   008:0037:2
                IF I NEQ ARCS[1,C[I]] OR J NEQ ARCS[2,C[I]] THEN                                                          00019300   008:003D:1
                ARCS[4,C[I]]:=D[I,J];                                                                                     00019400   008:0042:1
            END;                                                                                                      6   00019500   008:0042:1
        END;                                                                                                          5   00019600   008:0043:1
        FOR I:=1 STEP 1 UNTIL M DO                                                                                        00019700   008:0047:4
        IF ARCS[1,I] NEQ 0 THEN ARCS[4,I]:=* - ARCS[3,I];                                                                 00019800   008:004E:0
        MAX:= -1;                                                                                                         00019900   008:004E:5
        FOR I:=1 STEP 1 UNTIL M DO                                                                                        00020000   008:0053:2
        IF ARCS[1,I] NEQ 0 AND ARCS[4,I]>MAX THEN                                                                         00020100   008:0056:5
        BEGIN                                                                                                         5   00020200   008:0057:2
            MAX:=ARCS[4,I];                                                                                               00020300   008:0059:2
            P1:=I;                                                                                                        00020400   008:005A:2
        END;                                                                                                          5   00020500   008:005A:5
END;
```

```
%                                                                        4   008:005B:0
%                                                                            008:005B:0
   FEASIBLE:=TRUE;                                                           008:005B:0
   FOR I:=1 STEP 1 UNTIL M DO                                                008:005B:4
   BEGIN                                                                     008:0060:1
        ARCS[1,I]:=ARCS[2,I]:=0;                                         4   008:0060:1
        ARCS[3,I]:=ARCS[4,I]:=1000;                                          008:0063:5
        IF C[I] NEQ 1 THEN FEASIBLE:=FALSE;                                  008:0066:1
   END;                                                                      008:006A:5
   IF FEASIBLE THEN CHECKFORINCUMBENT ELSE                               4   008:006B:2
%                                                                            008:006C:5
%   THIS SOLUTION IS NOT FEASIBLE AND SO WE CHECK                            008:006C:5
%   WHETHER IT HAS A LOWER BOUND < INCUMBENT COST                            008:006C:5
%                                                                            008:006C:5
   BEGIN                                                                     008:006C:5
        LB:=LOWERBOUND;                                                  4   008:006D:2
        IF HIGHLB>LB THEN LB:=HIGHLB;                                        008:006E:4
        IF LB<INC THEN                                                       008:0071:0
        BEGIN                                                                008:0071:5
             GETLINE;                                                    5   008:0072:2
             IF MAX NEQ -1 THEN                                              008:0073:0
             BEGIN                                                           008:0073:5
                  FOR I:=1 STEP 1 UNTIL N DO SAVEC[I]:=C[I];             6   008:0074:2
                  P2:= IF C[ARCS[1,P1]]<C[ARCS[2,P1]] THEN                   008:007C:0
                       C[ARCS[1,P1]] ELSE C[ARCS[2,P1]];                     008:0080:1
                  P3:= IF C[ARCS[1,P1]]>C[ARCS[2,P1]] THEN                   008:0085:5
                       C[ARCS[1,P1]] ELSE C[ARCS[2,P1]];                     008:008A:0
                  FOR I:=1 STEP 1 UNTIL N DO                                 008:008F:4
                  IF C[I]=P3 THEN C[I]:=P2;                                  008:0094:1
                  LINES:=* + 1;                                              008:0098:5
                  LINKS[1,LINES]:=ARCS[1,P1];                                008:009A:1
                  LINKS[2,LINES]:=ARCS[2,P1];                                008:009D:1
                  SPLITNODE(C,LB);                                           008:00A0:1
                  FOR I:=1 STEP 1 UNTIL N DO C[I]:=SAVEC[I];                 008:00A2:1
                  LINKS[1,LINES]:=LINKS[2,LINES]:=0;                         008:00A9:5
                  LINES:=* - 1;                                              008:00AD:3
                  P2:=D[ARCS[1,P1],ARCS[2,P1]];                              008:00AE:5
                  D[ARCS[1,P1],ARCS[2,P1]]:=D[ARCS[2,P1],ARCS[1,P1]]:=1000;  008:00B2:5
                  SPLITNODE(C,LB);                                           008:00BB:1
                  D[ARCS[1,P1],ARCS[2,P1]]:=D[ARCS[2,P1],ARCS[1,P1]]:=P2;    008:00BD:1
                  FOR I:=1 STEP 1 UNTIL N DO C[I]:=SAVEC[I];                 008:00C5:3
             END                                                             008:00CD:1
        END                                                              6   008:00CD:1
   END                                                                   5   008:00CD:1
END;                                                                     4   008:00CD:1
                                                          SPLITNODE(008) IS 00D8 LONG
```

```
%
%%%%
%
    WRITE(LP,</,"GRAPH SIZE ",I4>,N);
    WRITE(LP,</,"NECC NODES ",I4>,M);
%
%%%%  SET UP A SKELETON TREE OF NECESSARY POINTS
%%%%  AND THEN ADD LINES IN TO FORM THE GRAPH
%
    FOR I:=1 STEP 1 UNTIL N DO
    FOR J:=1 STEP 1 UNTIL N DO D[I,J]:=0;
    D[1,2]:=D[2,1]:=ENTIER(RANDOM(A)*8);
    FOR I:=2 STEP 1 UNTIL M DO
    BEGIN
    K:=ENTIER(RANDOM(A)*(I-1));
    IF K=0 THEN K:=1;
    D[I,K]:=D[K,I]:=ENTIER(RANDOM(A)*8);
    END;
    WRITE(LP,</,"GENERATED GRAPH">);
    J:=ENTIER(RANDOM(A)*N*N);
    FOR I:=1 STEP 1 UNTIL J DO
    BEGIN
       K:=ENTIER(RANDOM(A)*N);
       L:=ENTIER(RANDOM(A)*N);
       IF K=0 THEN K:=N;
       IF L=0 THEN L:=N;
       IF K NEQ L THEN D[K,L]:=D[L,K]:=ENTIER(RANDOM(A)*8+0.5);
    END;
    FOR I:=1 STEP 1 UNTIL N DO
    WRITE(LP,</,*I3>,N,FOR J:=1 STEP 1 UNTIL N DO D[I,J]);
    FOR I:=1 STEP 1 UNTIL N DO
    FOR J:=I STEP 1 UNTIL N DO
    IF D[I,J]=0 THEN D[I,J]:=D[J,I]:=1000;
%
%%%%  INITIALIZE PROGRAM VARIABLES
%
    LINES:=0;
    INC:=1000;
    FOR I:=1 STEP 1 UNTIL N DO
    BEGIN
        C[I]:=I;
        INCSOL[1,I]:=INCSOL[2,I]:=0;
        LINKS[1,I]:=LINKS[2,I]:=0;
    END;
    I:=TIME(2);
    SPLITNODE(C,0);
    J:=TIME(2);
    I:=(J-I)/60;
    WRITE(LP,</,"TIME TO PROCESS ",I4>,I);
    IF INC>999 THEN WRITE(LP,</,"GRAPH DISCONNECTED">)
    ELSE WRITE(LP,</,"COST OF MST IS ",I4>,INC);
    FOR I:=1 STEP 1 UNTIL N DO
    IF INCSOL[1,I]=0 THEN I:=N ELSE
     WRITE(LP,</,2I4>,INCSOL[1,I],INCSOL[2,I]);
END;

END.
```

```
3    00025400   006:000C:3
     00025500   006:000C:3
     00025600   006:000C:3
     00025700   006:000C:3
     00025800   006:000C:3
     00025900   006:0013:2
     00026000   006:001A:2
     00026100   006:001A:2
     00026200   006:001A:2
     00026300   006:001A:2
     00026400   006:001A:2
     00026500   006:001E:5
     00026600   006:0026:4
     00026700   006:002C:1
     00026800   006:0030:5
3    00026900   006:0030:5
     00027000   006:0035:0
     00027100   006:0036:3
     00027200   006:003E:0
3    00027300   006:003E:3
     00027400   006:0043:2
     00027500   006:0047:2
     00027600   006:004B:5
3    00027700   006:004B:5
     00027800   006:004F:2
     00027900   006:0052:5
     00028000   006:0055:0
     00028100   006:0057:1
     00028200   006:0061:0
3    00028300   006:0061:3
     00028400   006:0066:0
     00028500   006:0075:5
     00028600   006:007A:2
     00028700   006:007E:3
     00028800   006:0087:0
     00028900   006:0087:0
     00029000   006:0087:0
     00029100   006:0087:0
     00029200   006:0087:4
     00029300   006:0088:4
     00029400   006:008D:1
3    00029500   006:008D:1
     00029600   006:008F:0
     00029700   006:0092:4
     00029800   006:0096:2
3    00029900   006:0096:5
     00030000   006:0096:3
     00030100   006:009A:1
     00030200   006:009B:5
     00030300   006:009D:3
     00030400   006:00A4:2
     00030500   006:00A8:5
     00030600   006:00B2:2
     00030700   006:00B6:5
     00030800   006:00B9:5
     00030900   006:00C6:5
   B.0001(006)  IS 000A LONG
2    00031000   003:0012:0
   B.0000(003)  IS 001B LONG
                DATA IS 0034 LONG
```

APPENDIX II ACTUAL PHYLOGENY DATA AND RESULTS

This appendix contains the nucleotide codings for a set of ten species, for two proteins.

The species are

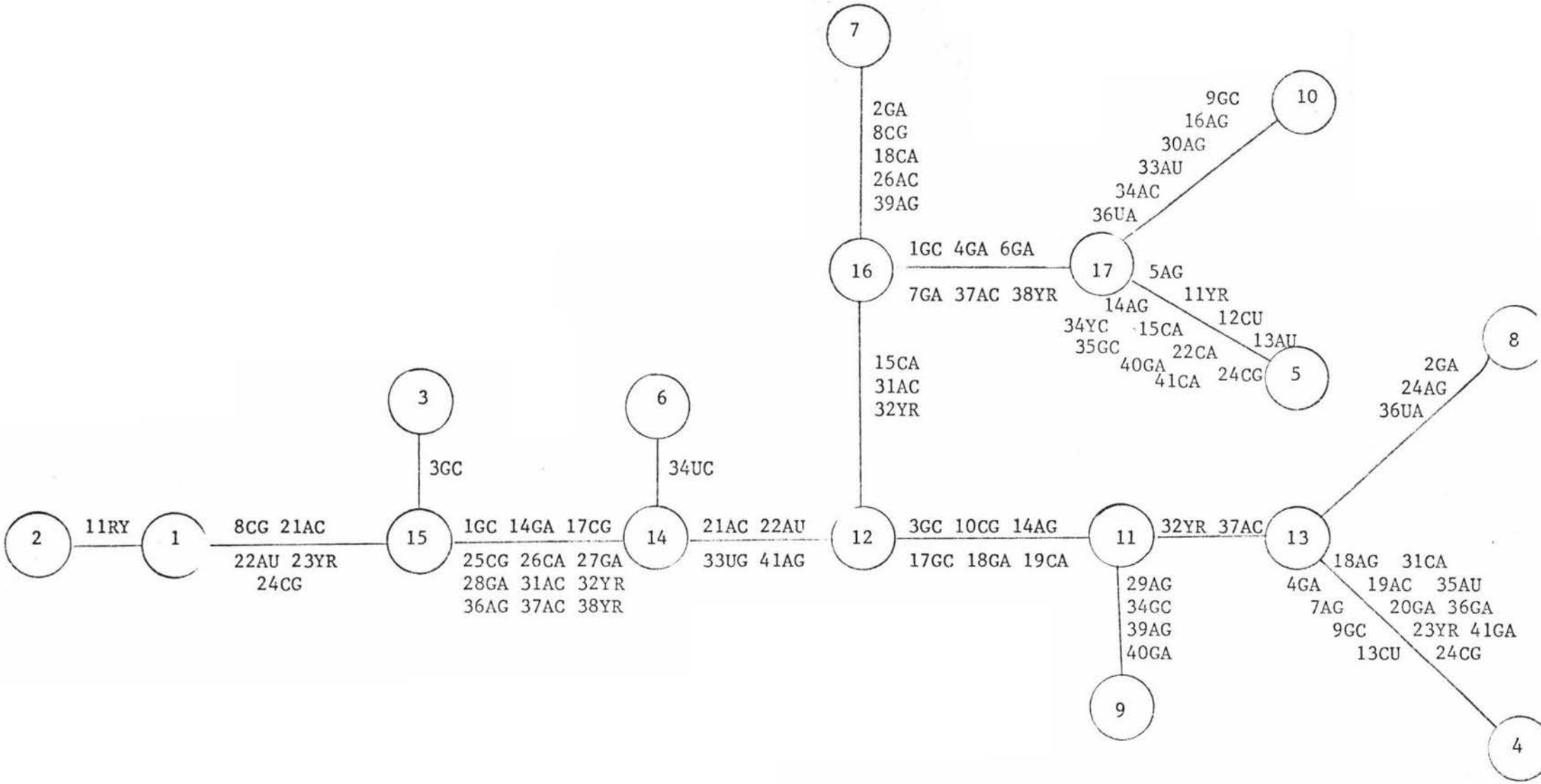
S_1	man
S_2	ape
S_3	monkey(mky)
S_4	rabbit(rab)
S_5	mouse(mus)
S_6	dog
S_7	horse(hos)
S_8	pig
S_9	cow
S_{10}	ewe

The two proteins from which the data sets have been taken are haemoglobin-α and fibrinopeptide-β.

For each of the data sets this appendix gives the actual data set used, the tree as generated by PST1, the tree as generated by PST2, the tree as generated by PST3 and the tree as given by Foulds et. al. using the interactive tree building process in the π_i approach.

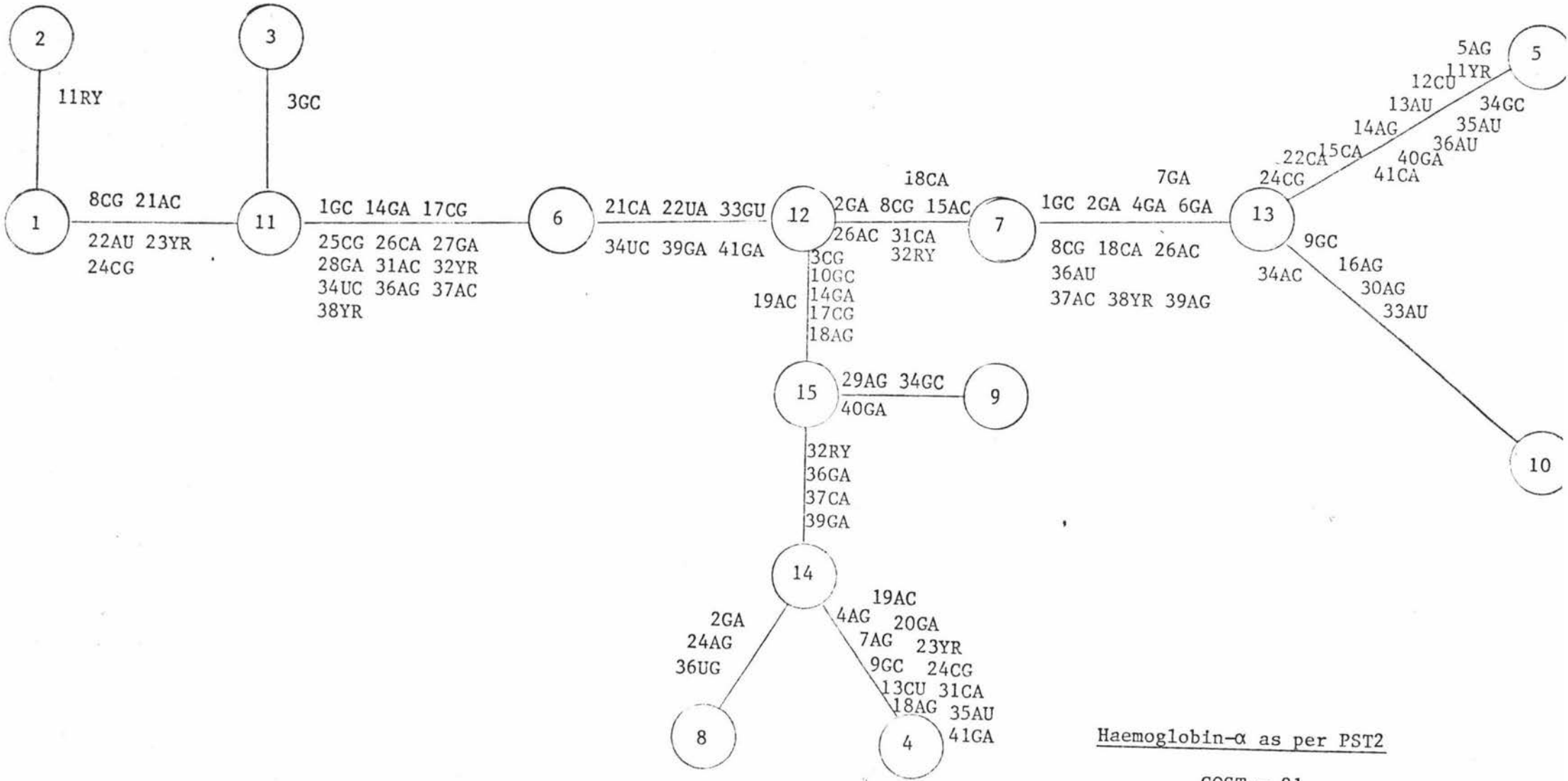
Haemoglobin-α nucleotide codings

	Site 1	2	3	4
	1234567890123456789012345678901234567890 1			
MAN	CACGGGGCCGRUUACGGAAAAAYCGAAAGGCRGCUGCRGAG			
APE	CACGGGGCCGYUUACGGAAAAAYCGAAAGGCRGCUGCRGAG			
MKY	CAGGGGGGCGRUUACGGAAACURGGAAAGGCRGCUGCRGAG			
RAB	GAGAGGAGGCRUCACGGAAGAAYCCCGGGGCRUCAGCYGAG			
MUS	CACAAAAGCGYCAACGCAAAACRCCCGGGGCRUGAACRGGC			
DOG	GACGGGGGCGRUUGCGCAAACURGCCGGGGAYGUUAAYGAG			
HOS	GGCGGGGCCGRUUGAGCCAAAARGCAGGGGCRUCUAAYAAA			
PIG	GGGGGGGGCCRUUACGGGCAAARACCGGGGARUCUUCYGAA			
COW	GAGGGGGGCCRUUACGGGCAAARGCCGGAGAYUGUAAYAGA			
EWE	CACAGAAGGGRUUGAACAAAAARGCCGGGACRAAUUCRGAA			



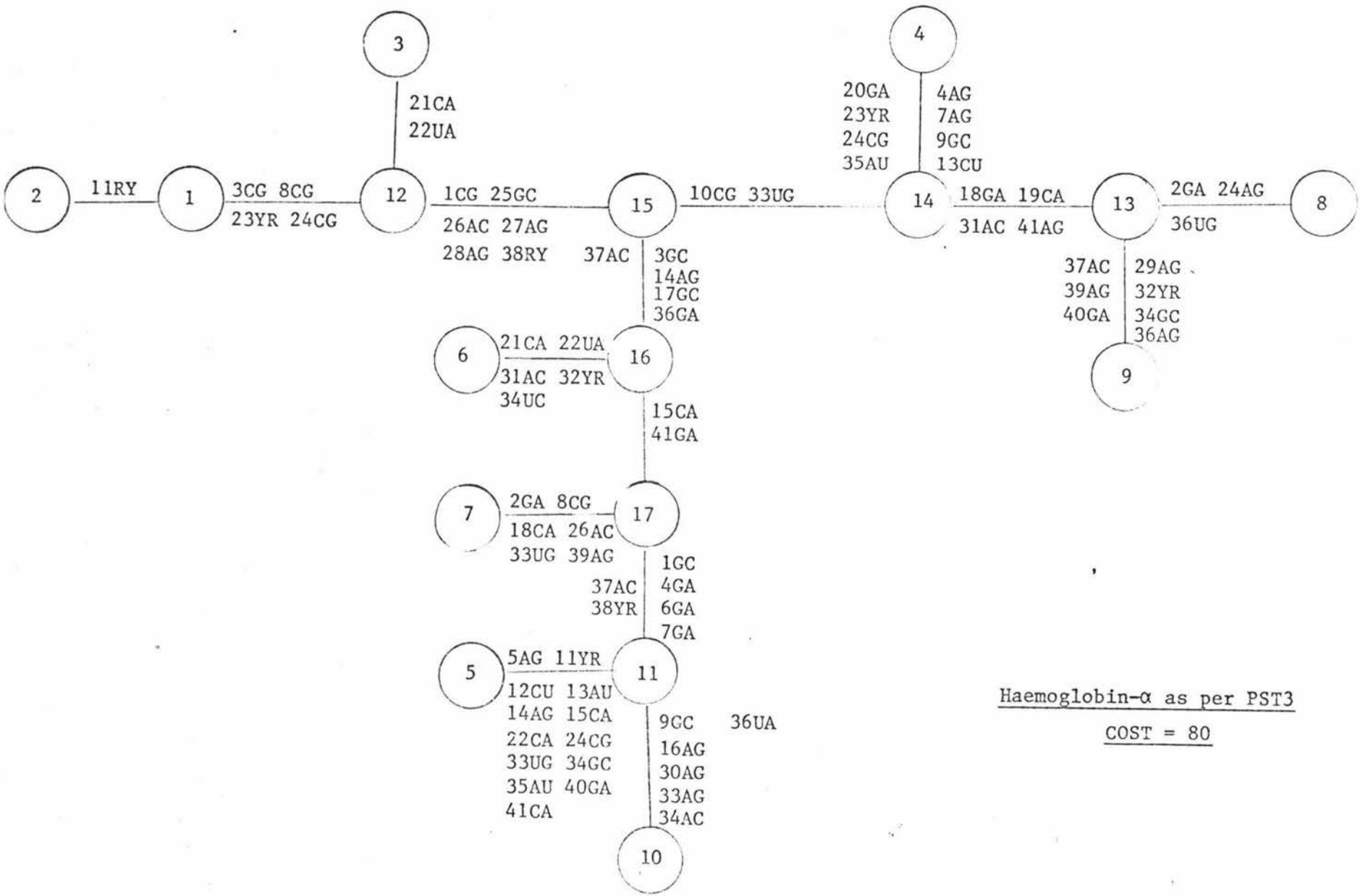
Haemoglobin-α as per PST1

COST = 84



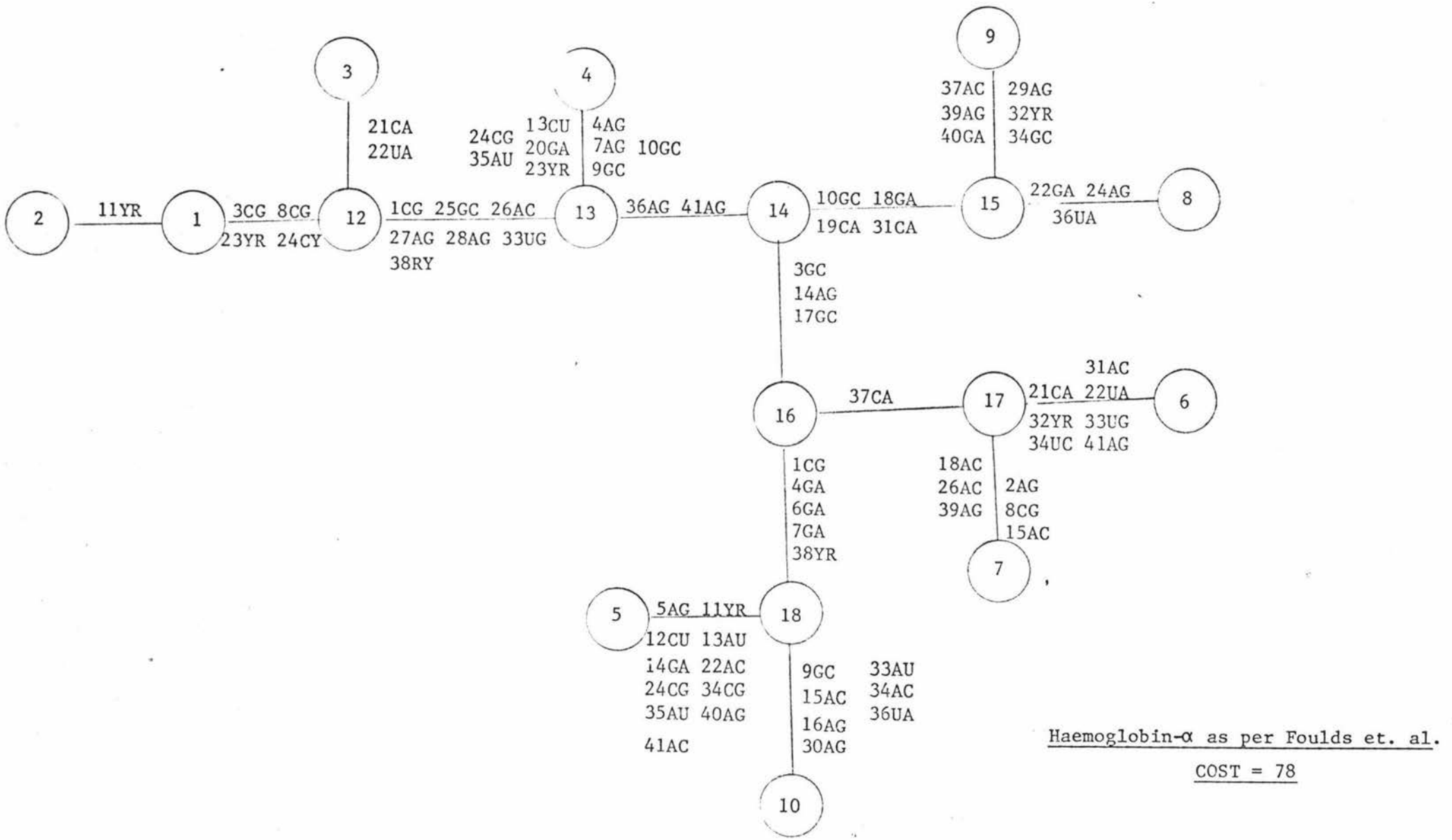
Haemoglobin-α as per PST2

COST = 91



Haemoglobin-α as per PST3

COST = 80

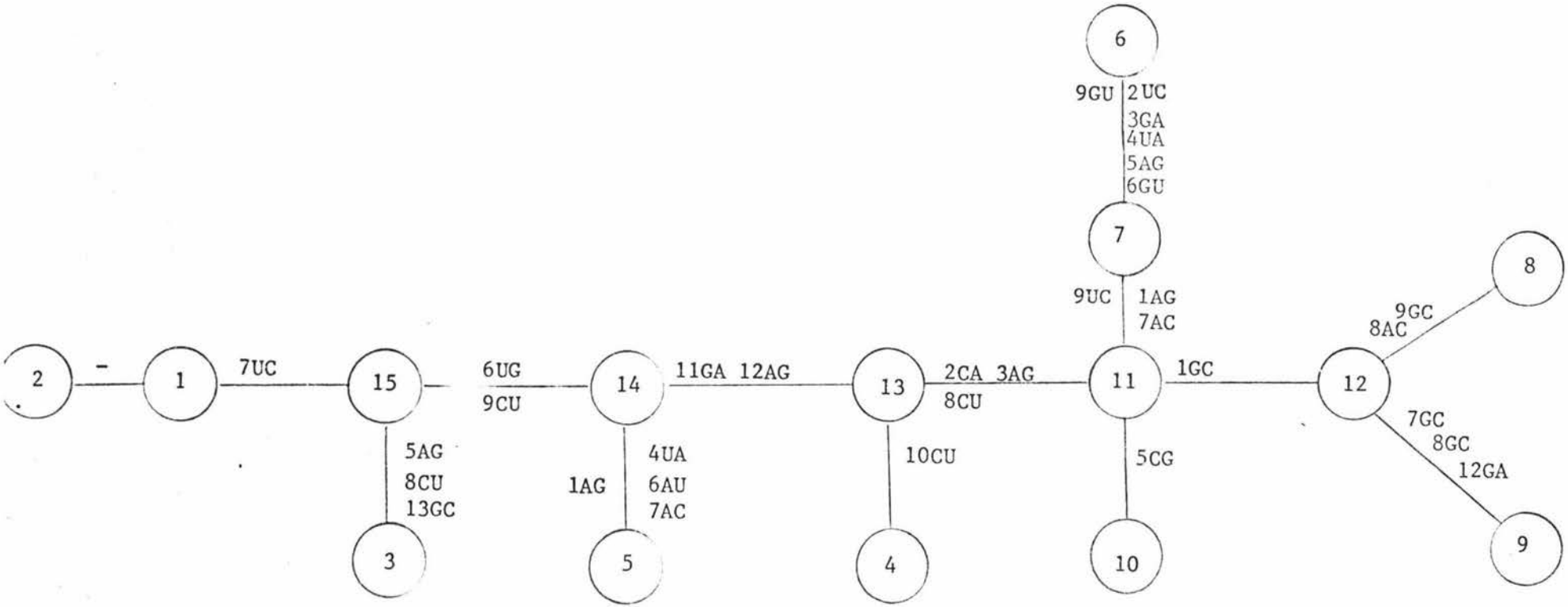


Haemoglobin-α as per Foulds et. al.

COST = 78

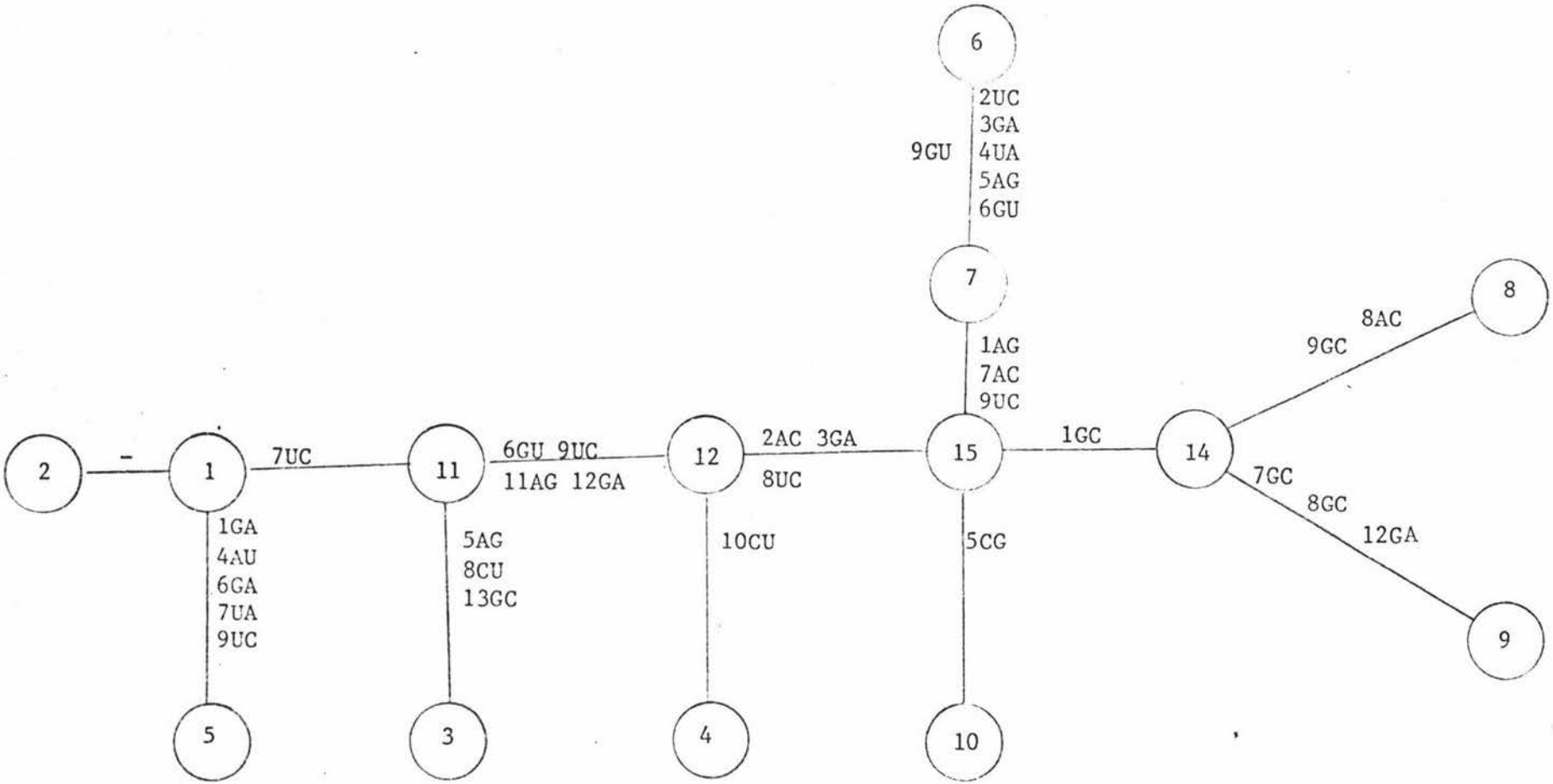
Fibrinopeptide-β nucleotide codings

	sites 1 1234567890123
MAN	GAGAGGUUUUAGC
APE	GAGAGGUUUUAGC
MKY	GAGAAGCCUUAGG
RAB	GAGAGUCUCCCAC
MUS	AAGUGAAUCUAGC
DOG	AUGUAGACGUGAC
HOS	ACAAGUACUUGAC
PIG	CCAAGUCAGUGAC
COW	CCAAGUGGCUGGC
EWE	GCAACUCCCUGAC



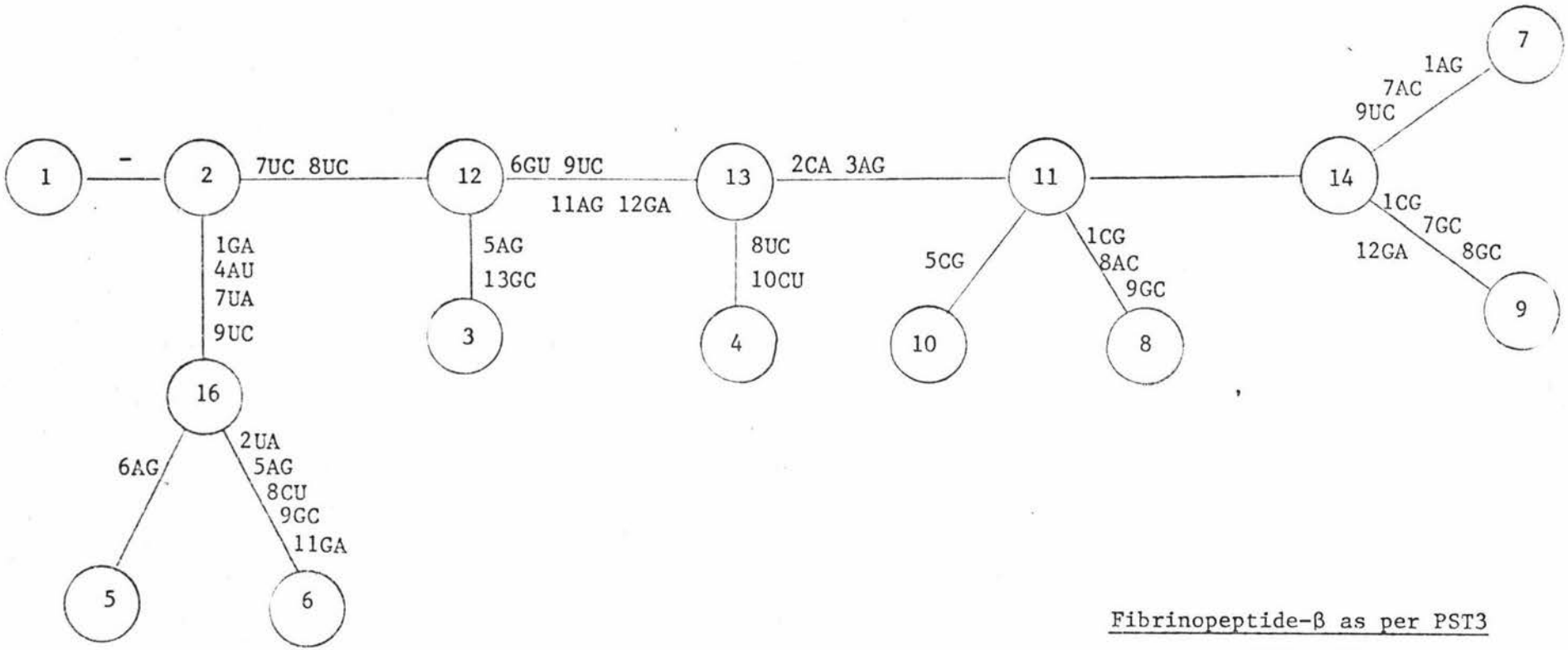
Fibrinopeptide-β as per PST1

COST = 32



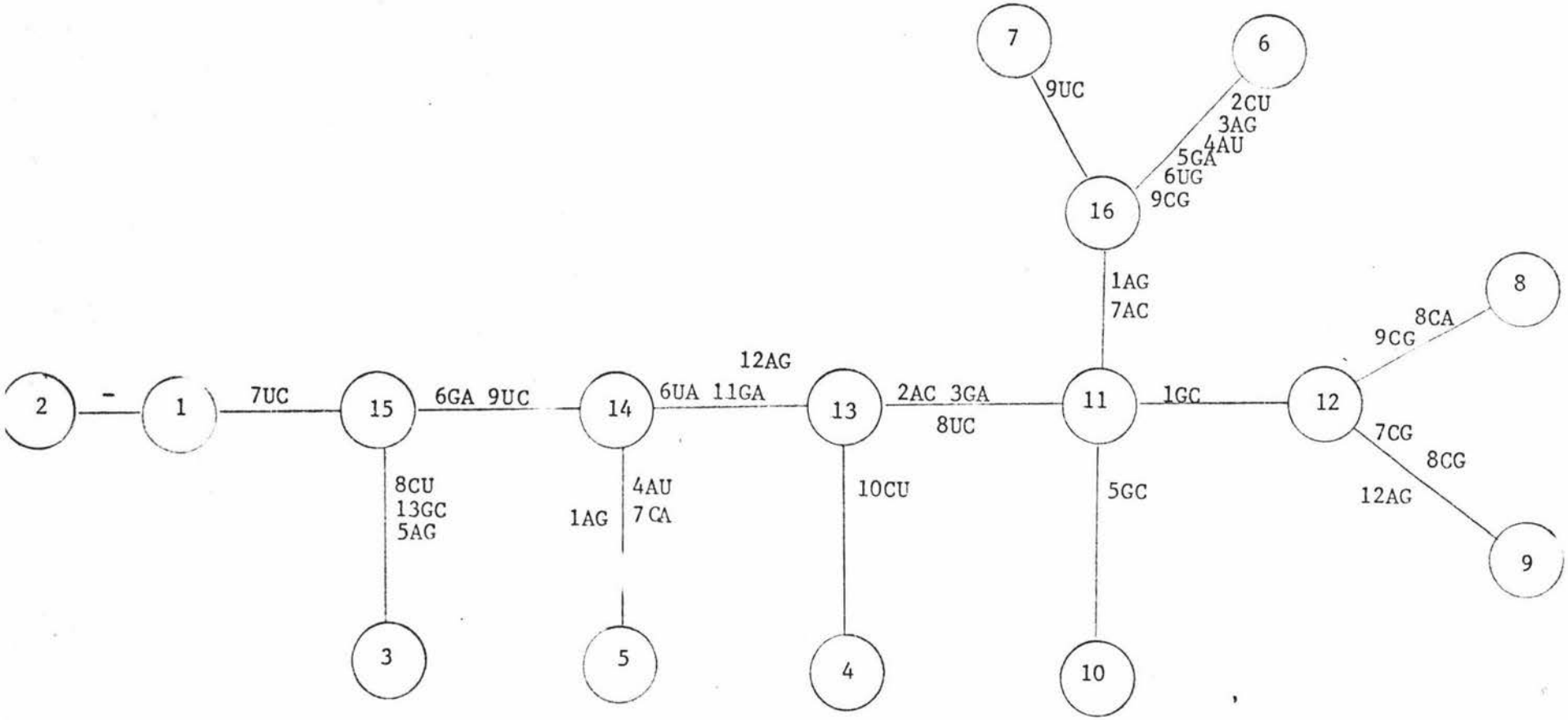
Fibrinopeptide-β as per PST2

COST = 33



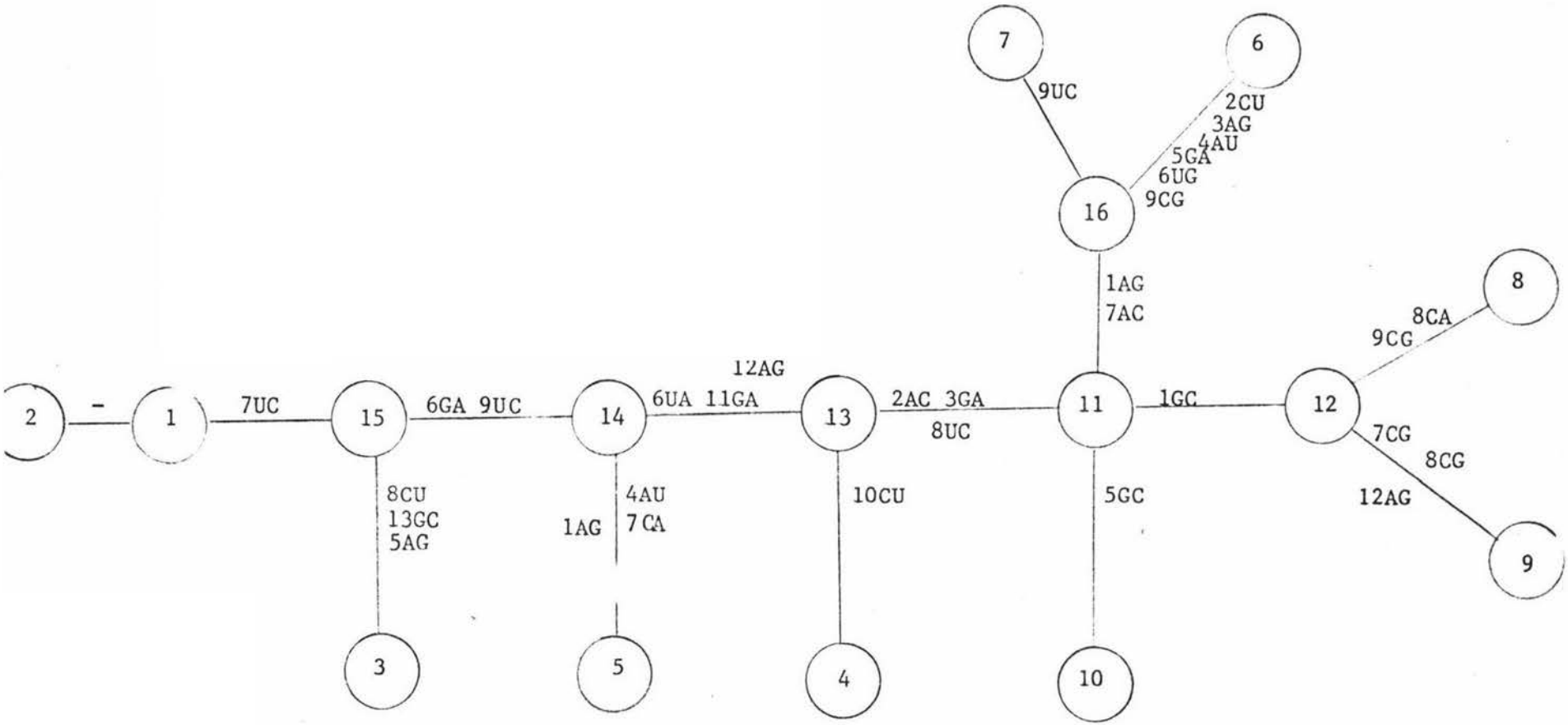
Fibrinopeptide-β as per PST3

COST = 34



Fibrinopeptide-β as per Foulds et. al.

COST = 32



Fibrinopeptide-β as per Foulds et. al.

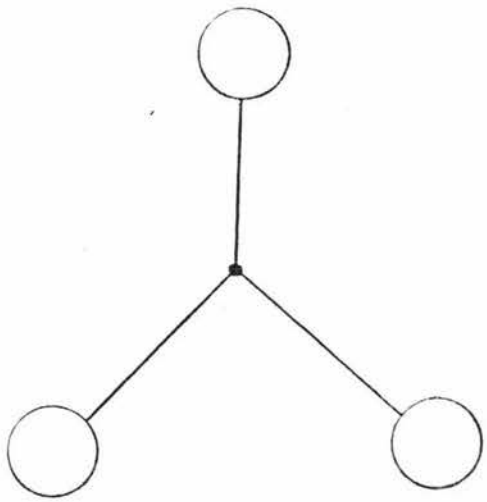
COST = 32

APPENDIX III MPST TOPOLOGIES

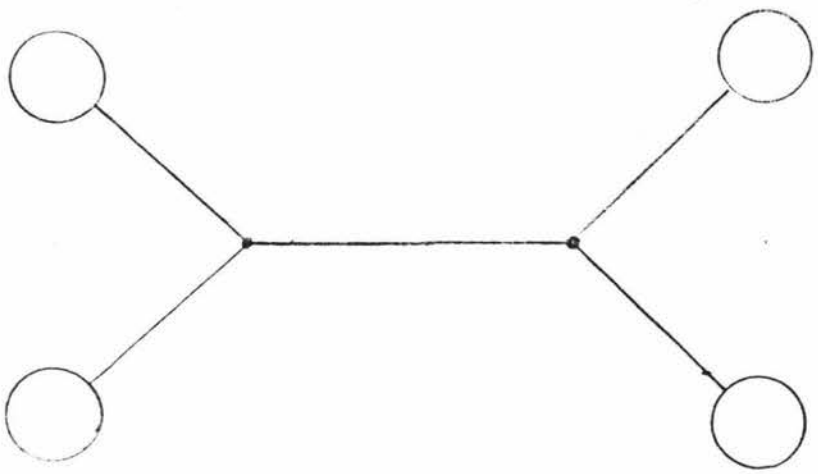
This appendix catalogues the possible MPST topologies for a given number of original species. The circles represent original species, the dots Steiner species.

MPST topology .... 1 species

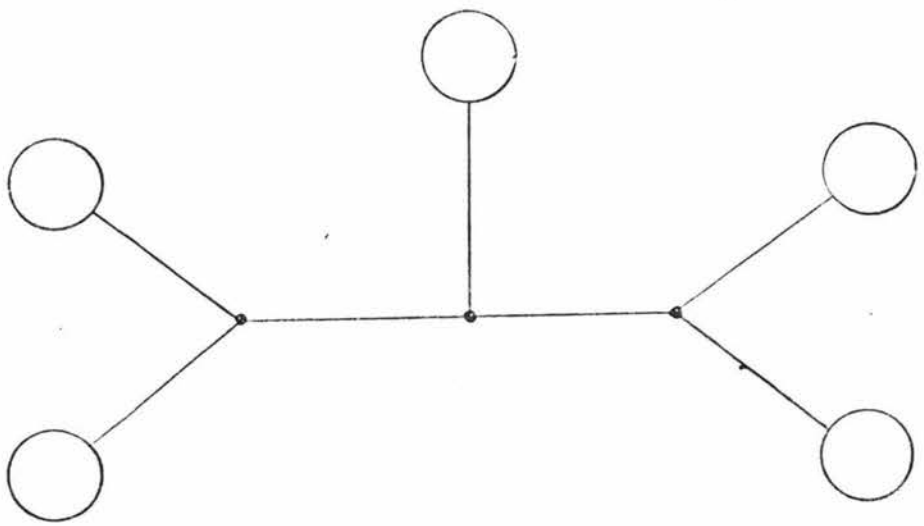
MPST topology .... 2 species



MPST topology .... 3 species

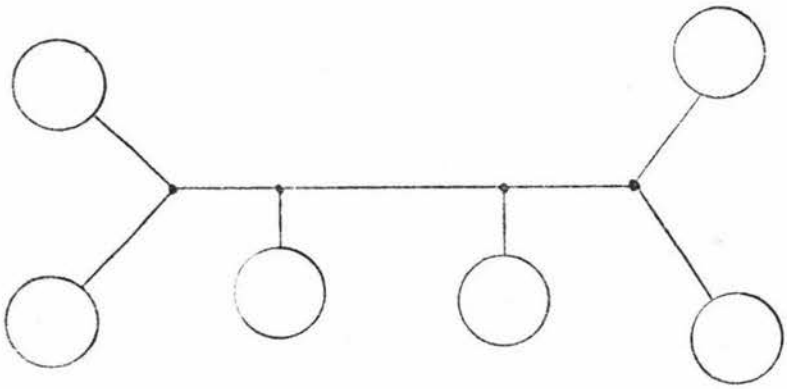


MPST topology .... 4 species

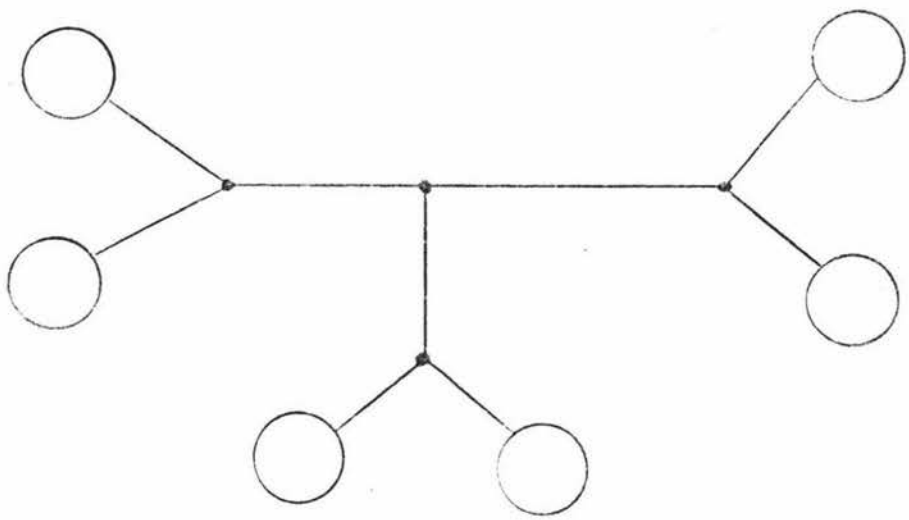


MPST topology .... 5 species

(a)

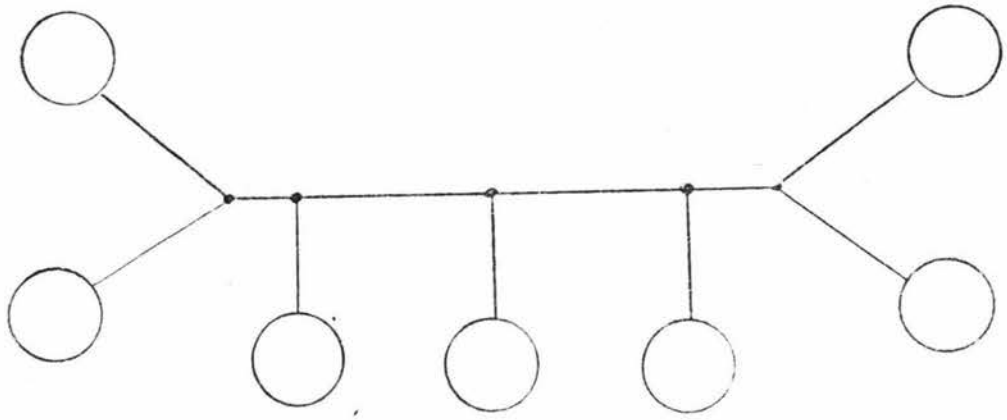


(b)

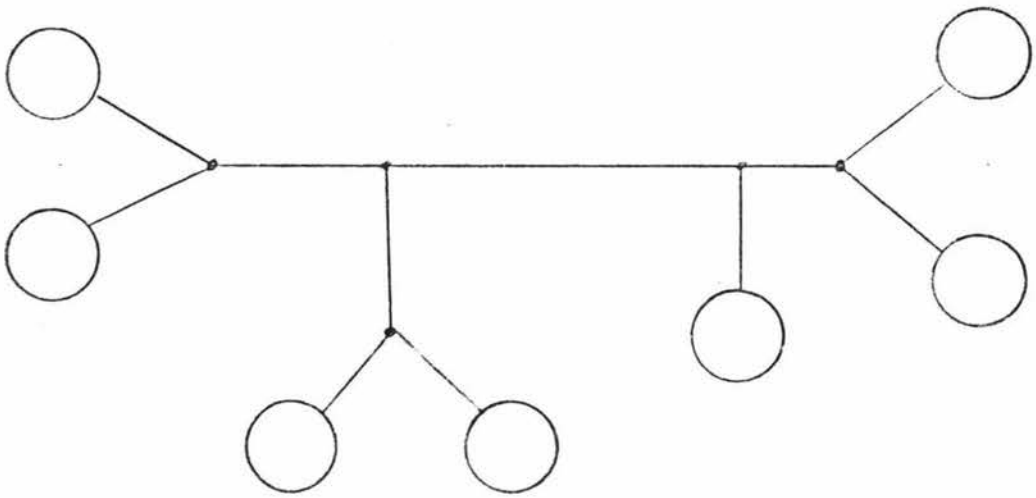


MPST topology .... 6 species

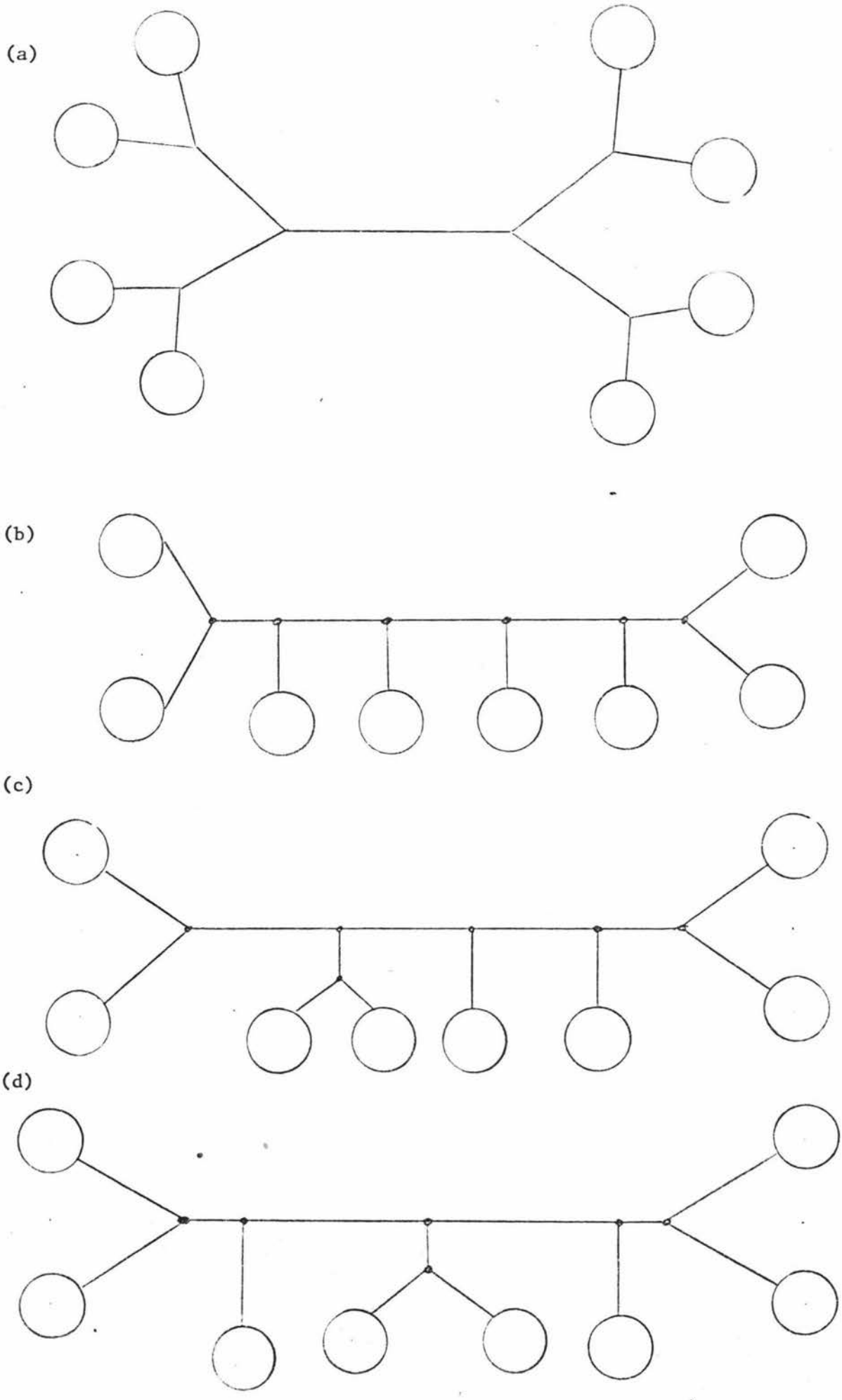
(a)



(b)



MPST topology .... 7 species



MPST topology .... 8 species