A SUCCESSFUL ALGORITHM FOR THE UNDIRECTED HAMILTONIAN PATH PROBLEM

by

G.L. Thompson and S. Singhal

DRC-70-27-84

December, 1984

(W. P. No. 18-83-84)

(Management Sciences Research Report No. 497)

A SUCCESSFUL ALGORITHM FOR THE UNDIRECTED HAMILTONIAN PATH PROBLEM

9 August 1984

by

Gerald L. Thompson

and

Sharad Singhal

This report was prepared as part of the activities of the Management Science Research Group, Carnegie-Mellon University, under Contract No. N00014-82-K-0329 NR 047-048 with the U.S. Office of Naval Research. Reproduction in whole or part is permitted for any purpose of the U.S. Government.

Management Science Research Group
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pa 15213

Abstract

In this **piper** a polynomial algorithm called the Minram algorithm is presented which finds a Hamiltonian Path in in undirected graph with high frequency of success for graphs up to 1000 nodes. It first reintroduces the concepts described in [13] and then explains the algorithm. Computational comparison with the algorithm by Posa [10] is given.

It is shown that a Hamiltonian Path is a spanning arborescence with zero ramification index. Given an undirected graph, the Minram algorithm starts by finding a spanning tree which defines a unique spanning arborescence. By suitable pivots it locates a locally minimal value of the ramification index. If this local minima corresponds to zero ramification index then the algorithm is considered to have ended successfully, else a failure is reported.

Computational performance of the algorithm on randomly generated Hamiltonian graphs is given. The random graphs used as test problems were generated using the procedure explained in Section 6.1. Comparison with our version of the Posa algorithm which we call Posa-Ran algorithm [10] is also made.

Keywords : Hamiltonian Paths, Hamiltonian Cycles, Ramification Index, Heuristic, Probabilistic Algorithms.

1. Introduction

The Hamiltonian Cycle problem is the problem of finding a path in a graph which passes through each node exactly once. This problem is well known and has been discussed in most graph theory books such as [2, 4, 5].

In [13] an algorithm was presented which found a Hamiltonian Path in a general directed graph with a high enough frequency of success so as to be of practical value; such an algorithm was called a *successful* algorithm. A general undirected graph G can be converted to a directed graph by replacing each edge of G by two directed arcs. In principle then, the same algorithm can be used for finding a Hamiltonian Path in an undirected graph also. However by exploiting the special properties of undirected graphs, it became possible to specialize the algorithm given in [13] for undirected graphs. It was empirically found that the specialized version, presented in this paper, worked much faster on undirected graphs.

If a graph has one or more Hamiltonian Paths, our algorithm will either find one of them or end with a message that it cannot proceed further. However, if there is no Hamiltonian Path in the graph, the algorithm will always end with the "cannot proceed further" message.

Despite the fact that the algorithm can fail we have found it to be of great practical value because it is fast and can be run again if a solution is not found. Each time it fails to produce an answer, the probability that the graph *has* a Hamiltonian Path decreases.

This algorithm belongs to a relatively untested class of algorithms for NP-hard problems. Instead of providing a near optimal solution all of the time, it provides the optimal solution most of the time. A survey of this class of algorithms is given in [7]. Posa wrote a theoretical paper [10] on a simple probabilistic algorithm which converges *atmost surety* for a graph having n nodes and c*n*log(n) arcs for c *t* 1 Improvements on the theoretical results in Pdsa's paper are given in [8]. Posa's algorithm was tested by R. Mcgregor [7] for problems having up to 500 nodes. We use this algorithm as a benchmark and present computational comparison of our algorithm with that of Posa's and present computations with both methods for graphs up to 1000 nodes.

Several algebraic functions associated with graphs, which can be used to characterize Hamiltonian Circuits are given in [31. However none of these functions is equivalent to the ramification index presented in this paper. Three algorithms for the Hamiltonian Path problem and their running time estimates are given in [1].

2. Exact statement of the problem

Let G = (V,A) **be a graph** with V as the set of nodes and A as the set of arcs. G has *n* nodes **and** they **are** numbered sequentially from 1 to n. Node *1* and node *n* each have degree one. The problem that is tackled in this paper is to find a Hamiltonian Path (HP) starting at node / and ending at node *n* which goes through all the intermediate nodes exactly once.

2.1. The Hamiltonian Cycle Problem

The problem of finding a Hamiltonian cycle in an undirected graph can be easily converted to a problem of finding a HP as shown in Figure 11. Given any graph G on *m* nodes we make the following transformations :

- Single out any node in the graph and call it s.
- Create a node labeled m+ / and connect it to all neighbors of s.
- Create node *0* and connect it to node *s.*
- Create node *m+2* and connect it to node m+/.
- Relabel the nodes such that node *0* is node /, node *s* is node 2, nodes *m+1* and *m+2* have their labels increased by one, and all other nodes are labeled in any order using numbers from 3 to m+1.
- Let n = m+3.

Note that the transformed graph has three nodes more than the original graph. Whenever we talk about the number of nodes in a graph we mean the number of nodes *after* the transformations have been performed.

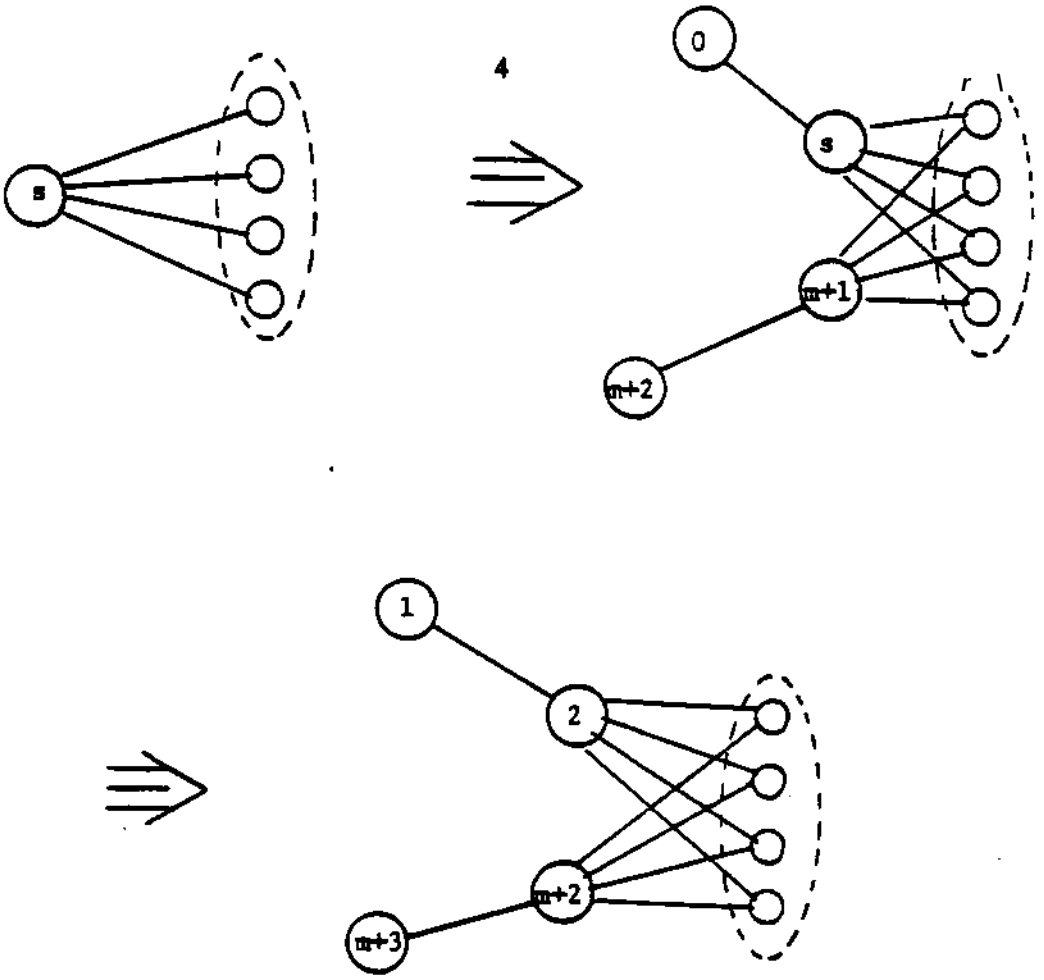


Figure 2.1: Converting a Hamiltonian Cycle Problem to a Hamiltonian Path Problem

3. Review of some Concepts and Deflnitions

Unless otherwise specified, by a graph we mean a connected undirected graph. These definitions are the same as those given in [13], modified for the undirected case wherever appropriate. The proofs for all lemmas and theorems in this section are given in [13].

Definition 1: A connected graph which has no cycles is called a *tree.*

Definition 2: A *rooted tree* is a tree in which an arbitrary (but fixed) node is given the name *root node.* If each edge of a rooted tree is replaced by a directed arc pointing towards the root node, it is called an *arborescence.*

Note that there is a unique arborescence corresponding to any rooted tree.

Definition 3: For any graph G = (V,A), the rooted tree T = $(V,A_T>$ is called a *spanning rooted tree* of G if A_T is a subset of A. When each edge of the spanning rooted tree is replaced by an arc directed towards the root node, we get a *spanning arborescence* of G.

Definition 4: For a given spanning arborescence of G = (V, A), if I j € V and (i,j) 6 A_T, then the *predecessor node of i* is said to be j or symbolically p(i) = j.

One can interpret predecessor of i to be like the "father" of i in the sense of a family tree.

Definition *Sr.* Nodes i and j are called the *end nodes* of arc (ùj).

Definition <K A node i € V is said to be a *junction node* of a rooted tree T = (V.A_T) if it is the end node of at least three arcs belonging to A_T.

Definition 7: A node i € V is said to be a *beginning node* if it is the end node of exactly one edge € A_y.

Definition 8: Let i and j be two nodes of an arborescence T = (V, A_T). If there is no directed path in T from node i to j or from node j to ù then the arcs (i.j) and (j.i) are called *cross arcs* with respect to T. If there is a directed path from i to j and (uj) p A_T then (Lj) is called an *up arc.*

Definition 9: For each i 6 V we define the *successor set* V_i = (h : p(h) = i>. Thus Y_i is the set of nodes which are immediately below i, i.e, the set of nodes for which i is the predecessor node.

Lemma 10: The successor set of a beginning node is the null set

Lemma 11: The successor set of a junction node has at least two elements.

Definition 12: The *successor function* of i represented by s(i), for i € V is defined inductively as follows :

$$s(i) = 1 + \sum_{h \in *_i} s(h)$$

In other words, s(i) = 1 • number of nodes in T below L

Lemma 13: If i is a beginning node then s(i) = 1.

Definition 14: The *ramification index* R of an arborescence T = $(V,A_T>$ is defined as follows:

$$R = R(D = n\cdot(n-1)/2 - \sum_{i \in V-\{n\}} s(i)$$

where n is the root node of the arborescence.

Lemma 15: The ~~nwrinwm~~ possible ramification index of an arborescence is (n-lMn-2)/l

Definition 16: Let J be the set of junction nodes of arborescence T. The *ramification index of a junction node* j € J is defined as :

$$R_J = \sum_{e.\ f \in *_i.\ e < f} s(e) * s(f)$$

The significance of Definition 16 is due to the following theorem.

Theorem 17: The ramification index of an arborescence T can be computed from the following equation:

$$R = R(T) = \sum_{j < J} l \quad R_J$$

Theorem 1& Let T = (V,A_T) be a spanning arborescence of a directed graph

G = (VJO. The following statements are equivalent :

1. T is a Hamiltonian Path starting from node 1 and ending at node n.

1 There is a directed path in T from node 1 to any other node x € V.

3. T has node 1 as its only beginning node.

4. T has no junction nodes.

5. T has zero ramification index.

4. New Results on Undirected Graphs

Given a spanning arborescence, it is possible to compute its ramification index either by using Definition 14 or else by using the equations of Theorem 17. However we now present a method of calculating the *change* in ramification index when a directed arc in the spanning arborescence is dropped and a directed arc not in the spanning arborescence is added. This method is much faster than computing two individual ramification indices.

Definition 19: Given an arborescence $T = (V,A_T)$ and an arc $(ij) \wedge A_{T>}$ a unique undirected cycle is formed when (ij) is added to A_T. The node on this cycle having the largest successor function is called the *Maximal Cycle Node* (in the cycle) *created by (f,j).*

Let the menial cycle node created by an arc $(i\ j) \wedge A_T$ be denoted by k. Note that

- If (ij) is a cross arc then k is the first junction node where the path from i to n intersects the path from j to n.

- If (ij) is an up arc then k = j.

Let there be e arcs on the path from i to k and f arcs on the path from j to k. The outgoing arc can be any arc in the cycle formed by (ij). Denote by s(g) [s(h)] the successor

function of the g*th (h'th) arc on the path from i (j) to fc so that the arc nearest to i (j) corresponds to g = 1 (h = 1). Since the successor functions of the nodes not in the cycle are not affected, we will concentrate on determining the change in the sum of successor functions of the nodes in the cycle.

The sum of successor functions before (ij) is brought in is

$$S = \sum_{i \in \text{cycle}} s(i). \qquad (1)$$

The sum of successor functions after (ij) is brought in and g*th arc is taken out (for some g between 1 and e)

$$T(g) = \sum_{i=g+1}^{e} s(i) - (e - g)\cdot s(g) \cdot (g - 1)\cdot s(g) - \sum_{i=1}^{g-1} s(i) * f\cdot s(g) + \sum_{i=1}^{f} s(i)$$

or

$$T(g) = \sum_{i \in \text{cycle}} s(i) * (f - e * 2g - 1) \cdot s(g) - 2 \cdot \sum_{i=1}^{g-1} s(0 \qquad (2)$$

The difference S - T(g) between equations (1) and (2), and the fact that an increase in the sum of the successor functions amounts to an equal decrease in the ramification index, gives us the following theorem.

Theorem 20: For a given incoming arc, the change in ramification index of a rooted tree depends on the outgoing arc If the g'th arc between i and the maximal cycle node created by the incoming arc (i,j) is taken out then this change in ramification index is given by :

$$A\ R(g) = 2 * \sum_{i=1}^{g-1} s\langle i) - \langle f - e \cdot 2g - 1) \cdot s(g)$$

where

k — The maximal cycle node created by (i,j).

e = Number of arcs from node i to node k.

f = Number of arcs from node j to node k.

g = The number of arcs on the path from node i to node v.

s(t) = The successor function of the t'th node on the path from node i to node k. t=1 corresponds to node i.

Theorem 21: If an arc (i,j) is to be brought into the solution, then the outgoing arc which gives the maximum decrease in the ramification index has the maximal cycle node as one of its end nodes.

Proof: Denoting by $\Delta S(g)$ the change in the sum of successor functions when the g'th arc is taken out, we have from Theorem 20 :

$$\Delta S(g) = (f - e + 2g - 1)\, s(g) - 2 * \sum_{t=1}^{g-1} s(t)$$

or

$$\Delta S(g) = (f - e + 2g + 1) * s(g) - 2 \sum_{t=1}^{g} s(t) \qquad (3)$$

Also

$$\Delta S(g+1) = (f - e + 2g + 1)\, s(g+1) - 2 \sum_{t=1}^{g} s(t) \qquad (4)$$

The difference between equations (4) and (3) gives

$$\Delta S(g+1) - \Delta S(g) = (f - e + 2g + 1) * (s(g+1) - s(g)) \qquad (5)$$

Since $s(g+1) - s(g) \geq 1$, the left hand side of equation (5) would be positive if

$$f - e + 2g + 1 \geq 0$$

which implies

$$g \geq \frac{e - f + 1}{2} \tag{6}$$

If Inequality (6) is satisfied for all g the proposition is proved. If not, the maximum decrease in ramification index may be obtained for g = 1 or for g = e. From Equation (2) we have

$$T(1) = \sum_{t \in \text{cycle}} s(t) + (f - e + 1)\, s(1) \tag{7}$$

$$T(e) = \sum_{t \in \text{cycle}} s(t) * (f * e - 1)\, s(e) - 2 \sum_{t=1}^{e-1} s(t)$$

or

$$T(e) = 2 \sum_{t \in \text{cycle}} s(t) * (f - e * 1)\, s(e) * 2 \left[e * s(e) - \sum_{t=1}^{e} *> \right] \tag{8}$$

Noting that the expression in square brackets in Equation (8) is non-negative, it is easy to see from Equations (7) and (8) that

$$T(e) * T(1).$$

This completes the proof.

Theorem 22: For any arborescence T = (V,A$_T$)

$$1 + \sum_{j \in J} I_j \leq BN * JN \tag{9}$$

where

J	is the set of all Junction nodes
I_j	is the indegree of junction node j
BN	is the number of beginning nodes in the arborescence
JN	is the number of junction nodes in the arborescence.

Proof: Let B denote the set of beginning nodes of T. Every node in the set V-B

that is not a junction node has indegree equal to 1 so that we may write

$$\sum_{j \in V-B} (I-1)_j = \sum_{j \in J} (O-1)_j \qquad (10)$$

Also since the indegree of each beginning node is zero we have

$$\sum_{j \in B} (I-1)_j = -BN \qquad (11)$$

Finally note that since there are n-1 edges in an arborescence of n nodes, we may write

$$\sum_{j \in V} (I-1)_j = (n-1) - n = -1 \qquad (12)$$

Qearly

$$\sum_{j \in V} (I-D)_j = \sum_{j \in V-B} (I-D)_j + \sum_{j \in B} (I-D)_j \qquad (13)$$

Substituting for the right hand side of Equation (13) from Equation (12), and the left hand side from Equations (10) and (11) we get

$$-1 = \sum_{j \in J} (I-D)_j - BN \qquad (14)$$

which is just another way of writing Equation 9.

Theorem 22 shows that if each pivot of an algorithm can reduce the number of beginning nodes in the spanning arborescence of an n node graph, then it will find the Hamiltonian Path in at most n pivots. Each pivot of the algorithm almost always reduces the number of beginning nodes in the arborescence and this explains why the time required to find a local minimum is of the order of n. However, theoretically the time required to reach the local minimum is CXn') because each pivot decreases the ramification index by at least one.

Corollary 23: An arborescence with no junction nodes has only one beginning node.

Definition 24: If every junction node of ah arborescence has indegree two, then it is known as a *binary arborescence.*

Corollary 25: For a binary arborescence, the number of junction nodes is one more than the the number of beginning nodes.

For a binary arborescence, reducing the number of beginning nodes also reduces the number of junction nodes.

5. A Successful Algorithm for Finding A Hamiltonian Path (If there is one)

Let <XV,A) be an undirected graph with n nodes as described in Section 1 The algorithm to be described makes use of Theorem 18 which says that the ramification index of a Hamiltonian Path is zero. We first find a spanning arborescence of G denoted by T by constructing a greedy suiting solution. By performing a sequence of pivots we try to reduce the ramification index of T and when (and if) the ramification index of T becomes 0, we have found a Hamiltonian path.

The algorithm can get stuck at a given arborescence T with ROD > 0 if it is unable to find a pivot which will result in a decrease in the ramification index. This may mean : (a) that the graph has no Hamiltonian Path, or (b) that the algorithm made an unfortunate choice of pivots which led it to get stuck. The alternative to stopping in this case is to choose a different beginning spanning arborescence and go through the algorithm again.

For ease of exposition, we present the algorithm in two separate parts.

5.1. Algorithm for Finding an Initial Spanning Arborcscence

We want to find a spanning arborcscence $T = (V,A_T)$ of a directed connected graph $G = (V,A)$. For this purpose we define a set M which we call the set of marked nodes. This algorithm begins with the set of edges $A_T = f$ and ends when set A_T has been completely defined.

- *Step 0 :* Set M = {nh $A_T = f$, and k=0.
- *Step 1 :* Choose any arc (ij) € A such that i *f-* M and j 6 M.
- *Step 2 :* Set M = M U UK $A_?$ = $A_?$ U {(Lj)> and k=k+l.
- *Step 3 :* If k « n - 1 then STOP, else go to Step 1.

The above is the greedy form of the algorithm for finding a spanning arborescence because it chooses as many as possible of those arcs which have one of their end nodes already marked to belong to A_T. A random greedy starting arborescence can be found by letting p be any number satisfying 0 < p < 1 and altering Step 2 as follows :

- *Step 2** : Let x be a random number between 0 and 1; if x < p go to step 1, otherwise set M = M U U) and $A_?$ = $A_?$ U Uij)} and k = k+1.

5.2. Minram Algorithm : Finding A Hamiltonian Path

- Step 0 : *Find Initial Solution.* Find an arborescence of G with *n* as the root node by using Algorithm 1.
- Step 1 : *Calculate the successor function and ramification index of* T. Using Definition 12 calculate s(i) for all i € V. Then from Definition 14 or from Theorem 17 calculate R(T). If R(T) = 0 STOP — T is a Hamiltonian Path. Else go to next step.
- Step 2 : *Find incoming arc.* For each (undirected) arc (i,j) € A - $A_?$ calculate using Theorem 20 the maximum possible decrease in ramification index if that arc is

brought into A_T in the direction from i to j or from j to i. Let AR be the maximum decrease in ramification index, let (e,f) the corresponding incoming arc and let k be the maximal cycle node thus created. Let the new ramification index be $R_{new} = R - AR$.

- Step 3 : *Check possible cases.* If

  $R_{MW} = 0$ STOP - A Hamiltonian Path has been found.

  $R_{new} \pounds R$ STOP - The algorithm failed to find a Hamiltonian path on this triaL If another trial is desired go back to Step 0 and generate a new, different spanning arborescence.

  $R_{new} < R$ Go to next step.

- Step 4 : *Updating.* Let Guk) be the outgoing arc given by Theorem 2L Let $A_y = A_T + \{(e,f)\} - \{(h.k)h\ R = R_{OEW}$. Update the successor functions. Go to Step 1

Some other references [9] also describe minimal spanning tree algorithms based on pivoting but we have found no references which employ pivoting techniques for spuming arboresceaccs.

It is interesting to note that with minor modifications Minram Algorithm can be used to find a spanning arborescence having *m*riyr\a\ ramification index. The modified algorithm is described in [11]. There are applications in which spanning arborescences which have maximal ramification index can be usefuL One such application is the design of computer terminal networks in which signal concentrators are located at junction nodes [6].

6. Computational Experience

Before the computational results are presented, two aspects of Minram algorithm need to be explained. In order to run the algorithm we need a method for generating a random graph and the generation process is explained in Section 6.1. The computational performance of Minram algorithm can be enhanced by making a few minor observations explained in Section 6.2 Section 6.3 gives the worst case analysis and Section 7 describes one version of the Posa

probabilistic algorithm. Section 8 discusses the actual computational results obtained.

6.1. Generation of **Random** Graphs

We need to generate a random graph of n nodes consisting of a specified number of arcs. We also need to be sure that the graph is Hamiltonian. The following simple algorithm was used to generate such a graph.

> Introduce the arcs (i,i+1) for i = 1,....n-1. This ensures that the graph has at least one Hamiltonian Path.
>
> Pick any two distinct random numbers i and j such that 1 ≤ i, j ≤ n. Include arc (ij) in the graph if it doesn't already exist Continue until the graph has the desired number of arcs.
>
> Perform the transformations explained in Section 11.

The number of arcs in the graphs were chosen to be

M * CONST • N • Log (N)

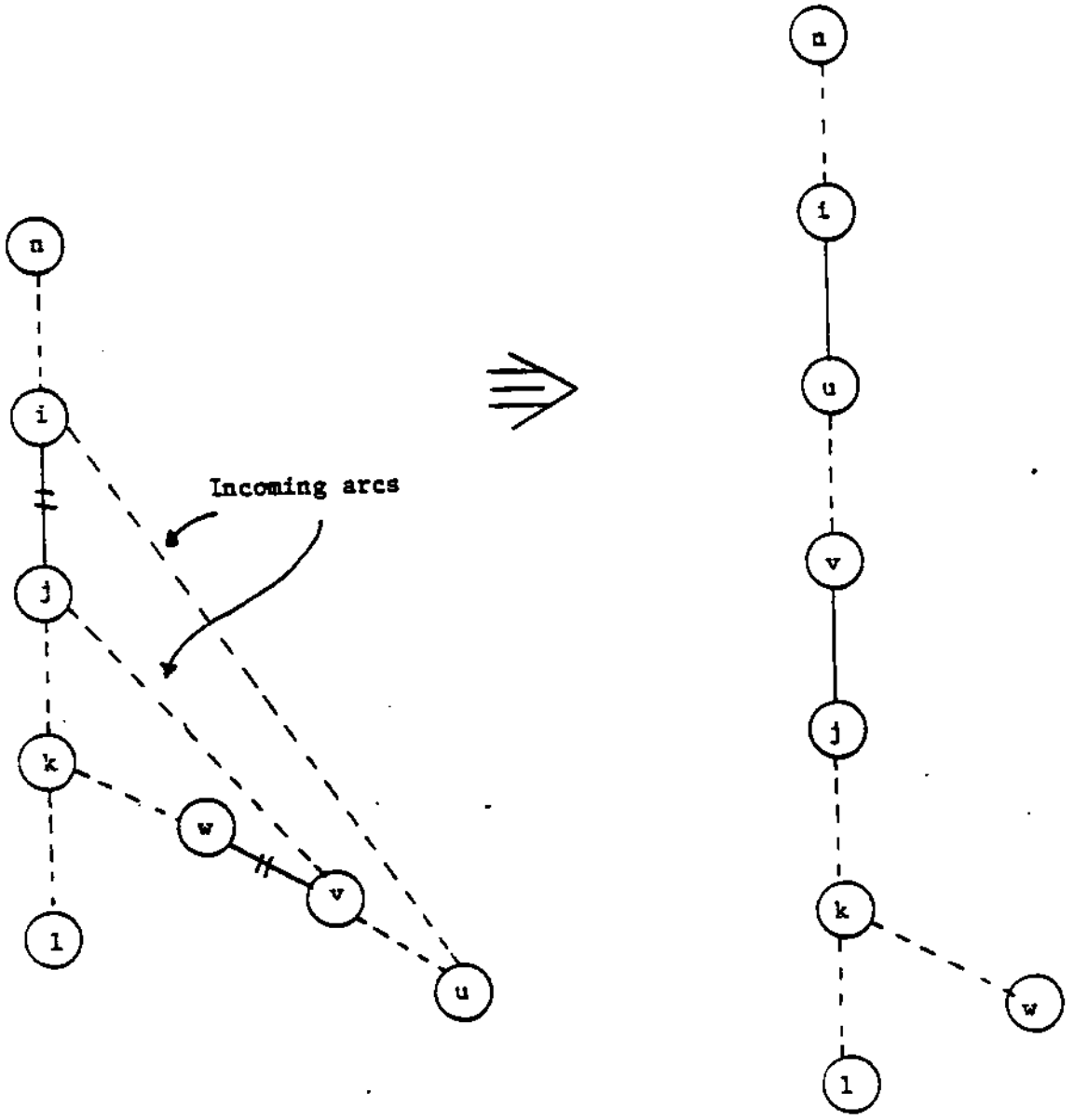
where CONST was varied as a parameter.

6.2. Modifications to Minram algorithm

A modified version of Minram algorithm was run on the DECSYSTEM-20 at Carnegie-Mellon University and the results obtained are shown in Figure 8.1. The following changes were made to Minram algorithm to improve its performance.

- To determine the incoming arc, the whole list of arcs was not searched. The first arc which would decrease the ramification index if brought into the solution was chosen to be the incoming arc.

- Make an Improving Double Pivot. If the situation shown in Figure 6.1 exists, then the ramification index of the arborescence can be reduced in two pivots (but not in one pivot). The code hunted for these double pivots and performed them when



Fifnre 6.1: **Decreasing** the Ramification index in two pivots found.

- The sequence of evaluation of incoming arcs was determined as follows :
  - Look first at all arcs one of whose end nodes is a beginning node.
  - Then look for an improving double pivot of the kind shown in Figure 6.1.
  - Then look at all arcs whose one end node has successor function less than a predetermined number (MAXSUC).

- Again look for an improving double pivot, etc

- If the program is unable to find any improving incoming arc according to the scheme described above, we reversed the algorithm to maximize (instead of minimize) the ramification index for a specified number of pivots. Since this "reshuffles'* the arborescence in some sense, one would hope to get out of the stalemate this way.

- After making the above changes, the code became an infinite code. Thus it was stopped after a predetermined number (MAXPVT) of pivots with the message that the search for the Hamiltonian Path proved unsuccessful

6.3. Complexity of the Minnm algorithm

We now analyze the complexity of the Minram algorithm in the worst case.

Each pivot reduces the ramification index by at least one and therefore we need $O(n^2)$ pivots. Before a suitable incoming arc is found we may have to look at every possible arc implying that the work involved in each pivot is O(nlog(n)). We will have to repeat the process of minimizing ramification index at most k times (where k is a predetermined limit) before we find a Hamiltonian Path or give up the search. Therefore

The Worst Case Complexity of Minram algorithm is Ln^3.login).

7. Posa-ran Algorithm

To evaluate the performance of Minram algorithm we wanted to compare it to another probabilistic algorithm for finding a Hamiltonian Path. Searching the relevant literature revealed that computational experience had only been published for exact methods except for Posa's probabilistic algorithm [7, 10].

Due to the above considerations Posa's Algorithm was chosen as a suitable competitor. It is a simple algorithm to determine a Hamiltonian Path which occasionally fails to yield an answer.

Since the pivoting involved is very simple, closed form estimates of the probability of success have also been derived. We now describe exactly the steps of Posa's algorithm as we programmed them and we call it the Posa-ran algorithm because it involves making some random choices when we are unable to extend a path.

7.1. Description of Posa-ran Algorithm

We wish to find a Hamiltonian Path from node 1 to node n in a graph having n nodes.

We start by tracing a path from node 1 to one of its neighbors. We increase the length of this path as long as we can taking care that we do not visit any node more than once. Also keep in mind that node n must be the last node on this path so it cannot be chosen until the length of the path has become n-2. If we are unable to extend the path any further we perform a type of pivot shown in Figure 7.1 (b) and count this as backtracking once. We continue this process until either the Hamiltonian Path is found or the number of backtracks exceed the specified limit.

Steps of the Posa-ran Algorithm

- *Step 1 :* Set the index of the node to be marked i = 1, length of the path found p = 1, list of the nodes marked MARK(1) = 1, and MARK(2),....,MARK(n) = 0. Set PATH(1) = 1, PATH(2),....,PATH(n) = 0. Set number of backtracks NBACK = 0 and let LIMIT be the permissible number of backtracks.

- *Step 2 :* Choose at random an unmarked neighbor j ($\neq$ n unless p = n-2) of node i. If no unmarked neighbor is found go to Step 4, else go to next step.

- *Step 3 :* Set MARK(j) = 1, p = p+1, PATH(p) = j, i = j as shown in Figure 7.1 (a). If p = n go to Step 5, else go to Step 2.

- *Step 4 :* Choose at random any marked neighbor of i. If PATH(t) = j then let k = PATH(t+1). Update the entries in the vector PATH such that PATH(1)=1,....,PATH(t)=j, PATH(t+1)=i,....,PATH(p)=k as shown in Figure 7.1 (b). Let

i = t Let NBACK = NBACK • 1. If NBACK * LIMIT then go to Step 6 else go to Step 1

- *Step S :* STOP — Hamiltonian Path is found.

- *Step 6 :* STOP — The algorithm failed to decide whether the graph has a Hamiltonian Path.

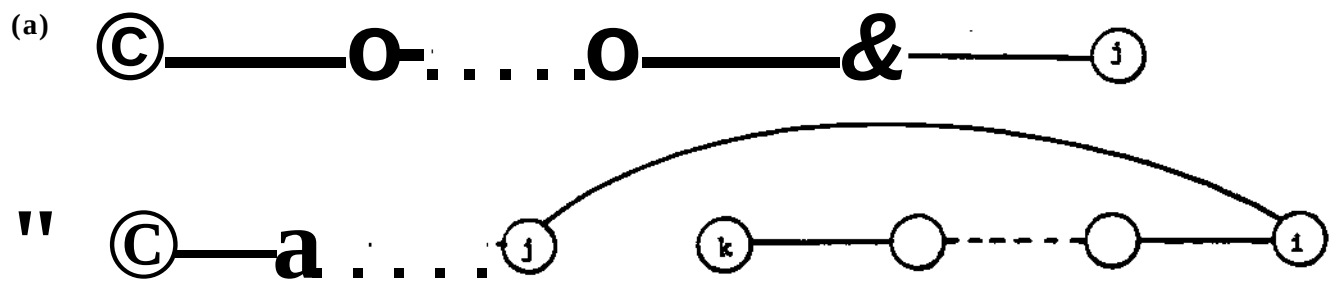


Figure 7.1: The pivoting process in Posa-ran Algorithm

7.2. Compatational Performance of Posa-ran Algorithm

We ran the Posa-ran Algorithm on randomly generated graphs (see Section 6.1) of 100 through 1000 nodes and the mean CPU time obtained is shown in the last column of the table in Figure 8.1. It is clear that it performed very efficiently on these kind of random graphs.

However this algorithm has one major drawback which needs to be pointed out Any node which once becomes a part of a trial path is never allowed to be disassociated from the path. This creates problems for graphs having regular patterns as is demonstrated by two simple examples shown in Figure 7.1

Consider the simple graph shown in Figure 7.2 (a). Posa-ran Algorithm will trace the path 1 to i to j with probability 0.5. Then it will extend the path from j on to m with probability 0.5 and in that case will never be able to find the Hamiltonian Path. Thus the probability of the Posa-ran algorithm failing on this graph is 0.25. This simple pattern can be made to

occur several times in the same graph thereby making the probability that the Posa-ran algorithm will end successfully be arbitrarily smalL

Another example is given in Figure 7.2 (b). In this graph node x is a cut node which divides the graph into two distinct node sets V_1 and V_2. To have any chance of succeeding, Posa's algorithm must correctly trace a Hamiltonian Path among all the nodes in V_j before including any node in \\ on the path. Depending on the structure of the subgraph in V_1, this can be a very unlikely event

In both these examples, the Minram algorithm will have little or no difficulty finding a Hamiltonian Path. We also ran the Posa-ran algorithm on examples of Rectangular Lattice Graphs described in [12] and it never succeeded in finding a Hamiltonian Path. On the other hand, starting with a greedy arborescence. Minram algorithm was always able to find a Hamiltonian Path within a matter of milliseconds for these problems.

Another drawback of the Posa method is that it does not apply to directed graphs. The directed version of the Minram method is given in [13].

8. Interpretation of Computational Results

Minram algorithm modified as explained in Section 6.2 was run on DECSYSTEM-20 at Carnegie-Mellon University. The summary of computational results obtained is shown in Figure 8.1.

We regressed the dependent variable CPU time against the independent variable square root of nodes and obtained the following fit :

$$CPU = .0000154 \cdot n^2 \cdot \text{login)}$$

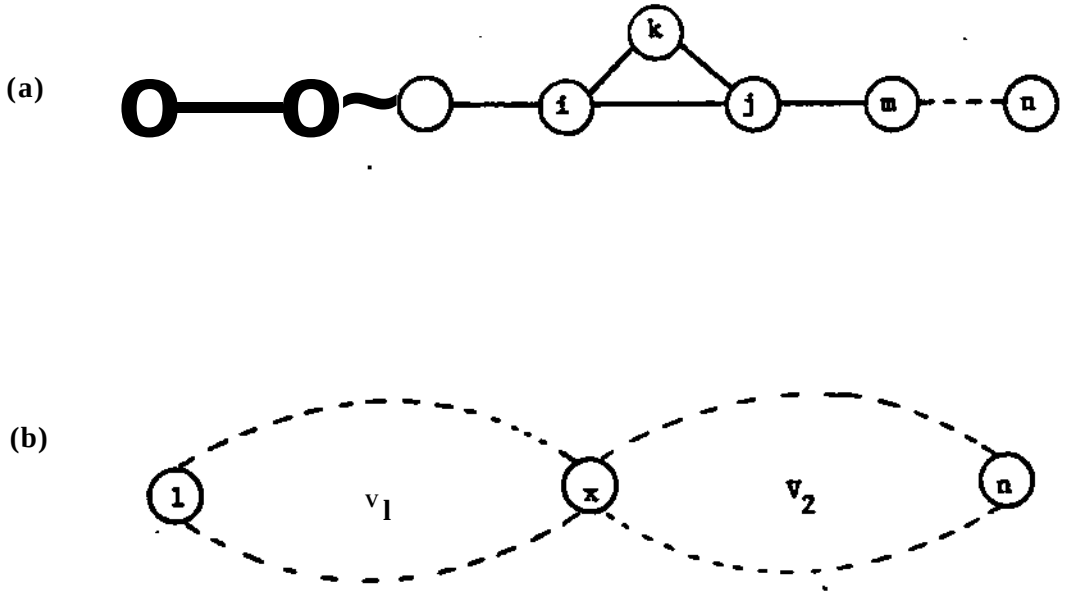


Figure 7.2: Examples of Graphs on which Posa-ran Algorithm may fail

The t-statistics for the co-efficient was 13.66 which is significant with 99% confidence leveL This demonstrates statistically that the CPU time is proportional to the square of the number of nodes times the logarithm of the number of nodes. This is CXn) better than the worst case complexity derived in Section 6.3.

The density of the test graphs generated as described in Section 6.1 is given by

$$\text{Density} = \frac{\text{Number of Arcs in the test graph}}{\text{Number of Arcs in the complete graph}}$$

n = Number of nodes *before* the transformations of Section 2.1

Number of Arcs = 2.0 * n * Log(n)

MAXSUC » 20 and MAXPVT = 4 • n (See Section 6.2)

Both Algorithms were able to solve ALL problems

# of Nodes	# Problems Tried	Average # of Pivots	CPU Time (DEC-20 Seconds) Mean	Min	Max	CPU Sec for Posa Alg.
100	5	100	0.33	0.22	0.47	0.27
200	5	209	1.03	0.65	2.05	0.53
300	5	299	1.95	1.11	4.00	0.69
400	5	529	7.39	2.19	10.85	1.23
500	5	759	18.64	5.54	41.76	1.58
600	5	•824	22.27	5.89	51.05-	2.31
700	7	1012	48.10	14.35	113.49	3.25
800	10	1206	46.67	16.42	114.38	2.84
900	10	1463	78.34	19.88	196.30	3.40
1000	10	1865	131.08	26.61	280.12	4.96

Figure 8.1: Computational Results for Modified Minram algorithm

$$s \frac{2n\log<n)}{0.5n(n-l)}$$

or

$$\text{Density} = \frac{41og(n)}{n-1}$$

This shows that density decreases as n increases which explains why the problems become harder for both methods as n increases.

9. Conclusions

We have described an algorithm for finding a Hamiltonian Path in an undirected graph. This algorithm is an extension of a similar algorithm for directed graphs which was presented in [13]. Several preliminary theoretical results on graphs and spanning arborescences were derived which were then used in the construction of the algorithm.

The algorithm starts with any spanning arborescence and then finds the arborescence with the smallest possible ramification index R. If R s 0 then the Hamiltonian Path was found, otherwise R > 0 which implies that either the graph has no Hamiltonian Path or else the algorithm failed to find one.

Computational experience with graphs having up to n s 1000 nodes was presented which suggests that the frequency of success of the algorithm is close to one, given that the graph had a Hamiltonian Path, for graphs having as few as 1On*login) arcs ; for this reason we call it a successful algorithm. For dense graphs the algorithm has never failed.

Future work by the authors will include improvement of efficiency of the algorithm so that larger problems can be handled. The efficiency can be improved by using new ^chniques for determining the best incoming arc and new ways of handling cases that faiL

References

1. Angluin, D., and Valiant, L.G. "Fast Probabilistic Algorithms for Hamiltonian Circuits and matchings." *Journal of Computer and System Sciences 18* (1979), 155-193.

2. Bondy J. A. and Murty U. S. R.. *Graph Theory with Applications.* North Holland, 1978.

3. Camerini, P.M., Galbiati, G., and Maffioli, F. "On the Complexity of finding Multi-constrained Spanning Trees." *Discrete Applied Mathematics* 5 (1983), 39-50.

4. Christofides, Nicos. *Graph Theory - An Algorithmic Approach.* Academic Press, London, 1975.

5. Garfinkel Robert S. and Nemhauser, George L.. *Integer Programming.* John Wiley and Sons, 1971

6. Gavish, BezaleL Topological Design of Centralized Computer Networks - Formulations and Algorithms." *Networks 12* (1982), 355-377.

7. Karp, Richard M. The Probabilistic Analysis of Some Combinatorial Search Algorithms. In *Algorithms and Complexity : New Directions and Recent Results,* Traub, J. F., Ed.,Academic Press Inc., 1976, pp. 1-19.

8. Komlos, J., and Szemeredi E. Ximit Distribution for the Existence of Hamiltonian Circuits in a Random Graph." *Discrete Mathematics 43* (1983), 55-63.

9. Papadimitriou, C.H., and Steiglitz, K.. *Combinatorial Optimization.* Prentice Hall Inc, Englewood Cliffs, NJ., 1981 Chapter 19

10. Posa, L. "Hamilton Circuits in Random Graphs." *Discrete Mathematics 14* (1976), 359-364.

11. SinghaL S. and Thompson, Gerald L. Maximizing Communications Network Reliability. Working Paper, GSIA, Carnegie-Mellon University, August 1984.

12. Thompson, Gerald L. "Hamiltonian Tours and Paths in Rectangular Lattice Graphs." *Mathematics Magazine 50,* 3 (May 1977), 147-150.

13. Thompson, G.L., and Singhai, S. "A Successful Algorithm for Solving Directed Hamiltonian Path Problems." *Operations Research Letters 3, 1* (Apr 1984), 35-41

Table of Contents