A Dynamic Programming Approach to Sequencing Problems

Michael Held; Richard M. Karp

Journal of the Society for Industrial and Applied Mathematics, Vol. 10, No. 1 (Mar., 1962), 196-210.

Stable URL:
http://links.jstor.org/sici?sici=0368-4245%28196203%2910%3A1%3C196%3AADPATS%3E2.0.CO%3B2-P

Journal of the Society for Industrial and Applied Mathematics is currently published by Society for Industrial and Applied Mathematics.

J. Soc. Indust. Appl. Math.
Vol. 10, No. 1, March, 1962
Printed in U.S.A.

A DYNAMIC PROGRAMMING APPROACH TO SEQUENCING PROBLEMS*

MICHAEL HELD† AND RICHARD M. KARP†

INTRODUCTION

Many interesting and important optimization problems require the determination of a best order of performing a given set of operations. This paper is concerned with the solution of three such *sequencing problems*: a scheduling problem involving arbitrary cost functions, the traveling-salesman problem, and an assembly-line balancing problem. Each of these problems has a structure permitting solution by means of recursion schemes of the type associated with dynamic programming. In essence, these recursion schemes permit the problems to be treated in terms of *combinations*, rather than *permutations*, of the operations to be performed. The dynamic programming formulations are given in §1, together with a discussion of various extensions such as the inclusion of precedence constraints. In each case the proposed method of solution is computationally effective for problems in a certain limited range. Approximate solutions to larger problems may be obtained by solving sequences of small derived problems, each having the same structure as the original one. This procedure of successive approximations is developed in detail in §2 with specific reference to the traveling-salesman problem, and §3 summarizes computational experience with an IBM 7090 program using the procedure.

1. PROBLEM FORMULATIONS

1.1 A scheduling problem. Suppose we are given a set of jobs $J_1, J_2, \cdots, J_n$ which are to be executed successively on a single facility. Any given job J_k is assumed to require the services of the facility for τ_k units of time. With J_k is also associated a function $c_k(t)$, giving the cost associated with completing J_k at time t. We assume that the facility is to be constantly in use, and that no job is to be interrupted before completion.[1] With these assumptions, any given order of execution of the jobs (a *schedule*) may be represented by an ordering $(i_1 i_2 \cdots i_n)$ of the integers from one through n, indicating that the jobs are to be executed in the order $J_{i_1}, J_{i_2}, \cdots, J_{i_n}$. Given such a schedule, the termination time t_{i_k} of J_{i_k} is $\sum_{j=1}^{k} \tau_{i_j}$, and

* Received by the editors August 31, 1961.

† International Business Machines Corporation, New York, N. Y.

[1] There is no advantage in violating these assumptions when the functions $c_k(t)$ are monotone nondecreasing, representing penalties incurred for deferring the completion of the jobs.

the total cost associated with the schedule is

$$\sum_{k=1}^{n} c_{i_k}(t_{i_k}). \tag{1}$$

We seek an ordering for which (1) assumes its minimum value $\mathcal{C}$.

Several authors have considered this problem for special kinds of cost functions. For example, McNaughton [9] considers the case in which each $c_k(t)$ is constant up to some point in time, and then increases linearly. The dynamic programming algorithm which follows requires no assumptions to be made about the cost functions.

Let $S = \{k_1, k_2, \cdots, k_{n(S)}\}$ be a subset of $\{1, 2, \cdots, n\}$, and denote by t_S the quantity $\sum_{j \in S} \tau_j$. Let $C(S)$ be the minimum cost incurred in executing the jobs $J_{k_1}, J_{k_2}, \cdots, J_{k_{n(S)}}$, in any order whatever, in the interval $[0, t_S]$. For $l \in S$, let $S - l$ denote the set obtained by deleting l from S. Then we may define the following recurrence relations, in which $C(S)$, for any set S, is expressed in terms of the values of C for subsets of S:

$$\begin{aligned} &\text{(a)} \quad (n(S) = 1): && C(\{l\}) = c_l(\tau_l), \quad \text{for any } l. \\ &\text{(b)} \quad (n(S) > 1): && C(S) = \min_{l \in S} [C(S - l) + c_l(t_S)]. \end{aligned} \tag{2}$$

To justify this, we argue as follows: In an optimal order of execution for the jobs with indices in S(i.e., one which realizes the cost $C(S)$) some job, call it J_l, must be executed last, and the remaining jobs (those with indices in $S - l$) must be executed optimally in $[0, t_{S-l}]$. Then the total cost incurred by such an ordering will be $C(S - l) + c_l(t_S)$. Taking the minimum over all choices of l, we obtain (2b).

It follows that an ordering $(i_1\, i_2 \cdots i_n)$ is optimum if and only if,

$$C(\{i_1, i_2, \cdots, i_p\}) = C(\{i_1, i_2, \cdots, i_{p-1}\}) + c_{i_p}(t_{\{i_1, i_2, \ldots, i_p\}}) \quad (2 \leqq p \leqq n). \tag{3}$$

The procedure for finding an optimum solution has two phases. In the first phase, the quantities $C(S)$, for all $S \subseteq \{1, 2, \cdots, n\}$, are computed recursively from (2); $\mathcal{C}$ is given by $C(\{1, 2, \cdots, n\})$. In the second phase, (3) is used to generate an optimum ordering $(i_1\, i_2 \cdots i_n)$; i_n is obtained first, and then, successively, $i_{n-1}, i_{n-2}, \cdots, i_1$.

The fundamental operations required are additions and comparisons, which occur in equal numbers. The number of additions in the first phase is given by $\sum_{k=1}^{n} k \binom{n}{k} = n2^{n-1}$; the number of additions in the second phase is at most $\sum_{k=2}^{n} k = [n(n + 1)/2] - 1$. If one storage location is assigned to each number $C(S)$, the number of locations required is $2^n - 1$.

1.2. The traveling-salesman problem. The dynamic programming formulation of the scheduling problem just considered is based on the fact that the cost incurred in executing a job depends only on which jobs preceded it. We now discuss the traveling-salesman problem, which exhibits a different kind of cost dependence.

"The traveling-salesman problem is that of finding a permutation $P = (1\ i_2\ i_3 \cdots i_n)$ of the integers from 1 through n that minimizes the quantity

$$a_{1i_2} + a_{i_2i_3} + a_{i_3i_4} + \cdots + a_{i_n1}, \tag{4}$$

where the $a_{\alpha\beta}$ are a given set of real numbers. More accurately, since there are only $(n-1)!$ possibilities to consider, the problem is to find an efficient method for choosing a minimizing permutation.

"The problem takes its name from the fact that a salesman wishing to travel by shortest total distance from his home to each of $n-1$ specified cities, and then return home, could use such a method if he were given the distance $a_{\alpha\beta}$ between each pair of cities on his tour. Or, if the salesman desired the shortest total travel time, the $a_{\alpha\beta}$ would represent the individual travel times" [6].

While no completely acceptable computational method exists for solving the traveling-salesman problem, several procedures have been developed for obtaining optimum or near optimum solutions. These procedures, however, are usually somewhat tedious, intuitive, and difficult to program for a computer. A "state of the art" discussion of the traveling-salesman problem may be found in [1]. We proceed to give a dynamic programming formulation for this problem.[2]

Given $S \subseteq \{2, 3, \cdots, n\}$ and $l \in S$, let $C(S, l)$ denote the minimum cost of starting from city one and visiting all cities in the set S, terminating at city l. Then

$$\begin{aligned} &\text{(a)}\ (n(S) = 1): \quad C(\{l\}, l) = a_{1l}, \quad \text{for any } l. \\ &\text{(b)}\ (n(S) > 1): \quad C(S, l) = \min_{m \in S-l} [C(S - l, m) + a_{ml}]. \end{aligned} \tag{5}$$

To see this, suppose that, in visiting the cities in S, terminating at city l, city m immediately precedes city l. Then, assuming that the other cities are visited in an optimum order, the cost incurred is $C(S - l, m) + a_{ml}$. Taking the minimum over all choices of m, we obtain (5b). Finally, if $\mathfrak{C}$ denotes the minimum cost of a complete tour, including the return to city one,

$$\mathfrak{C} = \min_{l \in \{2,3,\ldots,n\}} [C(\{2, 3, \cdots, n\}, l) + a_{l1}]. \tag{6}$$

[2] This formulation, discovered independently by the authors, is essentially identical to one proposed by Bellman in [2].

A permutation $(1\ i_2\ i_3 \cdots i_n)$ is optimum if, and only if,

$$\mathfrak{C} = C(\{2, 3, \cdots, n\}, i_n) + a_{i_n 1} \tag{7}$$

and, for $2 \leqq p \leqq n - 1$,

$$C(\{i_2, i_3, \cdots, i_p, i_{p+1}\}, i_{p+1}) = C(\{i_2, i_3, \cdots, i_p\}, i_p) + a_{i_p i_{p+1}}. \tag{8}$$

As in the scheduling problem, a two-phase computation is used to obtain an optimum solution. In the first phase, the quantities $C(S, l)$ are computed recursively from (5), and $\mathfrak{C}$ is computed from (6). In the second phase, (7) and (8) are used to compute an optimum permutation (i_n is determined first, and then, successively, $i_{n-1}, i_{n-2}, \cdots, i_2$).

Again, the fundamental operations employed in the computation are additions and comparisons. The number of each in the first phase is given by $\left(\sum_{k=2}^{n-1} k(k-1)\binom{n-1}{k}\right) + (n-1) = (n-1)(n-2)2^{n-3} + (n-1)$. The number of occurrences of each operation in the second phase is at most $\sum_{k=2}^{n-1} k = [n(n-1)/2] - 1$. If one storage location is assigned to each number $C(S, l)$, the number of locations required is

$$\sum_{k=1}^{n-1} k\binom{n-1}{k} = (n-1)2^{n-2}.$$

1.3. An assembly-line balancing problem. In the scheduling and traveling-salesman problems, all possible orderings of the elements (jobs or cities) were allowed. Many sequencing problems involve constraints which prohibit certain orderings from occurring. An advantage of the dynamic programming approach is that such constraints facilitate the solution, rather than hinder it. We illustrate this with reference to an assembly-line balancing problem posed by Salveson [10].

An assembly line is required to carry out, for each unit produced, a set of jobs denoted by $J_1, J_2, \cdots, J_n$. A given job J_k may be executed in τ_k units of time, and may be assigned to any of the locations (*work stations*) placed serially along the assembly line. The assignment is to be made so that:

(1) t_i, the sum of the execution times of jobs assigned to the ith work station does not exceed the *cycle time* T; $T - t_i$ is called the *idle time* at the ith station,

(2) all precedence requirements arising from the technology of the assembly process (we assume that these take the form "J_l must precede J_m") are satisfied,

(3) the number of work stations is minimized.

The existence of precedence constraints is reflected in the notions of feasible sets and feasible assignments. A given subset $S \subseteq \{1, 2, \cdots, n\}$

is said to be *feasible* if there is no pair (J_l, J_m) such that (a) $l \notin S$, (b) $m \in S$, and (c) J_l is required to precede J_m. An assignment of jobs to work stations 1, 2, $\cdots$, q is called *feasible*, if, for each $i \leqq q$, the set of jobs assigned to the first i work station is feasible. With any feasible set $S = \{k_1, k_2, \cdots, k_{n(S)}\}$ is associated a "cost" $C(S)$, which is the minimum, over all feasible assignments of the jobs with indices in S, of the quantity $(r-1)T + t_r$, where r is the number of work stations needed in the assembly line and t_r is the sum of the execution times of jobs assigned to the final work stations, according to a particular assignment. Then the quantity $C(S)$ may be computed according to the following recurrence relations, which hold *only* for feasible sets:

$$\text{(a)} \quad (n(S) = 1): \qquad C(\{l\}) = \tau_l$$

$$\text{(9)} \quad \text{(b)} \quad (n(S) > 1):$$

$$C(S) = \min_{\substack{l \in S \\ S-l \text{ feasible}}} [C(S-l) + \Delta(C(S-l), \tau_l)],$$

where[3]

$$\text{(i)} \quad \text{if } \left[\frac{x}{T}\right] = \left[\frac{x+y}{T}\right] \text{ or } \left[\frac{x+y}{T}\right] = \frac{x+y}{T}, \text{ then } \Delta(x, y) = y$$

$$\text{(ii)} \quad \text{if } \left[\frac{x}{T}\right] < \left[\frac{x+y}{T}\right] \text{ and } \left[\frac{x+y}{T}\right] < \frac{x+y}{T},$$

$$\text{then } \Delta(x, y) = T\left[\frac{x+y}{T}\right] + y - x.$$

The quantity $\Delta(C(S-l), \tau_l)$ denotes the incremental cost associated with assigning job J_l, assuming that the jobs specified by the elements of $S - l$ have already been assigned in an optimum manner.

The recurrence relations (9) are nearly identical to those used in the formulation of the scheduling problem (2), and a similar two-phase computation may be used to compute $C(\{1, 2, \cdots, n\})$, and thence to obtain an optimum assignment of an assembly line.[4]

1.4. Some extensions. The recurrence relations employed for the solution of the assembly-line balancing problem closely resemble those derived in treating the scheduling and traveling-salesman problems. The formulation of the line-balancing problem, however, necessarily included

[3] [ ] denotes "integer part of".

[4] The problem of cutting specified lengths $\tau_1, \tau_2, \cdots, \tau_n$ of stock from a minimum number of standard reels of length T may be treated by precisely the same recurrence relations, with all sets S taken as feasible. For linear programming approaches to the cutting stock problem see [5] and [7].

precedence relations, which were reflected in the restricted class of subsets considered. Precedence restrictions on the order of execution of jobs to be scheduled, or on the route of a traveling salesman, can be handled by similar means.

Another variation on the three problems under consideration involves the possibility of "parallelism": a set of cities to be visited may be distributed among several salesmen; assembly operations may be divided among several assembly lines; jobs may be scheduled on any of several facilities. As an illustration of this type of situation, we shall introduce some modifications into the scheduling problem.

Suppose that facilities $F^{(1)}, F^{(2)}, \cdots, F^{(v)}$ are available. Any given job J_k may be performed by any facility, with respective execution times $\tau_k^{(1)}, \tau_k^{(2)}, \cdots, \tau_k^{(v)}$. The cost associated with terminating J_k at time t on facility $F^{(i)}$ is given by $c_k^{(i)}(t)$. Then an optimum division of jobs among facilities, together with an optimum schedule for each facility, is obtained as follows:

(a) For each set S compute the functions $C^{(i)}(S)$ by means of (2), with $\tau_k^{(i)}$ and $c_k^{(i)}(t)$ substituted for τ_k and $c_k(t)$, respectively, in the ith computation.

(b) Compute the minimum[5], over all partitions of $\{1, 2, \cdots, n\}$ into sets $S^{(1)}, S^{(2)}, \cdots, S^{(v)}$, of the quantity $\sum_{i=1}^{v} C^{(i)}(S^{(i)})$; let this minimum occur for the partition $T^{(1)}, T^{(2)}, \cdots, T^{(v)}$.

(c) In the optimum schedule, the jobs in $T^{(i)}$ are performed on $F^{(i)}$. The order of execution on each facility is computed from (3), as before.

The numbers of operations in the algorithms under discussion can be considerably reduced when equivalences are noted among the elements to be ordered. Consider, for example, a scheduling situation in which the n jobs to be scheduled may be partitioned into q equivalence classes, such that any two jobs in the same class have the same execution time and cost function. Then, any set of jobs is characterized by a vector $V = (v_1, v_2, \cdots, v_q)$, specifying that the set contains v_l elements in the lth equivalence class; moreover, all sets for which the vector is the same may be treated as one. Then, if each class of sets is specified by a vector V, (2) may be replaced by

$$\begin{aligned} &\text{(a)} \quad C(0, 0, \cdots, 0) = 0 \\ &\text{(b)} \quad C(V) = \min_{1 \leq l \leq q \text{ and } v_l > 0} [C(V - e_l) + c_l(V \cdot \tau)]. \end{aligned} \tag{10}$$

Here $\tau = (\tau_1, \tau_2, \cdots, \tau_q)$, where τ_l is the execution time of a job in the lth equivalence class, $c_l(t)$ is the cost function for jobs in the lth equivalence

[5] It is possible to give a dynamic programming algorithm for the efficient computation of this minimum.

class, and $V - e_l$ is a vector obtained by subtracting one from the lth component of V.[6]

2. SUCCESSIVE APPROXIMATIONS

It is characteristic of the algorithms under discussion that their complexity, measured by numbers of arithmetic operations and storage requirements, grows quite rapidly. They are, however, a vast improvement over complete enumeration, and permit the rapid direct solution of problems of moderate size.[7] In this section we show how the algorithms can be combined with a method of successive approximations to provide a systematic procedure for treating large problems. This procedure yields a sequence of permutations, each obtained from its predecessor by the solution of a derived subproblem of moderate size having the same structure as the given problem. The associated costs form a monotone nonincreasing sequence which may not converge to the optimum solution; however, computer experimentation has yielded excellent results in a variety of cases (cf. §3). As an illustration, we shall specify a method of successive approximations for solving large traveling-salesman problems.

Given a permutation $P = (1\ i_2 \cdots i_n)$ representing a route through n cities, the cities may be partitioned into u ordered sets, each consisting of cities which occur successively in P, and maintaining the same order as in P. A u-city traveling-salesman problem is then solved in which each ordered set is treated as a city, and the cost of going from the set $(i_j\, i_{j+1} \cdots i_{k-1}\, i_k)$ to $(i_l\, i_{l+1} \cdots i_{m-1}\, i_m)$ is $a_{i_k i_l}$. If u is not too large, this derived problem may be solved by the dynamic programming algorithm of §1.2. The solution implies a reordering P' of P, with P' having cost less than or equal to that of P.

Two types of partitions have proved especially useful. In the first type, called a *local partition*, each of the ordered sets but one consists of a single element, so that the tours associated with P and P' differ locally, if at all. At the other extreme, a *global partition* takes the u sets as nearly equal in size as possible, so that, if changes are made, they tend to be of a global nature. Examples of these two types of partitions, and of possible reorderings associated with them, are shown in Figs. 1a and 1b. In these examples, $n = 16$ and $u = 8$. In the solution of large problems, it has proved desirable to employ alternate phases of local and global improvement.

Through experience with a computer program, systematic procedures

[6] This division into equivalence classes is a particularly useful device in the problem of cutting stock, where it is usually necessary to cut many rolls of the same length. The number of equivalence classes is then equal to the number of different lengths to be cut.

[7] An IBM 7090 program can solve any 13-city traveling-salesman problem in 17 seconds.

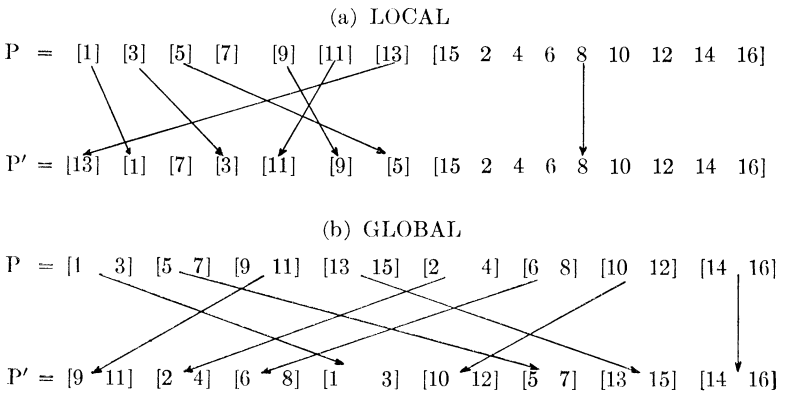

FIG. 1. *Examples of local and global improvements.*

for the selection of derived problems have been worked out. As part of this selection process, a technique was obtained for determining whether a given derived problem is likely to yield an improvement (i.e., cost of $P' <$ cost of P). Each derived problem is specified by a $u \times u$ cost matrix $C = (c_{ij})$, with a row and column for each set of cities to be rearranged. We assume that the rows and columns of this matrix are so ordered that the *slant elements* $c_{u1}, c_{12}, \cdots, c_{u-1,u}$ give the cost of the transitions which occur in the current permutation P. If the cost of P' is to be strictly less than the cost of P, the optimum tour for the derived traveling-salesman problem defined by C must employ transitions corresponding to off-slant elements of C. The ratio of slant elements which are row or column minima to the total number u of slant elements has proved to be a reliable measure of the "promise" of a derived problem. At any point in the computation, matrices for which this *slant ratio* exceeds a critical value (which may be adjusted as the computation proceeds) are not considered further.

If no such selection procedure is used, it can be shown that the successive approximation technique yields the same final tour for any two equivalent traveling-salesman problems. Two problems with associated $n \times n$ cost matrices (a_{ij}) and (b_{ij}) are said to be *equivalent* if there are constants α and β such that, for any cyclic permutation Γ of the indices $1, 2, \cdots, n$,

$$\sum_{\Gamma} a_{ij} = \alpha \sum_{\Gamma} b_{ij} + \beta.$$

Since corresponding submatrices of equivalence matrices (a_{ij}) and (b_{ij}) need not have the same slant ratio, the invariance of the computation under equivalence transformations is lost when selection is employed. However, it is possible to derive a canonical representative for each equivalence class (see Appendix 1) and invariance, if desired, can be maintained by transforming any given problem into its canonical form.

3. THE COMPUTER PROGRAM

In this section, we present some computational results obtained with an IBM 7090 program for the traveling-salesman problem. This program solves problems involving 13 or fewer cities by direct application of the dynamic programming algorithm of §1.2; larger problems are treated by means of the successive approximation technique of §2.

3.1. Specifications. In the program implementing the successive approximation procedure, each derived problem is of size 13. The program proceeds in alternating phases of local and global improvement. Each local improvement phase cycles through the n possible derived problems corresponding to the optimal reordering of consecutive cities, until a full cycle is considered without any improvement. At this point, the slant ratio threshold is increased, and, if it does not exceed an upper bound prescribed for the current phase, the local improvement continues; otherwise a global improvement phase is entered. During each global improvement phase, at most $n + 1$ derived problems are considered. The first of these is obtained by choosing a partition which maximizes the sum of the slant elements in the derived cost matrix, thus tending to make the slant ratio small. The program selects the n remaining problems by a definite but arbitrary rule which makes the sizes of the 13 sets as nearly equal as possible. The derived problems are examined in turn, and for those such that the slant ratio does not exceed the current threshold, a dynamic programming calculation is performed to obtain an optimum tour. As soon as a derived problem yields a solution better than that associated with its slant, the improvement is incorporated into the over-all problem, and a new phase of local improvement is initiated. If no derived problem yields an improvement, the slant-ratio threshold is increased, and, if it does not exceed an upper bound specified for the current phase, the problems are re-examined; otherwise a new phase of local improvement is initiated, and no global improvement is made at this stage of the calculation.

The calculation proceeds until a point of diminishing returns is reached, as measured by an empirical stopping criterion involving the number of derived problems treated by the dynamic programming algorithm without success in any given phase. In any event, the program is terminated after the fifth phase of global improvement.

The strategy of the calculation, including the choice of derived problems, the adjustment of slant-ratio thresholds, and the termination of the computation, has been suggested by computational experience. Within wide limits the successful performance of the program does not depend critically on these choices.

3.2. Results. We now present six examples which were treated by the IBM 7090 computer program. In these examples, the execution time for a

run varied between two minutes and fifteen minutes. In each case, the initial tour was chosen at random.

(i) The 42-city problem of Dantzig, Fulkerson, and Johnson. In [4] the cost of an optimum tour was shown by a special argument to be 699.

Run No.	*Cost of Initial Tour*	*Cost of Final Tour*
1	3235	699
2	3513	705
3	3207	704
4	3246	699
5	2913	704

(ii) A 20-city problem due to Croes. In [3], Croes states that the cost of an optimum tour is 246.

Run No.	*Cost of Initial Tour*	*Cost of Final Tour*
1	732	246
2	816	246
3	811	246

(iii) A 48-square knight's tour. If the distance between two squares of a chessboard is taken as the minimum number of knight's moves required to reach one square from the other, the problem of finding a closed knight's tour (a tour of the board by knight's moves, such that each square is visited exactly once, and terminating with a return to the initial square) may be treated as a traveling-salesman problem. Since a closed knight's tour is known to exist for an abbreviated 6×8 chessboard [8], the corresponding traveling-salesman problem has an optimum solution with cost 48.

Run No.	*Cost of Initial Tour*	*Cost of Final Tour*
1	132	56
2	132	52
3	124	54
4	132	56

(iv) A 36-square king's tour. A closed king's tour may be formulated as a traveling-salesman problem in the same manner as was the knight's tour. Obviously a closed king's tour exists for a 6×6 chessboard; hence, the corresponding traveling-salesman problem has an optimum solution with cost 36.

Run No.	*Cost of Initial Tour*	*Cost of Final Tour*
1	87	36
2	103	36

(v) A tour of 48 cities in the United States. Machine results coupled with visual inspection suggest the conjecture that an optimum tour has cost 11,470. The matrix for this problem and the conjectured optimum tour are given in Fig. 2.

	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	
1		1157	1353	1560	1461	486	917	361	[illegible]	1581	917	932	1163	352	1231	548	515	626	1176	341	582	1266	231	676	48
2	273		2010	2238	2435	1553	1835	796	[illegible]	2281	268	2044	220	1021	194	1187	739	861	1212	923	603	2336	1038	641	47
3	1272	999		229	727	1614	550	1500	[illegible]	2580	1908	1019	2151	1146	2179	1898	1752	1166	2520	1650	1717	940	1573	1890	46
4	744	809	1519		678	1787	701	1724	[illegible]	115	2138	1114	2380	1369	2408	2101	1975	1395	2734	1867	1943	959	1785	2114	45
5	1138	866	140	1425		1490	600	1744	[illegible]	531	2264	653	2523	1429	2571	1926	1963	1584	2594	1802	1986	347	1688	2121	44
6	1972	1722	937	1861	1052		1087	792	[illegible]	1720	1291	849	1507	828	1584	507	823	1110	1148	631	949	1204	516	959	43
7	1580	1338	697	1473	776	400		1162	[illegible]	638	1668	504	1925	833	1972	1434	1393	987	2089	1250	1402	579	1148	1547	42
8	1878	1640	951	1713	1049	182	304		[illegible]	1725	559	1272	811	354	875	598	254	466	1061	240	242	1589	291	392	41
9	1539	1266	267	1761	402	820	699	884		2477	330	1903	345	1008	426	776	457	996	695	645	429	2237	774	296	40
10	1457	1185	227	1617	361	721	538	755	177		2165	1005	2412	1371	2445	2063	1969	1427	2708	1849	1950	825	1759	2115	39
11	429	440	1229	370	1119	1735	1335	1612	1486	1362		1828	259	836	317	919	472	743	1010	660	344	2136	779	375	38
12	1129	894	587	1073	578	851	454	749	757	587	891		2083	1036	2145	1316	1446	1280	1982	1263	1508	357	1137	1607	37
13	1251	992	369	1304	406	740	393	690	506	335	1082	252		1094	83	1089	685	985	1019	881	582	2394	1007	552	36
14	1421	1173	554	1369	618	551	173	476	609	435	1199	308	222		1144	847	607	290	1397	529	580	1315	492	745	35
15	588	334	721	1092	581	1551	1198	1501	981	930	726	803	814	1025		1171	761	1018	1092	957	652	2421	1081	632	34
16	334	358	1212	453	1095	1769	1370	1654	1474	1358	96	920	1094	1227	663		469	1051	669	358	623	1673	355	547	33
17	837	626	739	798	670	1159	760	1049	967	819	583	309	510	617	632	610		697	815	197	154	1780	325	161	32
18	1364	1124	596	1283	641	613	216	516	681	504	1125	238	235	90	999	1156	546		1511	699	605	1524	708	792	31
19	229	358	1291	973	1152	2072	1692	1995	1552	1496	653	1252	1335	1525	572	557	983	1479		872	919	2339	949	736	30
20	961	847	1114	565	1060	1300	919	1149	1317	1153	563	569	820	835	972	642	397	745	1163		320	1603	129	350	29
21	754	533	701	1315	567	1605	1286	1580	936	927	947	940	892	1114	225	879	821	1105	[illegible]	1183		1829	435	188	28
22	1169	915	426	1204	433	807	435	739	594	428	986	165	100	263	763	1000	411	240	[illegible]	725	865		1480	1941	27
23	1488	1219	285	1796	374	1017	879	1079	197	341	1493	863	626	770	908	1467	1023	831	[illegible]	1399	821	699		479	26
24	720	481	676	846	579	1251	861	1161	928	803	560	414	541	700	451	558	180	645	[illegible]	549	644	453	950		25
25	1280	1009	155	1447	235	818	548	815	316	180	1183	454	219	400	767	1178	651	442	[illegible]	1004	790	290	410	624	
26	816	543	456	1143	325	1259	913	1214	723	649	813	552	524	740	293	780	478	723	[illegible]	869	388	483	690	325	
27	664	937	1936	959	1802	2596	2198	2485	2203	2119	882	1745	1897	2049	1240	831	1438	1983	[illegible]	1427	1374	1809	2147	1356	
28	1178	915	319	1275	331	826	483	780	500	343	1033	269	90	311	726	1038	476	316	[illegible]	818	803	107	594	480	
29	939	667	337	1213	217	1137	803	1100	604	521	902	482	410	630	420	879	485	623	[illegible]	882	484	384	590	369	
30	1698	1441	604	2085	665	1255	1181	1347	482	652	1763	1188	952	1087	1111	1726	1333	1152	[illegible]	1716	968	1024	326	1241	
31	983	812	907	742	862	1123	731	985	1104	939	642	355	605	630	862	700	235	543	[illegible]	214	1056	511	1191	413	
32	1119	848	214	1309	182	943	627	916	455	340	1032	397	238	459	617	1023	525	470	[illegible]	902	665	251	499	473	
33	1029	776	424	1479	312	1359	1086	1361	630	649	1131	833	706	924	443	1082	827	939	[illegible]	1222	318	712	504	680	
34	1815	1560	748	1760	864	188	292	260	641	533	1604	713	570	405	1374	1631	1022	482	[illegible]	1210	1420	646	838	1097	
35	721	526	817	703	732	1282	883	1171	1058	918	463	432	622	739	586	488	123	669	[illegible]	390	794	525	1098	166	
36	1753	1494	666	1727	783	271	279	328	562	451	1556	666	503	360	1299	1579	973	443	[illegible]	1184	1341	585	758	1038	
37	330	598	1592	872	1456	2300	1906	2202	1857	1783	663	1453	1581	1749	887	586	1155	1690	[illegible]	1225	1017	1499	1794	1049	
38	1499	1244	521	1479	608	483	178	445	528	362	1298	410	257	115	1070	1320	715	205	[illegible]	949	1137	330	703	781	
39	1107	1304	2172	686	2066	2540	2156	2385	2425	2290	947	1758	1985	2055	1633	982	1475	1969	[illegible]	1239	1836	1885	2439	1497	
40	1576	1306	356	1698	491	609	490	665	220	130	1461	642	396	428	1057	1463	902	510	[illegible]	1210	1056	495	414	905	
41	942	685	467	1057	400	1038	662	966	704	568	795	262	309	492	547	796	273	455	[illegible]	660	679	231	751	238	
42	484	668	1583	387	1466	2099	1699	1969	1845	1727	371	1260	1453	1568	999	371	953	1492	[illegible]	863	1200	1356	1837	925	
43	617	444	882	1252	744	1766	1430	1729	1122	1105	882	1051	1039	1256	252	802	882	1238	[illegible]	1207	189	999	1011	702	
44	896	1157	2139	904	2013	2699	2300	2568	2405	2301	967	1858	2043	2166	1483	949	1550	2091	[illegible]	1446	1645	1949	2374	1506	
45	1184	1359	2182	668	2082	2493	2117	2333	2428	2285	973	1737	1972	2026	1681	1021	1467	1938	[illegible]	1197	1891	1872	2455	1506	
46	1030	1176	1961	443	1865	2266	1888	2108	2204	2059	768	1508	1744	1796	1489	826	1240	1709	[illegible]	969	1704	1644	2237	1287	
47	1718	1475	781	1600	875	264	138	177	738	595	1472	592	514	303	1326	1508	898	354	[illegible]	1042	1403	567	928	998	
48	604	335	678	930	552	1398	1023	1327	945	853	588	598	661	853	236	550	396	813	[illegible]	741	442	591	921	216	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	

Conjectured optimum tour: (6 34 36 40 10 9 23 30 33 21 43 15 2 1 19 37 27 44 39 45 46 4 42 16 11 20 31 17 35 24 48 26 29 5 3 25 32 28 13 22 41 12 18 14 38 7 47 8).
Associated Cost: 11,470.

FIG. 2. *48-city problem.*

Note. This matrix is symmetric. For display purposes, as the numbering shows, the tail has been rotated.

2	182																							
3	86	107																						
4	129	142	167																					
5	211	195	249	82																				
6	185	22	99	164	217																			
7	185	50	99	115	175	72																		
8	41	144	58	163	245	144	144																	
9	174	85	88	220	280	63	127	146																
10	106	165	151	23	105	187	138	140	237															
11	18	200	104	146	228	203	203	59	192	123														
12	73	109	26	148	230	125	117	46	112	125	91													
13	186	167	100	267	349	145	199	158	82	251	204	126												
14	71	114	50	124	206	114	114	39	136	113	89	24	150											
15	26	208	112	136	218	211	211	67	200	113	22	99	212	97										
16	233	128	208	270	323	119	167	201	120	275	251	186	158	162	259									
17	288	231	252	341	423	209	271	256	178	330	306	241	189	217	314	114								
18	212	36	126	178	231	27	75	171	88	201	230	145	170	141	238	92	203							
19	201	144	167	254	336	122	184	169	79	243	219	154	96	130	227.	68	99	116						
20	183	126	162	236	318	104	166	151	95	225	201	136	114	112	209	50	105	98	18					
21	290	231	264	343	425	211	270	258	180	332	308	243	187	219	316	103	36	195	101	107				
22	208	45	122	187	240	23	85	167	65	210	226	148	147	137	234	108	186	23	99	81	188			
23	93	272	179	158	240	272	272	134	267	135	87	166	279	158	67	320	375	299	288	270	377	295		
24	373	305	308	426	500	283	355	341	240	415	391	326	208	302	399	206	94	288	193	190	128	271	460	
25	453	396	417	506	588	374	436	421	343	495	471	406	321	382	479	279	165	368	264	270	195	351	540	113
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Conjectured optimum tour: (5 4 10 23 15 11 1 8 14 12 3 13 24 25 17 21 16 20 19 9 22 18 6 2 7).
Associated Cost: 1711.

FIG. 3. *25-city problem.*

Run No.	*Cost of Initial Tour*	*Cost of Final Tour*
1	50,716	11,566
2	42,288	12,116
3	45,400	11,984
4	51,223	11,688

(vi) A 25-city problem for which the cost of an optimum tour is believed to be 1711. The cost matrix for this problem and the probable optimum tour are given in Fig. 3.

Run No.	*Cost of Initial Tour*	*Cost of F nal Tour*
1	5626	1711
2	5838	1711
3	5115	1711

4. CONCLUSIONS

The dynamic programming approach set forth in this paper is by no means a universal key to the solution of sequencing problems. However, it can be employed in a number of important situations, and, when applicable, offers several advantages. For small problems, optimum solutions can be obtained with only a small, and precisely predictable, amount of computation. Larger problems lend themselves to a successive-approximation technique which, although lacking rigorous justification, has worked out remarkably well in practice.[8] Finally, these techniques can be mechanized completely by rather simple computer programs.

APPENDIX 1

REDUCTION OF A MATRIX TO CANONICAL FORM

In §2, two n-city traveling-salesman problems defined by matrices (a_{ij}) and (b_{ij}) were said to be *equivalent* if there existed constants α and β such that, for any tour Γ,

$$\sum_\Gamma a_{ij} = \alpha \sum_\Gamma b_{ij} + \beta.$$

In this appendix, we characterize this equivalence directly in terms of the defining matrices, and show how to obtain a canonical representative of each equivalence class.

In the discussion that follows, it is convenient to define four special classes of matrices: (e_{ij}) is a *constant-row matrix* (*constant-column matrix*) if all its elements are zero, except for a single row (column), all of whose off-diagonal elements are equal (the diagonal element is arbitrary); (e_{ij}) is a *potential matrix*[9] if for every triple i, j, k, $e_{ij} + e_{jk} = e_{ik}$; (e_{ij}) is an

[8] An IBM 7090 program applying the successive approximation technique to the single-facility scheduling problem of §1.1 has met with success comparable to that reported above for the traveling-salesman problem.

[9] Any potential matrix defines a traveling-salesman problem for which the cost of any closed tour is zero.

image matrix if its diagonal elements are equal to zero, and each of its row and column sums is zero.

We now define a homomorphic mapping T of the additive group G of $n \times n$ (real) matrices onto the additive group $\bar{G}$ of $n \times n$ (real) image matrices. Given $(d_{ij}) \in G$ we define the mapping in two steps:

$$(1) \qquad d'_{ij} = n\, d_{ij} - \sum_{k=1}^{n} d_{ik} - \sum_{k=1}^{n} d_{kj}$$

$$(2) \qquad \bar{d}_{ij} = d'_{ij} + \sum_{k=1}^{n} \alpha_k [n\delta_{ik}\delta_{jk} - \delta_{ik} - \delta_{jk}] + \alpha_{n+1} .$$

Here, δ_{lm} is the Kronecker delta and $\alpha_1, \alpha_2, \cdots, \alpha_{n+1}$ are determined uniquely from a system of $n + 1$ linear equations[10] expressing the conditions that $\bar{d}_{ii} = 0$ for all i and $\sum_{i=1}^{n} \sum_{j=1}^{n} \bar{d}_{ij} = 0$. It is easily seen that $(\bar{d}_{ij})$ belongs to $\bar{G}$.

It can be shown that the kernel N of the map T is the normal subgroup of G generated by the set of constant-row, constant-column, and potential matrices. It follows, by the law of homomorphism for groups [11], that two matrices have the same image under the mapping T if and only if each can be derived from the other by the addition of constant row, constant column, and potential matrices.

The *canonical form* of a matrix is its image under the mapping T. Two matrices are said to be *equivalent* if each can be derived from the other using only the following operations: (a) addition of constant-row and constant-column matrices, (b) addition of potential matrices, and (c) multiplication of a matrix by a scalar constant. Thus, two matrices are equivalent if and only if their canonical forms differ by a constant scalar multiplicative factor.

Finally, it can be shown that two $n \times n$ traveling-salesman problems are equivalent if and only if their defining matrices are equivalent, so that an equivalence class of traveling-salesman problems is characterized by the canonical form of any one of the associated matrices.

Acknowledgments. The authors are particularly indebted to Mr. R. Shareshian of IBM's Mathematics and Applications Department, who was largely responsible for the preparation of the traveling-salesman computer program mentioned here. Acknowledgments are due as well to Dr. A. Bomberault of the same department for several stimulating discussions, particularly with regard to the canonical representation given in Appendix 1.

REFERENCES

1. R. L. Ackoff (ed.), *Progress In Operations Research*, Vol. I, Wiley, New York, 1961.

[10] This system will be nonsingular unless $n = 2$.

2. R. BELLMAN, *Combinatorial processes and dynamic programming*, Proceedings of the Tenth Symposium in Applied Mathematics of the American Mathematical Society, 1960; also see his later paper *Dynamic programming treatment of the traveling salesman problem*, J. Assoc. Comput. Mach., 9 (1962). pp. 61–63.
3. G. A. CROES, *A method for solving traveling-salesman problems*, Operations Res., 6 (1958), pp. 791–812.
4. G. B. DANTZIG, D. R. FULKERSON, AND S. M. JOHNSON, *Solution of a large-scale traveling salesman problem*, Operations Res., 2 (1954), pp. 393–410; also see these authors' later paper *On a linear programming, combinatorial approach to the traveling salesman problem*, Operations Res., 7 (1959), pp. 58–66.
5. K. EISEMANN, *The trim problem*, Management Sci., 3 (1957), pp. 279–284.
6. M. M. FLOOD, *The traveling-salesman problem*, Operations Res., 4 (1956), pp. 61–75.
7. P. C. GILMORE, AND R. E. GOMORY, *A linear programming approach to the cutting stock problem*, Operations Res., 9 (1961), pp. 849–859.
8. M. KRAITCHIK, *Mathematical recreations*, Dover, New York, 1953, Chapter 11.
9. R. MCNAUGHTON, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1–12.
10. M. E. SALVESON, *The assembly line balancing problem*, Journal of Industrial Engineering, 6 (1955), pp. 18–25.
11. B. L. VAN DER WAERDEN, *Modern Algebra*, Vol. I, Ungar, New York, 1950.