MIT/LCS/TM-208

THE TRAVELING SALESMAN PROBLEM
WITH MANY VISITS TO FEW CITIES

Stavros S. Cosmadakis
Christos H. Papadimitriou

November 1981

THE TRAVELING SALESMAN PROBLEM
WITH MANY VISITS TO FEW CITIES

Stavros S. Cosmadakis* and Christos H. Papadimitriou**

Laboratory for Computer Science
M.I.T., Cambridge, U.S.A.

Abstract

We study the version of the traveling salesman problem in which a relatively small number of cities --say, six-- must be visited a huge number of times --e.g., several hundred times each. (It costs to go from one city to itself.) We develop an algorithm for this problem whose running time is exponential in the number of cities, but logarithmic in the number of visits. Our algorithm is a practical approach to the problem for instances of size in the range indicated above. The implementation and analysis of our algorithm give rise to a number of interesting graph-theoretic and counting problems.

Keywords: Traveling salesman problem, dynamic programming, assignment problem, transportation problem, minimal Eulerian digraph, feasible sequence, Stirling's formula, Stirling numbers of the second kind, min-cost max-flow problem, Edmonds-Karp scaling method.

* Work based in part on the author's Batchelor's Thesis at M.I.T.

** Work supported in part by NSF Grant MCS79-08965.

1. Introduction

In this paper we study the following version of the traveling salesman problem (TSP): We are given n cities, an nxn distance matrix d_{ij} (not necessarily symmetric or with zero diagonal elements), and n integers $k_1,...,k_n > 0$. We are asked to find the shortest closed walk that visits the first city k_1 times, the second city k_2 times, and so on. (We are allowed to visit city i twice in a row, but this costs us d_{ii}.)

This problem, which we call the *many-visits TSP*, is obviously a generalization of the TSP (the TSP is our problem in the special case in which all k_i' s are equal to 1). It can also be considered as a *special case* of the TSP (more precisely, a nonstandard representation of the TSP), in which clusters of k_i cities with identical rows and columns are treated as a single city to be visited k_i times. The many-visits TSP arises in connection to the applications of the TSP in scheduling. In such applications, the cities are in fact *tasks* to be executed, and d_{ij} reflects the *overhead* associated with the task j immediately following task i. Now, in certain applications, each task belongs to one of a few *types*, and tasks of the same type have identical characteristics. For example, in the scheduling of airplane landings, there could only be four types of tasks --e.g., regular, jumbo, private, and military airplanes-- but several dozens of each may be pending at each time for landing. There is a certain delay between the landing of an aircraft and the landing of the next aircraft, depending on the types of the two airplanes. We wish to minimize the total delay. In the many-visits formulation of such a problem n would be 4, while the k_i's would be the number of airplanes of each type.

The many-visits TSP can be solved by extending the dynamic programming approach of [HK]; see [Ps] and Subsection 2.3 of this paper. This algorithm, however, requires time proportional to $n^2\Pi(k_i + 1)$. For $n = 5$, for example, this is already prohibitive when the k_i's are as small as 10. In this paper we present a drastically different approach to the many-visits TSP, which results in an algorithm with running time $O(e(n) \log(\Sigma k_i))$, where $e(n)$ is a moderately growing exponential function of n. For reasonably small values of n --say, up to 10-- our algorithm brings into the realm of realistic solution instances with virtually unlimited k_i's. As evidenced by the running time of our algorithm, which is sublinear in the k_i's, the output is not the optimal walk itself, but a list of the numbers of times that each edge (i,j) participates in the optimal walk. Naturally, since for $k_i = 1$ our problem becomes the ordinary TSP, this exponential dependence on n is expected (and most probably inherent).

We shall now outline our approach. It has been one of the basic and oldest observations in the area, that the TSP can be decomposed into two problems: The *assignment problem* [Ku, La, PS], whose solution guarantees that each city is visited and departed from exactly once, and a *connectivity problem*, which forbids "subtours" in the solution. The first problem is easy, so the hard part of the TSP is enforcing connectivity. Many branch-and-bound algorithms [Ch], algorithms for special cases [GG3], and heuristics [Ka] are based on this decomposition. Our algorithm is based on the following very simple idea: In the many-visits version of the TSP the first problem becomes only a little harder (namely, the *transportation problem* [EK, PS]), whereas the connectivity aspect becomes much easier, in the sense that it is a problem of size n, and therefore can be solved exhaustively if n is small --and this is our working hypothesis.

More specifically, we can restate the many-visits TSP as follows: Given an nxn distance matrix d_{ij} and n integers $k_1,\ldots,k_n$, find the shortest Eulerian directed graph with n nodes and with indegrees $k_1,\ldots,k_n$ in the corresponding nodes. Now an Eulerian digraph must be *strongly connected*, and it must also be *balanced*, that is, it must satisfy at each node i indegree(i) = outdegree(i). An Eulerian digraph that has no other Eulerian digraph as a proper subgraph is called *minimal*. So, a solution of the many-visits TSP can be decomposed into a minimal Eulerian digraph (this is the connectivity part) and a balanced (but possibly disconnected) digraph to bring the degrees up to the required levels of the k_i's (this is the transportation problem). The fortunate fact is that *there is a fixed number of minimal Eulerian graphs on* n *nodes*, independent of the k_i's. Our basic algorithm is now apparent:

1. Repeat the following step for each minimal Eulerian graph G on n nodes:

2. Let $\delta_1,\ldots,\delta_n$ be the sequence of indegrees of G. Solve the transportation problem with distance matrix d_{ij} and both capacities and requirements equal to $k_1-\delta_1,\ldots, k_n-\delta_n$. Superpose the solution to G.

3. Among the Eulerian graphs thus generated, pick the cheapest.

Step 1, generating all minimal Eulerian graphs, can be done in a computationally feasible way only by eploying some interesting graph theory, and using dynamic programming. We discuss this in Section 2. In Section 3 we make some calculations that are necessary for the analysis of the algorithm, solving some counting problems that are interesting in their own right. Finally, in Section 3 we also outline a modification of the algorithm, which replaces the repeated solutions of the transportation problem in step 2 above by the precomputation of the solution to a "master" problem, plus the solution of (much smaller) incremental problems. This modification reduces the computational complexity from $e(n) + e'(n) \log(\Sigma k_i)$ to $e(n) + n^3 \log(\Sigma k_i) + e'(n) \log n$, where $e(n)$ and $e'(n)$ are exponential functions of n, specified in Section 3.

2. Generating Minimal Eulerian Graphs

2.1 A Reduction

It is not at all clear how to implement the first step of our algorithm, i.e., enumerating all minimal Eulerian graphs on n nodes. In fact, the outlook is very bleak, because of the following, rather surprising, result, proven recently in [PY]:

Theorem 1: Testing whether a digraph is minimal Eulerian is coNP-complete.

Fortunately, with a little thought we can circumvent this difficulty. Suppose that two minimal Eulerian digraphs G and G' generated in step 1 have the same *indegree sequence* $(\delta_1,\ldots,\delta_n)$. Then the same transportation problem is solved for both in step 2. Therefore, we need only consider the *cheaper* (under d_{ij}) digraph among G, G'. Hence, for each sequence of integers, which is the indegree sequence of a minimal Eulerian digraph (hereafter called a *feasible* sequence) we may compute the cheapest Eulerian digraph with this indegree sequence. If the resulting digraph is minimal, we proceed to step 2. If it is not, it can be discarded: The final solution corresponding to it will certainly be considered, when we consider the indegree sequence of the minimal Eulerian digraph, which is necessarily a subgraph of the present non-minimal one. Of course, by Theorem 1, we cannot test efficiently each resulting Eulerian digraph for minimality. To improve the efficiency of our algorithm in practice, we could use a reasonably fast heuristic that detects some obvious non-minimal digraphs.

Thus we have reduced step 1 to the following two substeps:

1.1 Generate all *feasible indegree sequences* of length n.

1.2 For each such sequence, find the cheapest Eulerian graph G that has as an indegree sequence the given one.

We examine each of these substeps separately.

2.2 Feasible degree sequences

Surprisingly, although minimal Eulerian graphs are hard to recognize (Theorem 1), their degree sequences have a nice characterization:

Theorem 2: $(\delta_1,\ldots,\delta_n)$ is a feasible degree sequence iff it has at least $\max\ \delta_i$ 1's in it.

We prove the two directions separately.

Lemma 1: Let $G=(V,E)$ be a minimal Eulerian digraph, and suppose indegree$(v_0)=k$ for some $v_0 \in V$. Then there are at least k vertices with degree 1 in G.

Proof: The Lemma is obvious for $k=1$. To prove it in general, consider a set C of cycles whose union is G (by cycle we always mean simple directed cycle, i.e. directed cycle that does not repeat any vertex; any Eulerian digraph can be thought of, although not necessarily in a unique way, as the disjoint union of several cycles). Now construct the following finite sequence $\langle G_i \rangle$ of partial sub-digraphs of G (partial because the vertex set of each of them, with the exception of the last one, is a proper subset of V): each G_i is going to be the union of certain cycles in C. G_0 is the union of the k cycles in C that contain the vertex v_0 (for every $v \in V$, indegree(v) is equal to the number of cycles in C that contain i). Once G_i has been constructed, $i \geq 0$, then G_{i+1} is constructed as follows: If there are elements of C that have not yet been used, at least one of them must contain some vertex in G_i (else G would not be connected); pick such an element of C and add it to G_i to get G_{i+1}. Let $G_0,\ldots,G_m$ be a sequence that can be constructed in this way ($G_m=G$). Let $R(G_i)$, $i=0,\ldots,m$ be the property that G_i contains at least k cycles in C each of which satisfies the following:
(i) It contains a vertex with degree 1 in G_i.
(ii) The remaining cycles in C that make up G_i form a connected partial sub-digraph of G_i.
(G_i may also contain cycles in C that do not satisfy either (i) or (ii); $R(G_i)$ says that at least k of the cycles in C that G_i contains satisfy both (i) and (ii)).

We shall show by induction that $R(G_i)$ is true for all i, $i=0,\ldots,m$. First, we show that $R(G_0)$ is true: G_0 contains exactly k cycles in C, and each of these satisfies (ii) (since the remaining cycles have a common vertex, namely i_0). But also each of these cycles satisfies (i), because if one of them, say C_j, does not, then each vertex in C_j also belongs to some other cycle among the cycles that make up G_0; thus, by removing C_j from G_0 (i.e. by removing the arcs in C_j) we are left with a connected sub-digraph of G_0. Consequently, by removing C_j from G we obtain an Eulerian proper sub-digraph of G, which contradicts our hypothesis that G is minimal Eulerian.

Suppose now that $R(G_i)$, $i \geq 0$, is true, i.e. at least k of the cycles in $\mathbf{C}$ that G_i contains satisfy both (i) and (ii); call these cycles $C_1,...,C_l$, $l \geq k$. We shall show that $R(G_{i+1})$ is true. Call C_{l+1} the cycle in $\mathbf{C}$ that was added to G_i to obtain G_{i+1}; observe that C_{l+1} contains a vertex with degree 1 in G_{i+1}, or else we could remove C_{l+1} from G_{i+1} (and G) and obtain a proper Eulerian sub-digraph of G_{i+1} (and a proper Eulerian sub-digraph of G). Now distinguish three cases:

1. C_{l+1} does not have any vertices in common with any of $C_1,...,C_l$. Then $R(G_{i+1})$ is true, since $C_1,...,C_l$ satisfy both (i) and (ii) in G_{i+1}.
2. C_{l+1} has common vertices with only one of $C_1,...,C_l$, say with C_j. Then C_{l+1} and any of $C_1,...,C_l$ except possibly C_j satisfiy both (i) and (ii) in G_{i+1}, so again $R(G_{i+1})$ is true.
3. C_{l+1} has common vertices with $C_{j_1},...,C_{j_h}$, $1 \leq j_1,...,j_h \leq l$, $h>1$. Then for $p \neq j_r$, $r=1,...,h$, C_p, $1 \leq p \leq l$, satisfies both (i) and (ii) in G_{i+1}. Consider now C_{j_r}, $1 \leq r \leq h$; it clearly satisfies (ii) in G_{i+1}, since it satisfies it in G_i and C_{l+1} has at least one common vertex with C_{j_s}, $s \neq r$. But then C_{j_r} also satisfies (i) in G_{i+1}, since otherwise we could remove it and obtain a proper Eulerian sub-digraph of G. Therefore, $R(G_{i+1})$ is true. Since cases 1-3 exhaust all possibilities, the inductive proof is complete.

It follows that $R(G_m)$, i.e. $R(G)$, is true; but this means that G has at least k vertices with degree 1, and we are done. ▮

<u>Lemma 2</u>: Let $(\delta_1,...,\delta_n)$ be a sequence of integers such that there are at least max δ_i 1's in it. Then it is a feasible degree sequence.

<u>Proof</u>: Given such a sequence $(\delta_1,...,\delta_n)$, we shall construct a minimal Eulerian graph G with degree sequence $(\delta_1,...,\delta_n)$. First, suppose that the number of 1's is exactly equal to the largest δ, say k. Without loss of generality $k=\delta_1 \geq \delta_2 \geq ... \geq \delta_{n-k} > \delta_{n-k+1} = ... = \delta_n = 1$. G is constructed as the union of k cycles. Each of the k cycles contains some of the vertices $1,2,...,n-k$, and a different one among the vertices $n-k+1,...,n$. The δ_{n-k} first cycles are of the form $(1,2,...,n-k,j,1)$, where $j>n-k$. The $\delta_{n-k-1}-\delta_{n-k}$ next (possibly 0) are of the form $(1,2,...,n-k-1,j,1)$. The $\delta_{n-k-2}-\delta_{n-k-1}$ next are of the form $(1,2,...,n-k-2,j,1)$; and so on. Finally, the $\delta_1-\delta_2$ last are of the form $(1,j,1)$, for a total of $(\delta_1-\delta_2)+(\delta_2-\delta_3)+...+(\delta_{n-k-1}-\delta_{n-k})+\delta_{n-k}=\delta_1=k$ cycles, exhausting all k

indegree-1 nodes.

The construction is illustrated in Figure 1 for the sequence (5, 3, 2, 2, 1, 1, 1, 1, 1). It is immediate that (a) each node has the appropriate indegree, and (b) the resulting digraph is minimal Eulerian, since any cycle in it contains an indegree-1 node.

For the case of more than k 1's among the δ_i's, just insert the superfluous indegree-1 nodes in one of the cycles. ∎

Theorem 2 follows immediately from the two Lemmas.

As a consequence of this characterization, feasible degree sequences of length n can be easily enumerated as follows:

1. For $k=2,\ldots,n$ repeat step 2.
2. For each sequence $(\delta_1,\ldots,\delta_{n-k})$ with $k \geq \delta_1 \geq \ldots \geq \delta_{n-k} > 1$ repeat step 3.
3. Generate all distinct permutations of $(\delta_1,\ldots,\delta_{n-k},1,\ldots,1)$.

All enumerations implicit in the steps 2 and 3 are easy to do.

2.3 Optimal Eulerian Graphs

Given a degree sequence $\delta=(\delta_1,\ldots,\delta_n)$, and an nxn distance matrix d_{ij}, we can use dynamic programming [HK, Ps] in order to find the shortest Eulerian graph with this degree sequence. For each degree sequence $a \leq \delta$ (componentwise comparison), and each i, $1 \leq i \leq n$, let $C(a;\ i)$ be the cost of the shortest possible way of starting from city 1, visiting city j a_j times, $j=1,\ldots,n$, and ending up in city i. We then have the recurrence

$$C(a_1,\ldots,a_n;\ i) = \min_j \left[C(a_1,\ldots,a_{i-1},\ a_i-1,\ a_{i+1},\ldots,a_n;\ j) + d_{ji}\right]$$

with the initial conditions $C(1,\ 0,\ldots,0,\ 1,\ 0,\ldots,0;i) = d_{1i}$ (1's in the first and i-th position).

Finally, the cost of the optimal Eulerian graph with degree sequence δ is given by

$$C_{opt} = \min_{j} \left[C(\delta, j) + d_{j1} \right]$$

The straightforward implementation of these recurrences takes time $O(n^2 \, \Pi(\delta_i + 1))$. As usual, we can equally easily recover the optimal Eulerian graph in the same amount of time.

3. Analysis of efficiency

3.1 Preliminaries

Let $F(n)$ be the set of all feasible degree sequences of n nodes. Also, let us define the quantity

$$DP(n) = \sum_{(\delta_1,\ldots,\delta_n)\in F(n)} \prod_i (\delta_i + 1)$$

We can analyze the complexity of our algorithm as follows: The algorithm essentially boils down to solving an optimal Eulerian digraph problem, and an nxn transportation problem with capacities approximately k_i for each degree sequence in $F(n)$. The total effort expended in the dynamic programming algorithm is a small constant times $n^2 DP(n)$. If we use the Edmonds-Karp scaling method for the transportation problem (see [EK] and subsection 3.3), each such problem takes time $O(n^3 \log(\Sigma k_i))$ for a total of $O(|F(n)|\ n^3 \log(\Sigma k_i))$. We must therefore derive asymptotic estimates for $F(n)$ and $DP(n)$. This is the subject of the next subsection.

3.2 Counting Problems

Proposition 1:

(a) $|F(n)| = \sum_{k=2}^{n} C(n,k)\ (k-1)^{n-k}$

(b) $DP(n) = \sum_{k=2}^{n} C(n,k)\ 2^k\ [(k-1)(k+4)/2]^{n-k}$

(Here by $C(n,k)$ we denote the number of ways for choosing k objects among n).

Proof:

(a) Suppose $\delta_i = 1$ exactly for $i = i_m$, where $m = 1,\ldots,k$; each of the other $n-k$ elements can take any value between 2 and k, so there are $(k-1)^{n-k}$ such sequences. For any given k, there are $C(n,k)$ ways to pick $i_1,\ldots,i_k$; also, k can take any value between 2 and n.

(b) Suppose $\delta_i=1$ exactly for $i=i_m$, where $m=1,\dots,k$; $\prod\limits_{i=i_m}^{n}(\delta_i+1)=\prod\limits_{\substack{i=i_m\\ 1\le m\le k}}(\delta_i+1)$ $\prod\limits_{\substack{i\neq i_m \\ \text{for all } m \\ 1\le m\le k}}(\delta_i+1)$. The first factor is equal to 2^k, and in the second factor (δ_i+1) can take any value between 3 and $k+1$, so

$$\sum_{\substack{\delta\in M(n)\\ \delta_i=1 \text{ iff } i=i_m\\ 1\le m\le k}}\ \prod_{i=1}^{n}(\delta_i+1)=2^k[3+\dots+(k+1)]^{n-k}=$$

$= 2^k\ [(k+1)\ (k+2)/2\ -\ 3]^{n-k}= 2^k\ [(k-1)\ (k+4)/2]^{n-k}$, since $(k+1)(k+2)-6=k^2+3k-4=$ $=(k-1)(k+4)$. Again, given k there are $C(n,k)$ ways to pick $i_1,\dots,i_k$, and k can take any value between 2 and n. ∎

Since the counts for $|F(n)|$ and $DP(n)$ are not in closed form, we shall now derive lower and upper bounds for $|F(n)|$ and $DP(n)$, to obtain some more information about their respective rates of growth.

For $n\ge 3$, $2\le\lceil n/2\rceil<n$, and one can get lower bounds for $|F(n)|$ and $DP(n)$ in a trivial way, by taking the term corresponding to $k=\lceil n/2\rceil$ in the respective sum. Specifically,

$$|F(n)|\ >\ C(n,\lceil n/2\rceil)\ (\lceil n/2\rceil-1)^{\lfloor n/2\rfloor},\ \text{and}$$

$$DP(n)\ >\ C(n,\lceil n/2\rceil)\ 2^{\lceil n/2\rceil}\ [(\lceil n/2\rceil-1)\ (\lceil n/2\rceil+4)/2]^{\lfloor n/2\rfloor}\ge$$

$$\ge\ C(n,\lceil n/2\rceil)\ 2^{\lceil n/2\rceil-\lfloor n/2\rfloor}\ (\lceil n/2\rceil-1)^{2\lfloor n/2\rfloor}\ge$$

$$\ge\ C(n,\lceil n/2\rceil)\ (\lceil n/2\rceil-1)^{n-1}.$$

Since $\lceil n/2\rceil\ge n/2$ and $\lfloor n/2\rfloor>n/2-1$, we thus have

$$|F(n)|\ >\ C(n,\lceil n/2\rceil)\ (n/2-1)^{n/2-1},$$

$$DP(n)\ >\ C(n,\lceil n/2\rceil)\ (n/2-1)^{n-1}.$$

Moreover, by Stirling's formula $(n!\approx n^n\ e^{-n}\ (2\pi n)^{1/2})$ we have

$$C(2r,r) = (2r)!/r!r! \approx (2r)^{2r}\ e^{-2r}\ (2\pi 2r)^{1/2}/(r^r\ e^{-r})^2\ 2\pi r = 2^{2r}\ (\pi r)^{-1/2}\ ,\qquad \text{and}$$

$$C(2r+1,r+1) = (2r+1)!/(r+1)!r! = [(2r)!/r!r!]\ [(2r+1)/(r+1)] \approx 2^{2r+1}\ (\pi r)^{-1/2},$$

so $\quad C(n,\lceil n/2\rceil) \approx 2^n\ (\pi\lfloor n/2\rfloor)^{-1/2} \approx 2^n\ (\pi n/2)^{-1/2}$

($a_n \approx b_n$ means $\lim_{n\to\infty} a_n/b_n = 1$); this gives an idea about the rate of growth of these lower bounds.

We can also obtain trivial upper bounds by replacing $(k-1)^{n-k}$ in the summation expression for $|F(n)|$ by $(n-1)^n$, and by replacing $2^k\ [(k-1)\ (k+4)/2]^{n-k}$ in the summation expression for $DP(n)$ by $2^n\ [(n-1)\ (n+4)/2]^n = [(n-1)\ (n+4)]^n$. We thus obtain, using the well-known fact that the sum of the binomial coefficients $C(n,k)$ for $k=0,\ldots,n$ is equal to 2^n,

$$|F(n)| < [2(n-1)]^n \qquad \text{and} \qquad DP(n) < [2(n-1)\ (n+4)]^n$$

Observe that it immediately follows from these straightforward bounds that $\log|F(n)| = \Theta(n\log n)$, and $\log DP(n) = \Theta(n\log n)$.

We shall now derive some more elaborate bounds. First, we derive an upper bound for $|F(n)|$ by estimating the maximum of $(k-1)^{n-k}$ when k ranges from 2 to n.

<u>Theorem 3</u>: For any $\varepsilon>0$, $|F(n)| < [(2+\varepsilon)\ n/\log n]^{(1+\varepsilon)n}$ for large enough n (log denotes the natural logarithm).

<u>Proof</u>: Consider the function $y:(1,\infty)\to\mathbb{R}$ defined by $y(x)=(x-1)^{n-x}$, where $n>2$; $y'(x)=(x-1)^{n-x}g(x)$, where $g(x) = (n-x)/(x-1) - \log(x-1) = -1 + (n-1)/(x-1) - \log(x-1)$. Now $g'(x) = -(n-1)/(x-1)^2 - 1/(x-1) < 0$ $(x>1)$, and $g(2)=n-2>0$, $g(n)=-\log(n-1)<0$, so $g(x)$ has a unique root x_0 in $(2,n)$. Also $g(x)>0$ for $1<x<x_0$, $g(x)<0$ for $x>x_0$, so y has an absolute maximum y_{max} at x_0. Since $g(x_0)=0$, $(n-x_0)/(x_0-1)=\log(x_0-1)$, and $y_{max}=y(x_0)=(x_0-1)^{n-x_0}=(x_0-1)^{(x_0-1)\log(x_0-1)}$. Since $x_0-1>1$, we have that if $x_0-1<\mu$ then $y_{max}<\mu^{\mu\log\mu}$. But now if k is such that $n>ke^{k-1}+1$, then $1+\log[(n-1)/k]>k$, so for $x\geq 1+(n-1)/k$ we have $1+\log(x-1)\geq 1+\log[(n-1)/k]> k\geq$

$\geq(n-1)/(x-1)$, which gives $g(x)<0$, so $x_0<1+(n-1)/k$. Thus, if $n>ke^{k-1}+1$, $x_0-1<(n-1)/k$ and $y_{max}<[(n-1)/k]^{[(n-1)/k]\log[(n-1)/k]}$. Taking $k=\log n-\log\log n$, we have that $n>ke^{k-1}+1$ iff (after some calculations) $n[1-1/e+(\log\log n)/(e\log n)]>1$, which is true since for $n>2$ we have $n>e$ and $\log\log n>0$, and thus $n[1-1/e+(\log\log n)/(e\log n)]>n(1-1/e)>2(1-1/e)>2(1-1/2)=1$. Now for any $\varepsilon>0$ we have for large enough n $n^{\varepsilon/(1+\varepsilon)}>\log n$, which is equivalent to $1/(\log n-\log\log n)<(1+\varepsilon)/\log n$, so $y_{max}<[(1+\varepsilon)\ n/\log n]^{[(1+\varepsilon)\ n/\log n]\log[(1+\varepsilon)\ n/\log n]}$. Since $(1+\varepsilon)/\log n<1$ for large enough n, we obtain

$y_{max}<[(1+\varepsilon)\ n/\log n]^{[(1+\varepsilon)\ n/\log n]\ \log n}=[(1+\varepsilon)\ n/\log n]^{(1+\varepsilon)n}$. But now for each k in $[2,n]$, $(k-1)^{n-k}\leq y_{max}$ by the definition of y, so $|F(n)|=\sum_{k=2}^{n} C(n,k)(k-1)^{n-k}\leq$

$\leq y_{max}\sum_{k=2}^{n} C(n,k)=y_{max}\,(2^n-n-1)<2^{(1+\varepsilon)n}\ y_{max}<[(1+\varepsilon)\ 2n/\log n]^{(1+\varepsilon)n}$. Replacing ε with $\varepsilon/2$, we obtain $|F(n)|<[(2+\varepsilon)\ n/\log n]^{(1+\varepsilon/2)n}$, which gives $|F(n)|<[(2+\varepsilon)\ n/\log n]^{(1+\varepsilon)n}$, for large enough n. ∎

The following analogous result for $DP(n)$ is proved by exactly the same method.

Theorem 4: For any $\varepsilon>0$, $DP(n)<[(2+\varepsilon)\ n/\log n]^{(2+\varepsilon)n}$ for large enough n. ∎

To improve the lower bound on $|F(n)|$, we first find alternative summation expressions for $|F(n)|$ by calculating the exponential generating function of the sequence $|F(n)|$.

Proposition 2:

(a) $$|F(n)|=(-1)^{n-1}+\sum_{2k_1+3k_2+\ldots=n} n!\ /\ k_1!(1!)^{k_1}\ k_2!(2!)^{k_2}\ \ldots$$

where $n\geq 2$ and the k_i's are non-negative integers.

(b) $$|F(n)|=(-1)^{n-1}+\sum_{r=1}^{\lfloor n/2\rfloor} n!/(n-r!)\ S(n-r,\ r)$$

where $n\geq 2$ and $S(n,\ k)$ is the Stirling number of the second kind which is equal to the number of partitions of an n-element set into exactly k classes.

Proof: We first calculate the exponential generating function of the sequence $|F(n)|$:

$$f(x) = \sum_{n=2}^{\infty} |F(n)|\ x^n/n! = \sum_{n=2}^{\infty} \left(\sum_{k=2}^{n} C(n,k)\ (k-1)^{n-k} \right) x^n/n! =$$

$$= \sum_{n=2}^{\infty} \sum_{k=2}^{n} n!/k!(n-k)!\ (k-1)^{n-k}\ x^n/n! =$$

$$= \sum_{k=2}^{\infty} \sum_{n=k}^{\infty} [(k-1)x]^{n-k}/(n-k)!\quad x^k/k! =$$

$$= \sum_{k=2}^{\infty} x^k/k!\ e^{(k-1)x} = e^{-x} \sum_{k=2}^{\infty} (xe^x)^k/k! = e^{-x}\,(e^{xe^x} - xe^x - 1) = e^{x(e^x-1)} - x - e^{-x}.$$

(a) We find an alternative expression for the coefficients in the expansion of $f(x)$:

$$-x - e^{-x} = -1 + \sum_{n=0}^{\infty} (-1)^{n-1}\ x^n/n! = -1 + \sum_{n=2}^{\infty} (-1)^{n-1}\ x^n/n!\ , \text{ and}$$

$$e^{x(e^x-1)} = \sum_{n=0}^{\infty} [x(e^x-1)]^n/n! = \sum_{n=0}^{\infty} 1/n!\ \left(\sum_{i=1}^{\infty} x^{i+1}/i! \right)^n =$$

$$= \sum_{n=0}^{\infty} 1/n! \sum_{k_1+k_2+\ldots=n} n!/k_1!k_2!\ldots \prod_{i=1}^{\infty} x^{(i+1)k_i}/(i!)^{k_i} =$$

$$= \sum_{k_1\geq 0,\ k_2\geq 0,\ldots} x^{2k_1+3k_2+\ldots}/k_1!(1!)^{k_1}\ k_2!(2!)^{k_2}\ldots =$$

$$= \sum_{n=0}^{\infty} \left[\sum_{2k_1+3k_2+\ldots=n} n!/k_1!(1!)^{k_1}\ k_2!(2!)^{k_2}\ldots \right] x^n/n! =$$

$$= 1 + \sum_{n=2}^{\infty} \left[\sum_{2k_1+3k_2+\ldots=n} n!/k_1!(1!)^{k_1}\ k_2!(2!)^{k_2}\ldots \right] x^n/n!$$

Thus, since $f(x) = (-x - e^{-x}) + e^{x(e^x-1)}$, we obtain

$$|F(n)| = (-1)^{n-1} + \sum_{2k_1+3k_2+\ldots=n} n!/k_1!(1!)^{k_1}\ k_2!(2!)^{k_2}\ldots$$

(b) We re-write the summation expression obtained in (a) as follows:

$$|F(n)| = (-1)^{n-1} + \sum_{r=1}^{\lfloor n/2 \rfloor} \sum_{\substack{k_1+2k_2+\ldots=n-r \\ k_1+k_2+\ldots=r}} n!/(n-r)! \ (n-r)!/k_1!(1!)^{k_1} \ k_2!(2!)^{k_2}\ldots$$

But now observe that $(n-r)!/k_1!(1!)^{k_1} \ k_2!(2!)^{k_2}\ldots$ is equal to the number of partitions of an $(n-r)$-element set in which there are exactly k_i classes with i elements, so the inner sum is equal to $n!/(n-r)! \ S(n-r, \ r)$. Therefore,

$$|F(n)| = (-1)^{n-1} + \sum_{r=1}^{\lfloor n/2 \rfloor} n!/(n-r)! \ S(n-r, \ r) \ . \qquad \blacksquare$$

Using (b) of Proposition 2, we can improve our lower bound as follows: We first obtain a simple estimate for $S(n, \ k)$:

<u>Lemma 3</u>: $S(n, \ k) \geq k^{n-k}/k!$

<u>Proof</u>: The number of ways of putting n distinct objects into k distinct boxes is equal to $k!S(n, \ k)$; the number of ways of putting n distinct objects into k distinct boxes such that object i is in box i is equal to k^{n-k}; clearly, $k!S(n, \ k) \geq k^{n-k}$. $\blacksquare$

By considering the term corresponding to $r = \lfloor pn \rfloor$ in the summation expression for $|F(n)|$ given in Proposition 2 (b), and using Lemma 3 and Stirling's approximation, we have

<u>Theorem 5</u>: For all $0<p<1/2$, $|F(n)|$ is bounded from below for large enough n by

$$(c_p \ n^{1-2p})^n \ (2\pi p(1-p) \ n^3)^{-1/2} \ (e^{1- \ 1/p} - \varepsilon)$$

where $c_p = p^{1-3p}(1-p)^{p-1}$, and $\varepsilon>0$.

The first few values of $|F(n)|$ are given in Table 1.

n	$\lvert F(n)\rvert$	*DP(n)*
3	4	44
4	15	456
5	66	5,992
6	335	101,212
7	1,898	1,889,428

Table 1

3.3 Solving the Transportation Problem

In this subsection we briefly outline the Edmonds-Karp scaling method for the min-cost network flow problem, of which the transportation problem is a special case. Recall that we wish to find the cheapest "pseudo-Eulerian" (i.e., with balanced indegrees-outdegrees but perhaps not connected) digraph with the given indegrees $c_i = k_i - \delta_i$. This is equivalent to the min-cost max-flow problem on the following network N ([FF, La, EK, PS]): The nodes of N are $\{s,t\} \cup \{s_i, t_i: i=1,...,n\}$ and the arcs are $\{(s,s_i), (t_i,t): i=1,...,n\} \cup \{(s_i,t_j): i,j=1,...,n\}$. Arcs (s,s_i), (t_i,t) have cost 0 and capacity c_i, whereas arc (s_i,t_j) has cost d_{ij} and capacity ∞.

A flow f from s to t in N is called *extreme* if it is of minimum cost among the flows of equal value. It is called *pseudo-extreme* if there exist real numbers u_i, v_i, $i=1,...,n$ such that (a) $u_i - v_j + d_{ij} \geq 0$ for all i, j and (b) whenever $u_i - v_j + d_{ij} > 0$ we have 0 flow in f from s_i to t_j. If we start with a pseudo-extreme initial flow we can perform flow augmentations that preserve the pseudo-extreme property. The maximum flow we end up with is therefore pseudo-extreme, and it turns out that the maximum pseudo-extreme flow is also extreme, and thus the desired solution (see [EK] for a proof).

Define now the *p-th Approximation* to our problem to be a min-cost max-flow problem on the same nodes, arcs and costs, only with capacities $\{\lfloor c_i/2^p \rfloor\}$. The original problem is thus the 0-th Approximation. If f is a pseudo-extreme flow in the p-th Approximation, then obviously $2f$ is a pseudo-extreme flow in the $(p-1)$-th. The Edmonds-Karp scaling method computes in this way successively maximum pseudo-extreme flows for Approximations l, $l-1$,..., 0, where $l = \lceil \log_2(\max_i c_i) \rceil$. Each Approximation can be solved in $O(n^3)$ time, and the total complexity is $O(n^3 \log(\Sigma c_i))$.

For our problem we must solve $|F(n)|$ such min-cost max-flow problems, all with the same nodes, arcs and costs, and with capacities varying slightly (namely, $c_i = k_i - \delta_i$) for a total complexity $O(|F(n)|\ n^3 \log_2(\Sigma k_i))$. Instead, however, we could solve a single "master" problem with capacities $\{\lfloor (k_i - n)/2^p \rfloor\}$, where p is to be determined. Then we solve each of the $|F(n)|$ problems by starting with the $(p-1)$-th Approximation, and with initial flow $2f$, where f is the optimum flow in the master problem. By taking $p=\lceil \log n \rceil$ we can solve each of the $|F(n)|$ problems in $O(n^3 \log n)$ time (notice that always $k_i - \delta_i \geq k_i - n$). The total computation for the transportation problems is therefore reduced from $O(|F(n)|\ n^3 \log(\Sigma k_i))$ to $O(n^3 \log(\Sigma k_i) + |F(n)|\ n^3 \log n)$.

4. Discussion

A good part of our investigations has been of rather theoretical interest --e.g., the asymptotic improvement sketched in Subsection 3.3. Nevertheless, we think that our algorithm is of practical value, since it can be used to solve instances of size far beyond those previously thought possible. One of the most attractive features of our algorithm in practice is that, if n and the distance matrix are known and fixed in advance, then the best part of the computation (i.e., the generation of $F(n)$ and the computation of the optimal Eulerian graph for each sequence in it) can be done once and for all, and the results stored in a large table. Besides, our algorithm can be adapted to find the optimal solution of a *dynamically evolving* instance (e.g., by performing a few more augmentations in the transportation problem whenever the k_i's are increased), whereas the dynamic programming approach is not very flexible in this direction. Naturally, there is a drawback: Our approach is best suited for minimizing the length of the walk (the *makespan*, or finishing time of the last job, in scheduling terminology), while dynamic programming can be adapted to optimize other objectives as well [Ps]. We also mention in passing that our approach to the many-visits TSP is reminiscent in spirit of the classical "precomputation" approach to the cutting-stock problem [GG1].

A practical implementation of our algorithm would most probably incorporate a less sophisticated code for the transportation problem than the Edmonds-Karp scaling method, and could use a heuristic test for minimality for the digraph G produced in step 1. Of course, the ultimate heuristic would be to first solve the transportation problem with requirements and capacities k_i, and then check whether, by a stroke of luck, the resulting digraph is connected. One might expect that this should happen much more often in this problem than in the ordinary TSP.

References

[Ch] N. Christofides *Graph Theory: An Algorithmic Approach*, Academic Press, 1975.

[EK] J. Edmonds, R.M. Karp "Theoretical Improvements in the Algorithmic Efficiency for Network Flow Problems", *J.ACM, 19,* pp.248-64, 1972.

[FF] L.R. Ford, Jr., D.R. Fulkerson *Flows in Networks,* Princeton Univ. Press, 1962.

[GG1] P.C. Gilmore, R.E. Gomory "A Linear Programming Approach to the Cutting Stock Problem", *J.ORSA, 9,* pp. 849-859, 1961. Part II, *J.ORSA, 11,* pp. 863-888, 1963.

[GG3] P.C. Gilmore, R.E. Gomory "Sequencing a One-State Variable Machine: A Solvable Case of the Traveling Salesman Problem", *J.ORSA, 12,* pp. 655-679, 1964.

[HK] M. Held, R.M. Karp "A Dynamic Programming Approach to Sequencing Problems", *J.SIAM, 10,* pp. 196-210, 1962.

[Ka] R.M. Karp "A Patching Algorithm for the Non-symmetric Traveling Salesman Problem", *SIAM J. Comp., 8,* pp. 461-473, 1979.

[Ku] H.W. Kuhn "the Hungarian Method for the Assignment Problem", *Naval Res. Log. Quart., 3,* pp. 253-258, 1956.

[La] E.L. Lawler *Combinatorial Optimization: Networks and Matroids,* Holt-Rinehart-Winston, 1976.

[PS] C.H. Papadimitriou, K. Steiglitz *Combinatorial Optimization: Algorithms and Complexity,* Prentice-Hall, 1981 (in press).

[PY] C.H. Papadimitriou, M. Yannakakis "On Minimal Eulerian Graphs", *Information Proc. Letters,* to appear, 1981.

[Ps] H.N. Psaraftis "A Dynamic Programming Approach for Sequencing Groups of Identical Jobs", *J.ORSA, 28,* pp. 1347-59, 1980.

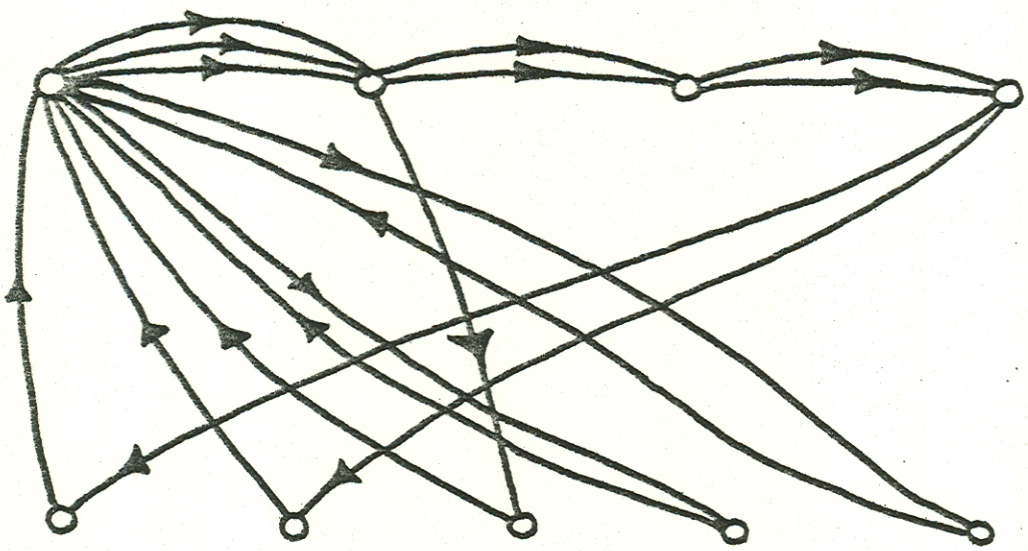

Figure 1