stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 38/74 NOVEMBER

J.K. LENSTRA & A.H.G. RINNOOY KAN

SOME SIMPLE APPLICATIONS OF THE TRAVELLING SALESMAN PROBLEM

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS (MOS) subject classification scheme (1970): 90B35, 90C50

SOME SIMPLE APPLICATIONS OF THE TRAVELLING SALESMAN PROBLEM

J.K. LENSTRA
Mathematisch Centrum, Amsterdam, The Netherlands

A.H.G. RINNOOY KAN
Graduate School of Management, Delft, The Netherlands

ABSTRACT

The travelling salesman problem arises in many different contexts. In this paper we report on typical applications in computer wiring, vehicle routing, clustering and job-shop scheduling. We show that the formulation as a travelling salesman problem is essentially the simplest way to solve these problems by establishing complete equivalence. Most applications originated from real world problems and thus seem to be of particular interest. Illustrated examples are provided with each application.

NOTE

This paper has been published simultaneously by the Mathematisch Centrum as Report BW 38 and by the Graduate School of Management as Working Paper WP/74/12.

KEY WORDS AND PHRASES: *travelling salesman problem, computer wiring, vehicle routing, clustering a data array, job-shop scheduling with no intermediate storage.*

CONTENTS

1. THE TRAVELLING SALESMAN PROBLEM

1.1. Formulation

A salesman wishes to find the shortest route through a number of cities and back home again. This problem is known as the *travelling salesman problem* and can be stated more formally as follows.

Given a finite set of cities N and a distance matrix (c_{ij}) $(i,j \in N)$, determine

$$\min_\pi \sum_{i \in N} c_{i\pi(i)}$$

where π runs over all *cyclic permutations* of N (i.e. $\pi^{|N|}(i) = i$, $i \in N$); $\pi^k(i)$ is the k-th city reached by the salesman from city i. If $N = \{1,\ldots,n\}$, then an equivalent formulation is

$$\min_\nu \left(\sum_{i=1}^{i=n-1} c_{\nu(i)\nu(i+1)} + c_{\nu(n)\nu(1)}\right)$$

where ν runs over all *permutations* of N; here $\nu(k)$ is the k-th city in a salesman's tour. If G denotes the complete directed graph on the vertex set N with a weight c_{ij} for each arc (i,j), then an optimal tour corresponds to a *hamiltonian circuit* on G (i.e. a circuit passing through each vertex exactly once) of minimum total weight.

If $c_{ij} = c_{ji}$ for all (i,j), the problem is called *symmetric*, otherwise it is called *asymmetric*. If $c_{ik} \leq c_{ij} + c_{jk}$ for all (i,j,k), the problem is called *euclidean*.

1.2. Applications

The number of applications of the TSP is surprisingly large; the problem arises in widely varying contexts, such as scheduling, sequencing, distribution, routing and location decisions. In this paper we report on four typical applications in *computer wiring*, *vehicle routing*, *clustering a data array* and *job-shop scheduling with no intermediate storage*.

For the last two applications, their complete equivalence to the TSP is non-trivial and will be established in sections 4.3 and 5.3. Formulation as a TSP thus is essentially the simplest way to solve these problems.

Three of the applications originated from real world problems that were not immediately recognized as TSPs; their interpretation as a TSP led to better solutions, as will be amply illustrated in sections 2.3, 3.3 and 4.4.

1.3. Solution methods

In [2], [13] and [5] recent surveys of known solution methods are presented.

We can distinguish between *optimal* and *suboptimal* algorithms. The first type of algorithm produces solutions that are *guaranteed to be optimal* but may require inordinate running times; of special interest are the branch-and-bound methods developed by Little, Murty, Sweeney and Karel [23], Held and Karp [11;12;10] and Bellmore and Malone [1]. Suboptimal algorithms produce *approximate* solutions in reasonable times; we mention the successful heuristic methods of Lin [21], Christofides and Eilon [3] and Lin and Kernighan [22].

In fact, we shall be using the following algorithms:

(a) a branch-and-bound procedure based on [23], incorporating an improved branching strategy that allows early pruning of a branch through sufficiently large penalties;

(b) a branch-and-bound procedure based on [12] for symmetric TSPs;

(c) a heuristic procedure for generating *3-optimal* tours for symmetric TSPs, following the enumeration scheme given by Lin [21,p.2266] with deletion of some superfluous checks for improvement.

Descriptions of these algorithms as well as computational experience and ALGOL 60-procedures can be found in [18].

2. COMPUTER WIRING

2.1. Problem description

The following problem arises frequently during the design of computer interfaces at the Institute for Nuclear Physical Research in Amsterdam.

An interface consists of a number of modules, and on each module several pins are located. The position of each module has been determined in advance. A given subset of pins has to be interconnected by wires. In view of possible future changes or corrections and of the small size of the pin, at most two wires are to be attached to any pin. In order to avoid signal cross-talk and to improve ease and neatness of wirability, the total wire length has to be minimized.

2.2. TSP formulation

Let P denote the set of pins to be interconnected, c_{ij} the distance between pin i and pin j, and H the complete graph on the vertex set P with weights c_{ij} on the arcs.

If any number of wires could be attached to a pin, an optimal wiring would correspond to a minimum *spanning tree* on H, which can be found efficiently by the algorithms of Kruskal [16] or Prim [28] and Dijkstra [4]. However, the degree requirement implies that we have to find a minimum *hamiltonian path* on H (i.a. a path passing through each vertex exactly once). This problem corresponds to finding a minimum *hamiltonian circuit* on G with $N = P \cup \{*\}$ and $c_{i*} = c_{*i} = 0$ for all $i \in N$. The wiring problem can thus be converted into a symmetric euclidean TSP.

A more difficult problem occurs if the positions of the modules have not been fixed in advance but can be chosen so as to minimize the total wire length for all subsets of pins that have to be interconnected. For a review of this *placement problem* and the associated *quadratic assignment problem* we refer to [9].

2.3. Results

The procedure that was used originally produced clearly non-optimal wiring schemes like the example with two subsets of pins in Figure 1a. The size and number of the problems was such that Lin's heuristic had to be used. The 3-optimal results on the example are given in Figure 1b.

More examples and details about the computer implementation can be found in [36].

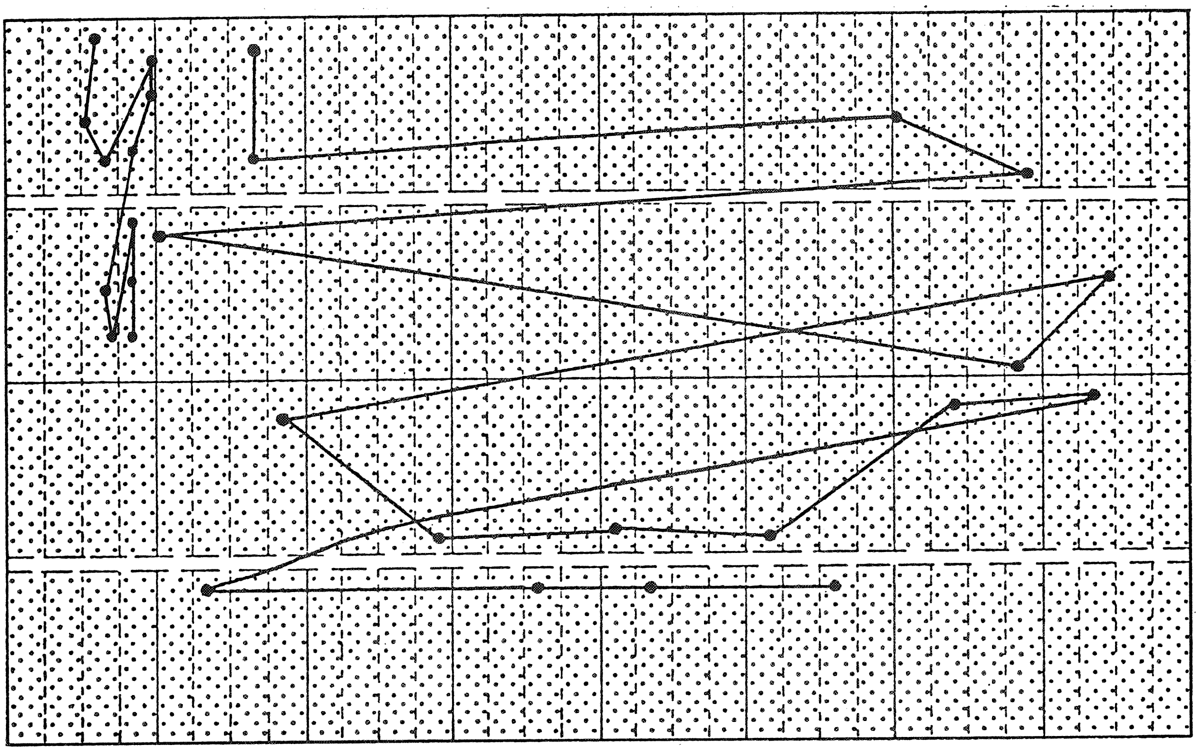


Figure 1a Wiring without optimization.

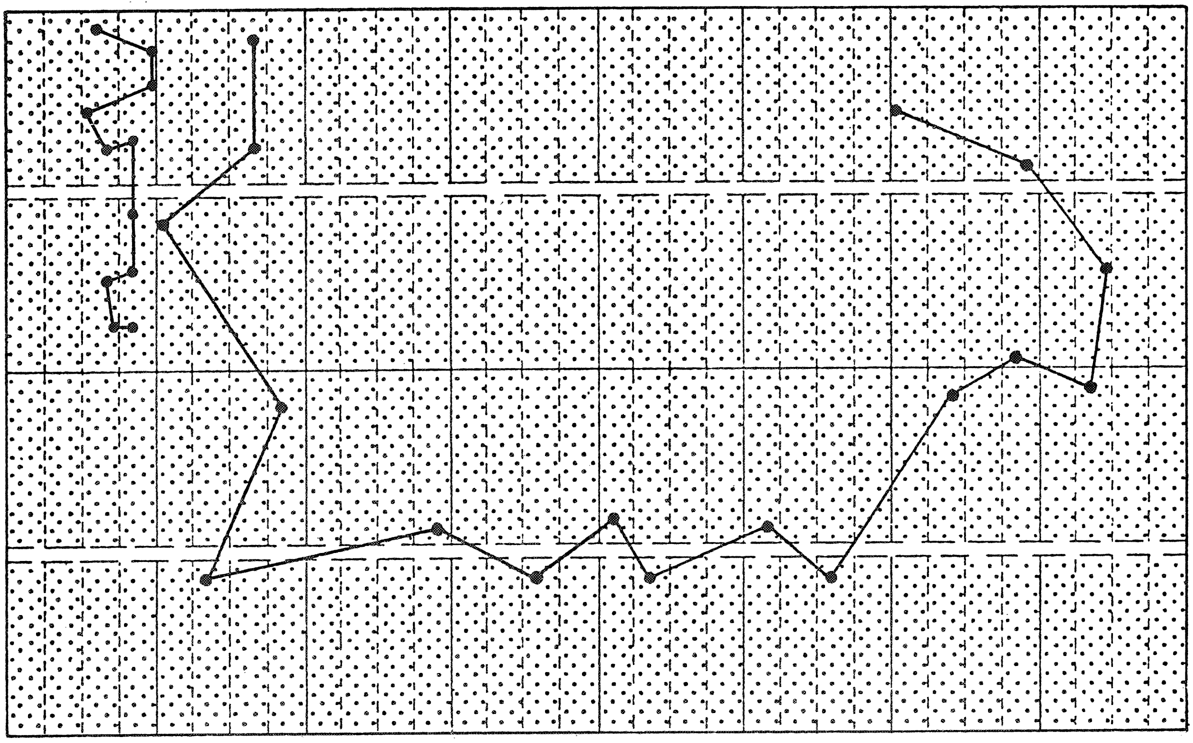


Figure 1b 3-Optimal wiring.

3. VEHICLE ROUTING

3.1. Problem description

In 28 towns in the Dutch province of North-Holland telephone-boxes have been installed by the national postal service (PTT). A technical crew has to visit each telephone-box once or twice a week to empty the coin-box and, if necessary, to replace directories and perform minor repairs. Each working day of at most 445 minutes begins and ends in the provincial capital Haarlem. The problem is to minimize the number of days in which all telephone-boxes can be visited and the total travelling time.

A similar problem arose in the city of Utrecht. Here ca. 200 mail-boxes have to be emptied each day within a period of one hour by trucks operating from the central railway station. The problem is to find the minimum number of trucks able to do this and the associated minimum travelling time.

3.2. TSP formulation

Both problems are types of classical *vehicle routing problems* (VRP), that are extensively discussed in [5,Ch.9]. They will be denoted by P1 and P2, respectively, and can be characterized more formally as follows.

- n cities i ($1 \le i \le n$) (the *customers*) are to be visited
  [P1: 28 towns; P2: 200 mail-boxes]
- by m *vehicles*
  [P1: m working days; P2: m trucks]
- operating from city * (the *depot*)
  [P1: Haarlem; P2: central railway station];
- the travelling time between cities i and j is $d_{ij} = d_{ji}$ minutes, for $i,j \in \{1,\dots,n\} \cup \{*\}$;
- the time to be spent in city i is e_i minutes, for $i \in \{1,\dots,n\}$
  [P1: 8 × number of telephone-boxes in town i; P2: 1];
- the maximum allowable time for any vehicle to complete its route is f minutes
  [P1: 445; P2: 60];

- there may be additional constraints

  [P1: one town (nr.28, Den Helder) has to be visited twice on different days];
- criteria by which solutions are judged are:

  A , the number of vehicles used;

  B(A), the total time used for A vehicles.

If a city has to be visited twice, it is duplicated, appropriate travelling and visiting times are added, and n is increased by one.

[P1: Den Helder is split up into two cities 28 and 29; $d_{28\ 29} := \infty$; $n := 29$.]

We replace the depot (city $*$) by m artificial depots (cities $n+1,\dots,n+m$) and extend the definition of (d_{ij}) and (e_i) as follows (cf. Figure 2):

$$\begin{aligned} d_{i\ n+\ell} &= d_{i*} && \text{for } 1 \le \ell \le m;\\ d_{n+k\ j} &= d_{*j} && \text{for } 1 \le k \le m;\\ d_{n+k\ n+\ell} &= \lambda && \text{for } 1 \le k,\ell \le m;\\ e_{n+k} &= 0 && \text{for } 1 \le k \le m. \end{aligned}$$

$$\begin{array}{c|ccccc|ccc|}
 & 1 & \dots j & \dots n & & & n+1 & \dots & n+m \\ \hline
1 & d_{11} & \dots d_{1j} & \dots d_{1n} & & & d_{1*} & \dots & d_{1*} \\
\vdots & \vdots & \vdots & \vdots & & & \vdots & & \vdots \\
i & d_{i1} & \dots d_{ij} & \dots d_{in} & & & d_{i*} & \dots & d_{i*} \\
\vdots & \vdots & \vdots & \vdots & & & \vdots & & \vdots \\
n & d_{n1} & \dots d_{nj} & \dots d_{nn} & & & d_{n*} & \dots & d_{n*} \\ \hline
n+1 & d_{*1} & \dots d_{*j} & \dots d_{*n} & & & \lambda & \dots & \lambda \\
\vdots & \vdots & \vdots & \vdots & & & \vdots & & \vdots \\
n+m & d_{*1} & \dots d_{*j} & \dots d_{*n} & & & \lambda & \dots & \lambda \\ \hline
\end{array}$$

<u>Figure 2</u> The matrix (d_{ij})

We obtain a symmetric euclidean TSP by defining $N = \{1,\dots,n+m\}$ and $c_{ij} = \frac{1}{2}e_i + d_{ij} + \frac{1}{2}e_j$ for all $i,j \in N$. A salesman's tour is feasible for the VRP provided that the time constraint for each vehicle and possible additional constraints are respected. If a TSP solution contains m-A links between artificial depots, then the corresponding VRP solution uses only A vehicles. The choice of λ now becomes important.

- $\lambda = +\infty$ will lead to $\min_{\pi} B(m)$,
  i.e. the minimum total time for m vehicles (cf. [5,p.188]);
- $\lambda = 0$ will lead to $\min_{\pi}\{B(A) | 1 \leq A \leq m\}$,
  i.e. the minimum total time for any number of vehicles (cf. [5,p.188]);
- $\lambda = -\infty$ will lead to $\min_{\pi} B(\min\{A | 1 \leq A \leq m\})$,
  i.e. the minimum total time for the minimum number of vehicles.

The latter objective is the criterion function for both P1 and P2.

An appropriate method for obtaining good VRP solutions is the following.

- Choose an initial tour which satisfies the VRP constraints.
- Apply an iterative procedure for improving the tour and check the constraints whenever a possible decrease in tour length occurs.

An interesting variation on this type of problem arises in the context of money collection at post-offices. For security reasons, several good routes have to be available. The problem is then equivalent to the *moonlighting salesman problem* [15], where k disjoint hamiltonian circuits of minimum total weight are sought. No algorithms for this problem have been proposed so far.

3.3. Results

Figures 3 and 4 illustrate some results, obtained for P1 and P2. In both figures, the links with the depot (*) have not been drawn.

For P1, Lin's heuristic method was used. All 3-optimal solutions obtained require four days, representing a 50 percent decrease with respect to the schedule that was previously used. An example is given in Figure 3a. Exchanging three links in this solution resulted in the schedule given in Figure 3b; it involves only three days, including however one of $449\frac{1}{2}$ minutes. Computational experience revealed that the heuristic procedure converged much faster with $\lambda = -\infty$ than with $\lambda = 0$. More details about this application can be found in [17].

For P2, a variation on Lin's method was used, whereby only a limited number of promising potential improvements was checked. The number of trucks needed was reduced from ten (Figure 4a) to eight (Figures 4b,c,d). In view of the size of the problem, both possibilities $\lambda = 0$ and $\lambda = -\infty$ have been run only once; the convergence with $\lambda = -\infty$ was relatively slow.

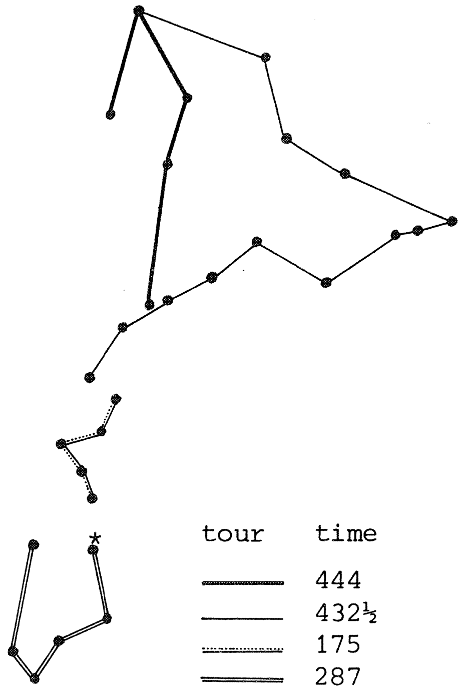


Figure 3a
P1: 3-optimal solution;
$\lambda = -\infty$;
$B(4) = 1338\frac{1}{2}$.

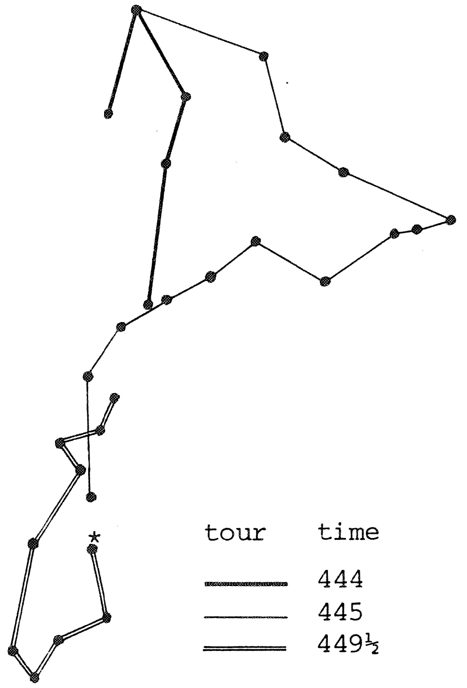


Figure 3b
P1: infeasible solution,
obtained by hand from Figure 3a;
$B(3) = 1338\frac{1}{2}$.

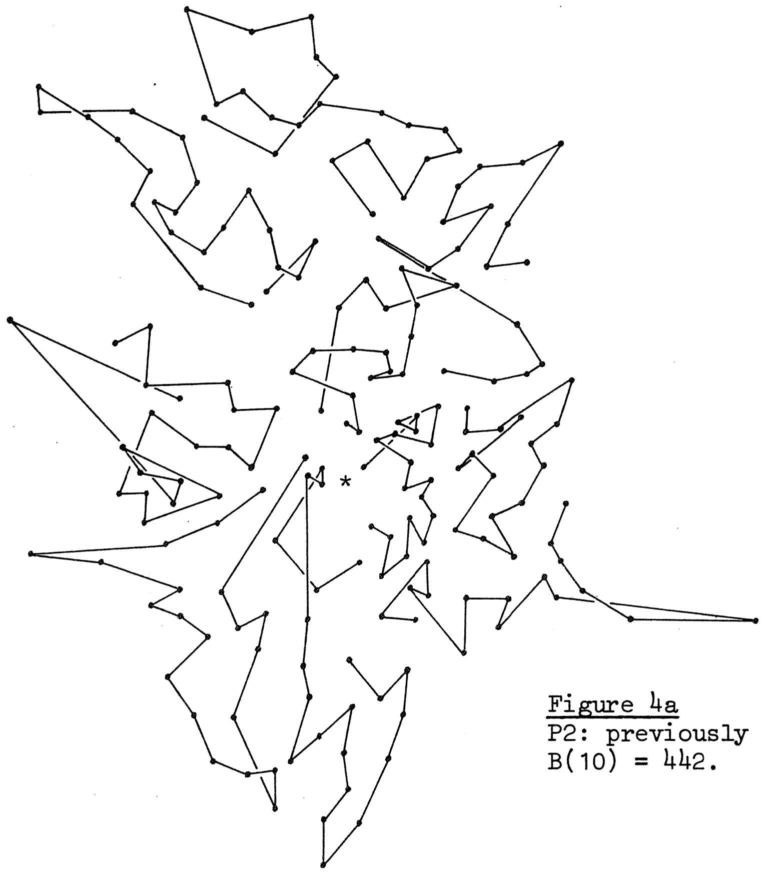


Figure 4a
P2: previously used solution;
B(10) = 442.

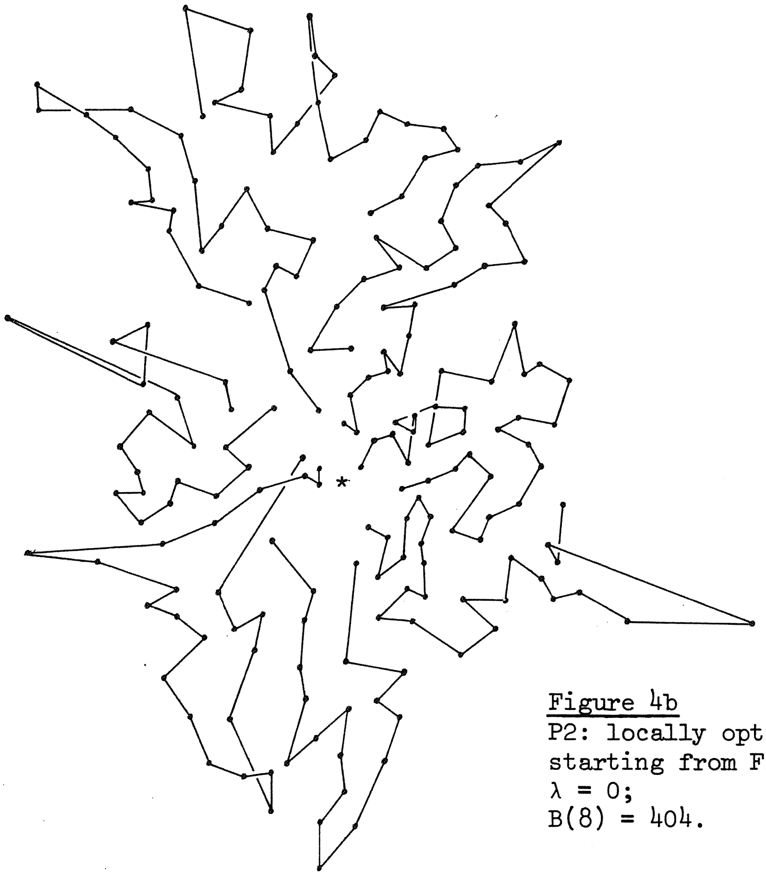


Figure 4b
P2: locally optimal solution,
starting from Figure 4a;
λ = 0;
B(8) = 404.

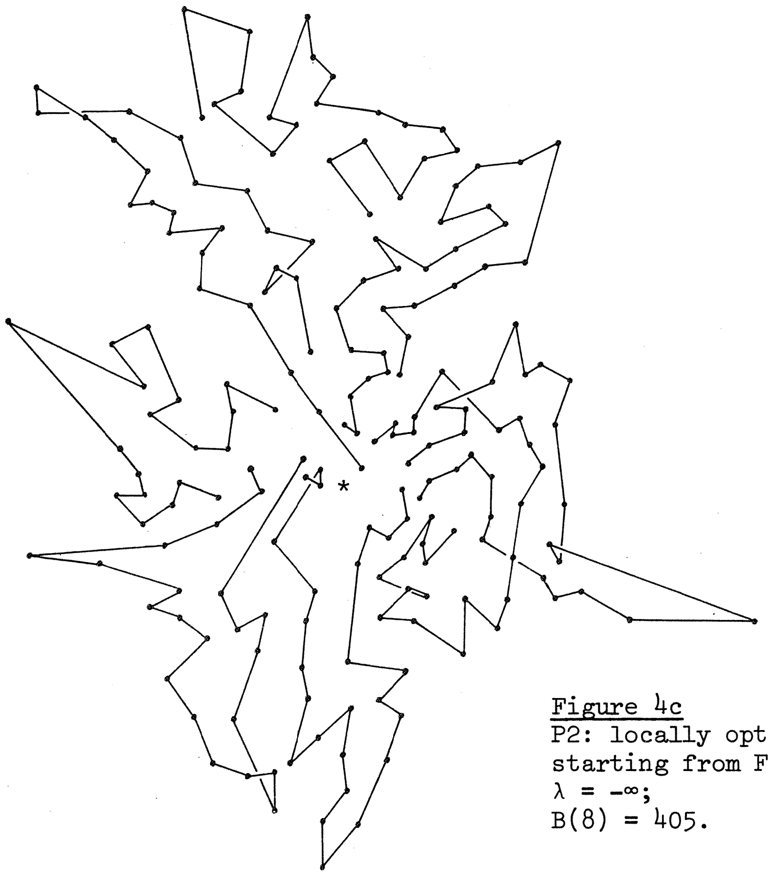


Figure 4c
P2: locally optimal solution,
starting from Figure 4a;
$\lambda = -\infty$;
B(8) = 405.

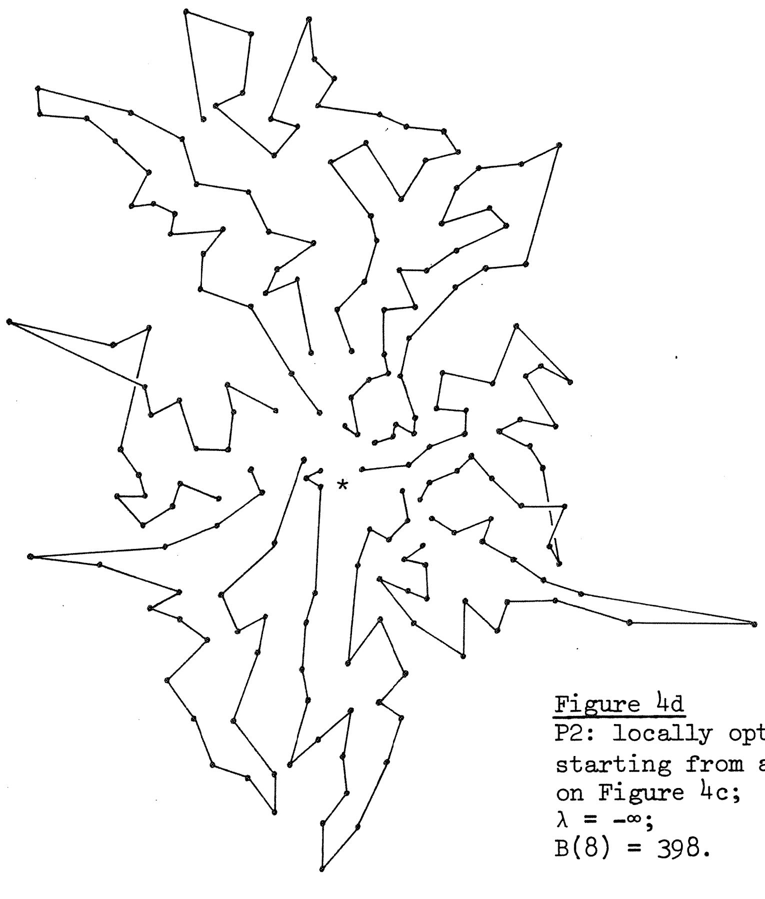


Figure 4d
P2: locally optimal solution,
starting from an improvement by hand
on Figure 4c;
$\lambda = -\infty$;
B(8) = 398.

4. CLUSTERING A DATA ARRAY

4.1. Problem description

Suppose that a *data array* (a_{ij}) $(i \in R,\ j \in S)$ is given, where a_{ij} measures the strength of the relationship between elements $i \in R$ and $j \in S$. A *clustering* of the array is obtained by permuting its rows and columns and should identify subsets of R that are strongly related to subsets of S.

This situation occurs in widely different contexts. Here we will apply a clustering technique to three examples. In the first one [24] R is a collection of 24 *marketing* techniques, S is a collection of 17 marketing applications, $a_{ij} = 1$ if technique i has been successfully used for application j, and $a_{ij} = 0$ otherwise. The second example [24] arises in *airport design*; R (= S) is a set of 27 control variables and a_{ij} measures their interdependence. The third example [33] deals with an *import-export matrix*; R (= S) is a set of 50 regions on the Indonesian islands, $a_{ij} = 1$ if in 1971 a quantity of at least 50 tons of rice was transported from region i to region j, and $a_{ij} = 0$ otherwise.

These three examples indicate that the approach is useful for *problem decomposition* and *data reorganization*. A more elaborate discussion of its applicability and more examples can be found in [24].

To convert this problem into an optimization problem, some criterion has to be defined. In [24] the proposed *measure of effectiveness* (ME) is the sum of all products of horizontally or vertically adjacent elements in the array. Figure 5 (adapted from [24]) shows how this criterion relates to various permutations of a 4×4 array. The problem is to find permutations of rows and columns of (a_{ij}) maximizing ME.

$$
\begin{array}{c|cccc|}
 & 1 & 2 & 3 & 4 \\ \hline
1 & 1 & 0 & 1 & 0 \\
2 & 0 & 1 & 0 & 1 \\
3 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 0 & 1 \\ \hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & 1 & 2 & 3 & 4 \\ \hline
1 & 1 & 0 & 1 & 0 \\
2 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 0 & 1 \\
3 & 1 & 0 & 1 & 0 \\ \hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & 1 & 2 & 3 & 4 \\ \hline
1 & 1 & 0 & 1 & 0 \\
3 & 1 & 0 & 1 & 0 \\
2 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 0 & 1 \\ \hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & 1 & 3 & 2 & 4 \\ \hline
1 & 1 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 1 & 1 \\
3 & 1 & 1 & 0 & 0 \\ \hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & 1 & 3 & 2 & 4 \\ \hline
1 & 1 & 1 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 1 & 1 \\ \hline
\end{array}
$$

$$
\mathrm{ME} = 0 \qquad\qquad \mathrm{ME} = 2 \qquad\qquad \mathrm{ME} = 4 \qquad\qquad \mathrm{ME} = 6 \qquad\qquad \mathrm{ME} = 8
$$

Figure 5 ME for various permutations of a 4×4 array.

4.2. TSP formulation

Let $R = \{1,\dots,r\}$ and $S = \{1,\dots,s\}$. With the conventions

$$\rho(0) = \rho(r+1) = \sigma(0) = \sigma(s+1) = *,$$
$$a_{i*} = a_{*j} = 0 \quad \text{for } i \in R,\ j \in S,$$

the ME, corresponding to permutations ρ of R and σ of S, is given by

$$\begin{aligned} ME(\rho,\sigma) &= \\ &= \tfrac{1}{2} \textstyle\sum_{i\in R} \sum_{j\in S} a_{\rho(i)\sigma(j)} \left(a_{\rho(i)\sigma(j-1)}+a_{\rho(i)\sigma(j+1)}+a_{\rho(i-1)\sigma(j)}+a_{\rho(i+1)\sigma(j)}\right) = \\ &= \textstyle\sum_{j=0}^{j=s} \sum_{i\in R} a_{i\sigma(j)}\ a_{i\sigma(j+1)} + \sum_{i=0}^{i=r} \sum_{j\in S} a_{\rho(i)j}\ a_{\rho(i+1)j} = \\ &= ME(\sigma) + ME(\rho), \end{aligned}$$

so $ME(\rho,\sigma)$ decomposes into two parts, and its maximization reduces to two separate and similar optimizations, one of $ME(\sigma)$ for the columns and the other of $ME(\rho)$ for the rows. It is stated in [24] that both subproblems may be rewritten as *quadratic assignment problems*. More precisely, they are symmetric TSPs:

$$TSP^{col}: \quad N^{col} = S \cup \{*\}, \quad c^{col}_{jk} = -\textstyle\sum_{i\in R} a_{ij}\ a_{ik} \quad \text{for } j,k \in N^{col},$$
$$TSP^{row}: \quad N^{row} = R \cup \{*\}, \quad c^{row}_{hi} = -\textstyle\sum_{j\in S} a_{hj}\ a_{ij} \quad \text{for } h,i \in N^{row},$$

for $ME(\sigma)$ and $ME(\rho)$, respectively (cf. [19]). In general, the clustering problem for a p-dimensional array can be stated as p TSPs. It may be attacked by any algorithm for the TSP; in fact, the *bond energy algorithm* (BEA), proposed in [24], is a simple suboptimal TSP method which constructs a tour by successively inserting the cities (cf. [25,p.76]).

If the data array is symmetric (i.e. $a_{ij} = a_{ji}$ for all i,j), then TSP^{row} and TSP^{col} are identical and only one optimization needs to be performed (see the airport example).

If the data array is square (i.e. $r = s$) but not necessarily symmetric and we want to have equal permutations of rows and columns (i.e. $\rho = \sigma$), then one symmetric TSP results:

$$TSP^{cow}: \quad N^{cow} = N^{col} = N^{row}, \quad c^{cow}_{ij} = c^{col}_{ij} + c^{row}_{ij} \quad \text{for } i,j \in N^{cow}$$

(see the import-export example).

The size of the TSPs might be reduced by assigning identical rows or columns to one single city under the assumption that these rows or columns will be adjacent in at least one optimal solution. This assumption is justified under the conditions expressed by the following theorem.

THEOREM. *If* $a_{ij} \in \{0,1\}$ *for all* $i \in R$, $j \in S$, *and* $c^{\text{row}}_{kk} = c^{\text{row}}_{k\ell} = c^{\text{row}}_{\ell\ell}$ *for some* $k,\ell \in N^{\text{row}}$, *then row* k *and row* ℓ *are identical, and adjacent in at least one optimal solution to* TSP^{row}.

Proof. We define $S_i = \{j | j \in S,\ a_{ij} = 1\}$ for all $i \in N^{\text{row}}$. Since $a_{ij} \in \{0,1\}$ for all $i \in R$, $j \in S$, we have

(1) $$c^{\text{row}}_{ij} = -|S_i \cap S_j| \quad \text{for all } i,j \in N^{\text{row}},$$

and $c^{\text{row}}_{kk} = c^{\text{row}}_{k\ell} = c^{\text{row}}_{\ell\ell}$ implies that $S_k = S_k \cap S_\ell = S_\ell$. Hence row k and row ℓ are identical:

(2) $$a_{kj} = a_{\ell j} \quad \text{for all } j \in S.$$

Now consider any permutation ρ of R with $\rho(p) = k$, $\rho(q) = \ell$, $|p - q| > 1$. Insert ℓ between k and $\rho(p+1)$. This will not decrease $\text{ME}(\rho)$ if

$$c^{\text{row}}_{k\rho(p+1)} + c^{\text{row}}_{\rho(q-1)\ell} + c^{\text{row}}_{\ell\rho(q+1)} \geq c^{\text{row}}_{k\ell} + c^{\text{row}}_{\ell\rho(p+1)} + c^{\text{row}}_{\rho(q-1)\rho(q+1)}.$$

By (1) and (2), this is equivalent to

$$|S_{\rho(q-1)} \cap S_\ell| + |S_\ell \cap S_{\rho(q+1)}| \leq |S_\ell| + |S_{\rho(q-1)} \cap S_{\rho(q+1)}|,$$

which is true, since

$$\begin{aligned} &|S_{\rho(q-1)} \cap S_\ell| + |S_\ell \cap S_{\rho(q+1)}| = \\ &= |S_\ell \cap (S_{\rho(q-1)} \cup S_{\rho(q+1)})| + |S_\ell \cap S_{\rho(q-1)} \cap S_{\rho(q+1)}| \leq \\ &\leq |S_\ell| + |S_{\rho(q-1)} \cap S_{\rho(q+1)}|. \end{aligned}$$ (Q.E.D.)

Analogous theorems hold for TSP^{col} and TSP^{cow}. Defining $R_j = \{i | i \in R,\ a_{ij} = 1\}$ for all $j \in N^{\text{col}}$, we have in the latter case

(3) $$c^{\text{cow}}_{ij} = -|S_i \cap S_j| - |R_i \cap R_j| \quad \text{for all } i,j \in N^{\text{cow}},$$

and we have to show that

(4) $$\begin{aligned} a_{kj} &= a_{\ell j} \quad \text{for all } j \in S, \\ a_{ik} &= a_{i\ell} \quad \text{for all } i \in R. \end{aligned}$$

It follows from (3) and $c^{cow}_{kk} = c^{cow}_{k\ell} = c^{cow}_{\ell\ell}$ that $|S_k| + |R_k| = |S_k \cap S_\ell| + |R_k \cap R_\ell| = |S_\ell| + |R_\ell|$. If $|S_k| > |S_k \cap S_\ell|$, then $|R_k| < |R_k \cap R_\ell|$, which is impossible; hence $|S_k| = |S_k \cap S_\ell| = |S_\ell|$ and $|R_k| = |R_k \cap R_\ell| = |R_\ell|$, which trivially leads to (4).

These results cannot be generalized to cover the case where a_{ij} can take on other values than 0 or 1. For example, if $R = \{1,2,3\}$ and $a_{1j} = a_{2j} = 1$, $a_{3j} = 2$ for $j \in S$, then the identical rows 1 and 2 are separated by row 3 in the optimal solution.

4.3. TSP equivalence

Not only can the clustering problem be formulated as one or more symmetric TSPs, but the symmetric TSP can be formulated as a clustering problem as well. Any method for maximizing the ME of a data array could therefore be used to solve the TSP. A polynomial-bounded clustering algorithm would lead to efficient algorithms for a number of notorious combinatorial problems and its existence seems highly unlikely (cf. [14]).

Analogous equivalence statements on computer wiring or vehicle routing problems and the TSP are easily proved. For the clustering problem, the proof is as follows.

The symmetric TSP corresponds to finding a minimum hamiltonian circuit in the complete undirected graph G with a vertex set $N = \{1,\dots,n\}$, an edge set $E = \{(i,j) | i,j \in N,\ i < j\}$ and a weight c_{ij} for each edge $(i,j) \in E$. This problem is equivalent to finding a minimum hamiltonian path in the graph G' with $N' = \{0\} \cup N$, $E' = \{(0,j) | j \in N\} \cup E$ and weights c'_{ij}, defined as follows:

$$\begin{aligned} c'_{01} &= 2\lambda, \\ c'_{0j} = c'_{1j} &= c_{1j} + \lambda \quad \text{for } 2 \le j \le n, \\ c'_{ij} &= c_{ij} \quad \text{for } 2 \le i < j \le n, \end{aligned}$$

where λ is greater than the length of any tour. Such a path will have vertices 0 and 1 as extreme points and these vertices can then be joined to arrive at the optimal tour. We now define a clustering problem with $R = N'$, $S = E'$ and

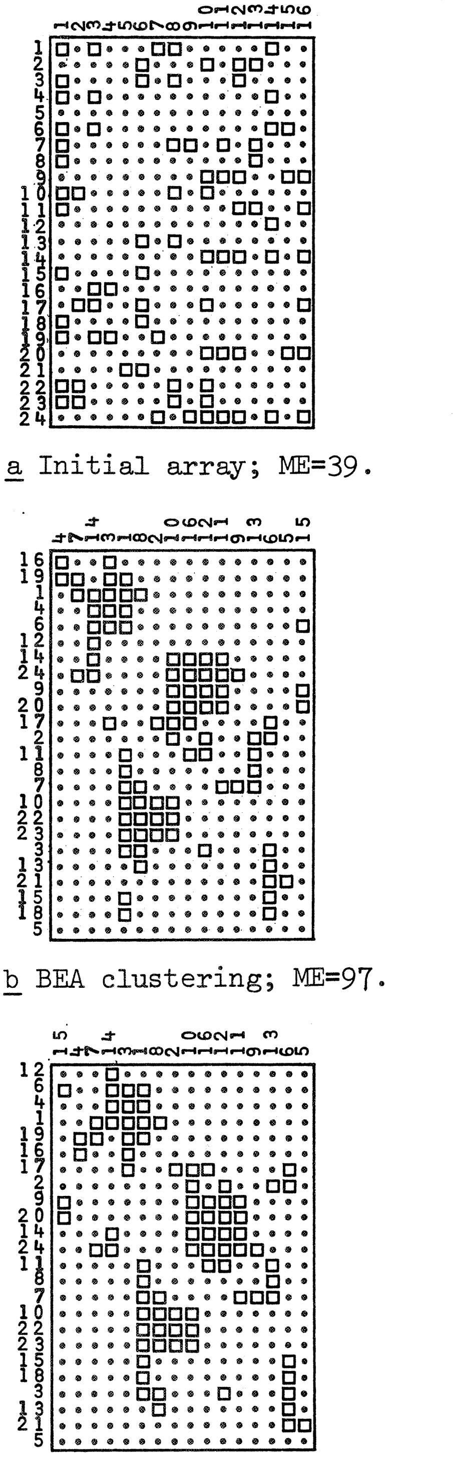
TABLE I. MARKETING EXAMPLE

Marketing techniques

1. Regression & correlation analysis
2. Discounted cash flow
3. Incremental analysis
4. Multiple regression/correlation
5. Random sampling
6. Sampling theory
7. Bayesian approach
8. Cost-benefit analysis
9. Critical path method
10. Decision trees
11. Dynamic programming
12. Exponential smoothing
13. Industrial dynamics
14. Input-output analysis
15. Linear programming
16. Markov processes
17. Monte Carlo simulation
18. Nonlinear programming
19. Numerical taxonomy
20. PERT
21. Queueing models
22. Risk analysis
23. Sensitivity analysis
24. Technological forecasting

Marketing applications

1. Advertising research
2. Acquisition screening
3. Brand strategy
4. Customer segmentation
5. Customer service
6. Distribution planning
7. Market segmentation
8. Pricing strategy
9. Product life-cycle analysis
10. Product line analysis
11. Product planning
12. R&D planning
13. ROI analysis
14. Sales forecasting
15. Test marketing
16. Venture planning



a Initial array; ME=39.

b BEA clustering; ME=97.

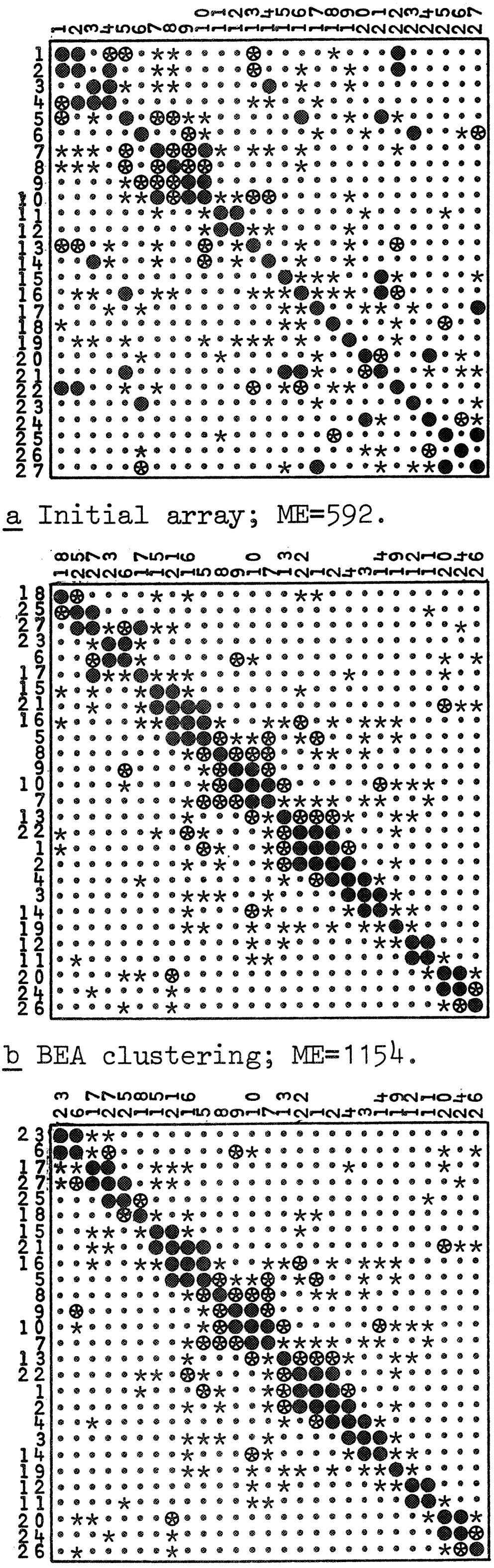
c Optimal clustering; ME=97.

Figure 6 Marketing example; • = 0, □ = 1.

TABLE II. AIRPORT EXAMPLE

Control variables

1. Passenger check-in
2. Baggage check-in
3. Baggage claim
4. Baggage moving system
5. Intra-airport transportation system
6. Cargo terminal
7. Close-in parking lots
8. Remote parking lots
9. Main access roads to and from airport
10. Circulation roads within airport
11. Service area for rental cars
12. Parking lots for rental cars
13. Curb space for unloading
14. Curb space for loading
15. Waiting areas at gates
16. Stations for intra-airport transportation system
17. Aircraft loading system
18. Concessions
19. Rental car desk
20. Runway capacity
21. Number of gates
22. Passenger information
23. Cargo transfer
24. Air-traffic-control system
25. Refuse removal
26. Flight operations and crew facilities
27. Aircraft service on the apron

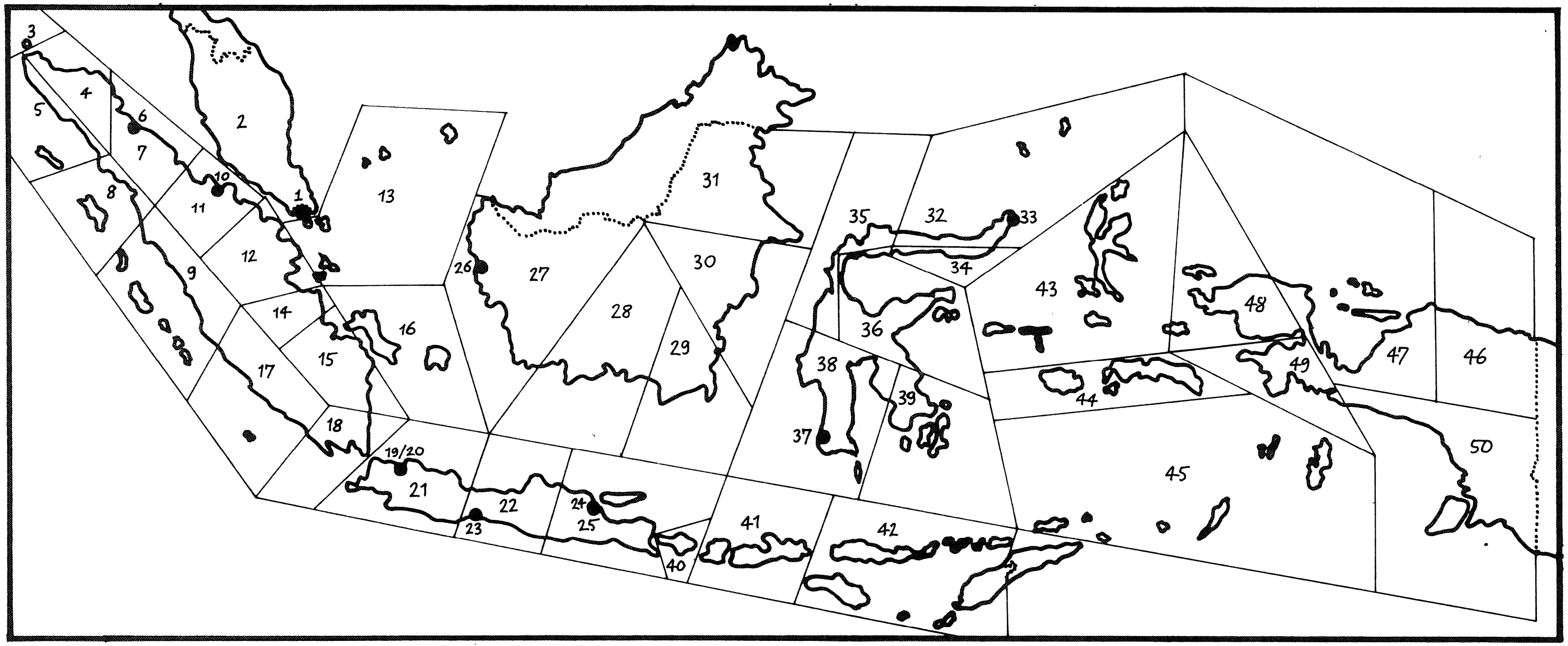


a Initial array; ME=592.

b BEA clustering; ME=1154.

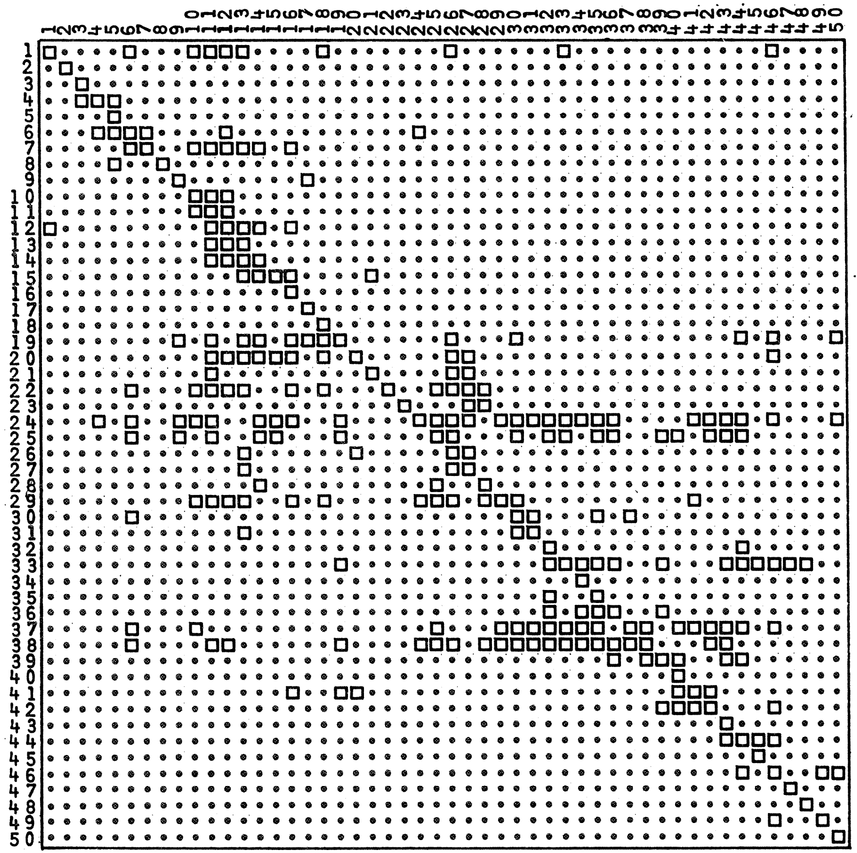
c Optimal clustering; ME=1160.

Figure 7 Airport example;
• = 0, ⋆ = 1, ⊛ = 2, ● = 3.

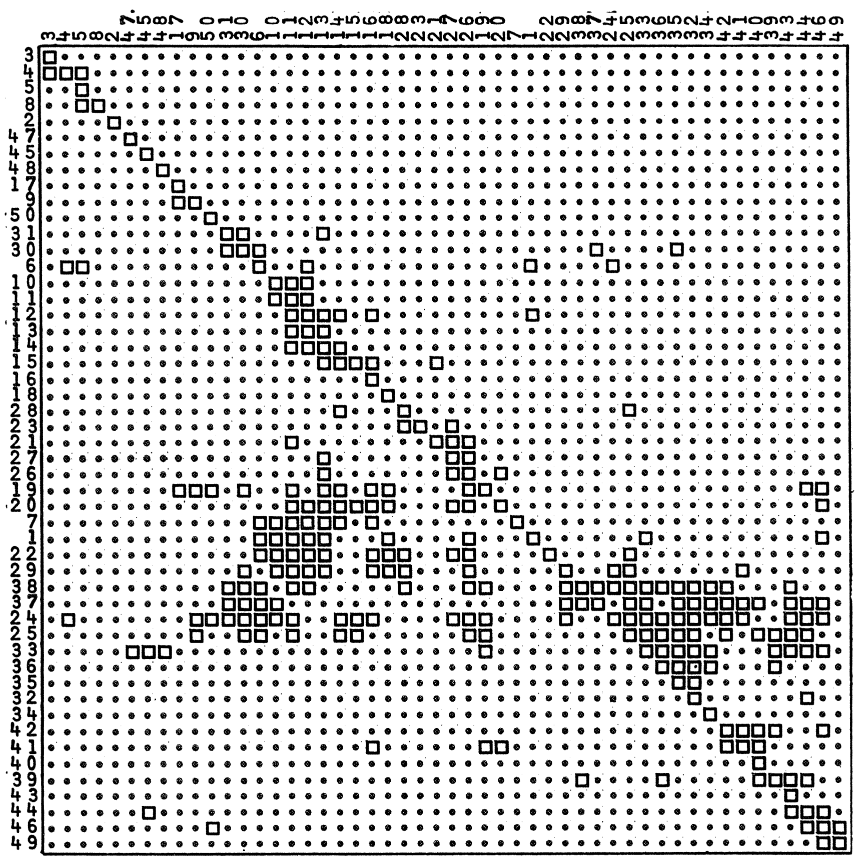


1. Singapore
2. Malay
3. Sabang
4. Aceh I
5. Aceh II
6. Belawan
7. Sumut I
8. Sumut II
9. Sumbar
10. Dumai
11. Ridar I
12. Ridar II
13. Rikep
14. Jambi
15. Sumsel I
16. Sumsel II
17. Bengkulu
18. Lampung
19. Jaya I
20. Jaya II
21. Jabar
22. Jateng I
23. Jateng II
24. Surabaya
25. Jatim
26. Pontianak
27. Kalbar
28. Kalteng
29. Kalsel
30. Kaltim I
31. Kaltim II
32. Sulut I
33. Bitung
34. Sulut II
35. Suteng I
36. Suteng II
37. Makasar
38. Sulsel
39. Sulteng
40. Bali
41. Nusa Tenguara Barat
42. Nusa Tenguara Timur
43. Malut
44. Malteng
45. Malsel
46. Irbaut I
47. Irbaut II
48. Irbaut III
49. Irbasel I
50. Irbasel II

Figure 8 Import-export example: regions on the Indonesian islands.



a Initial array; ME=223.



b 3-Optimal clustering subject to ρ=σ; ME=290.

Figure 9 Import-export example; • = 0, □ = 1.

$$a_{i(i,\ell)} = -c'_{i\ell} \quad \text{for } i \in R,\ (i,\ell) \in S;$$
$$a_{i(k,i)} = 1 \quad \text{for } i \in R,\ (k,i) \in S;$$
$$a_{i(k,\ell)} = 0 \quad \text{for } i \in R,\ (k,\ell) \in S,\ k,\ell \neq i.$$

The contribution of the adjacency of rows i and j with, say, $i < j$ to the ME is equal to

$$\textstyle\sum_{(k,\ell)\in S} a_{i(k,\ell)}\, a_{j(k,\ell)} = a_{i(i,j)}\, a_{j(i,j)} = -c'_{ij},$$

and it follows that any permutation ρ of R maximizing ME(ρ) minimizes the weight of the hamiltonian path $(\rho(0),\rho(1),\ldots,\rho(n))$ in G'

We can even show that the symmetric TSP with integer distances is equivalent to a clustering problem with $a_{ij} \in \{0,1\}$ for all $i \in R$, $j \in S$, by setting $c'_{ij} := c'_{ij} - 3\lambda$ for all (i,j) and expanding column (i,j) into $-c'_{ij}$ columns, each containing two ones and $n - 1$ zeros.

4.4. Results

The techniques and applications pertaining to the marketing example are given in Table I. Figure 6 shows the initial data array, the clustering produced by the BEA as reported in [24], and a clustering corresponding to optimal solutions of TSP^{col} and TSP^{row}, found by Little's algorithm after application of the theorem on row identification. It turns out that the BEA clustering is optimal.

The control variables in the airport example are given in Table II. Figure 7 shows the symmetric initial data array, the BEA clustering [24], and a clustering corresponding to an optimal solution of TSP^{col} (= TSP^{row}), found by Held and Karp's method. The BEA clustering is not optimal, and, in fact, not even 3-optimal, since it can be improved by exchanging three links.

The geographical distribution of the regions on the Indonesian islands in the import-export example is given in Figure 8. Figure 9 shows the square but asymmetric initial data array and a clustering corresponding to a 3-optimal solution of TSP^{cow}, found by Lin's heuristic.

5. JOB-SHOP SCHEDULING WITH NO INTERMEDIATE STORAGE

5.1. Problem description

One of the basic assumptions in most existing theory on machine scheduling is that a job is allowed to wait arbitrarily long before being processed on its next machine [32]. This assumption is highly unrealistic in some real world situations where intermediate storage space is finite or may even be non-existing. The former situation exists for instance in a computer system where buffer space is limited and costly; the latter situation is met in steel or aluminium rolling where the very high temperature of the metal has to be maintained throughout the production process.

Several researchers [27;29;37;20;7;30;8;34] have studied the problem of minimizing the total processing time under the restriction of no intermediate storage in a flow-shop, where the machine order of each job is identical; see [35] for a different criterion. These assumptions imply that the processing order on each machine will be identical, which simplifies the analysis.

In [31] a more general production process is considered, but the resulting definitions and theorems are not very clear and the proposed algorithm seems highly inefficient. In fact, extensions both to non-zero but finite intermediate storage and to different processing orders per machine seem to complicate the situation considerably. We shall restrict our attention to a job-shop where

(a) the machine order may vary per job;

(b) each job visits each machine at least once;

(c) no passing is permitted, i.e. the processing order is identical on all machines;

(d) no intermediate storage is allowed.

5.2. TSP formulation

The job-shop scheduling problem can be described as follows.

- n *jobs* J_i $(1 \le i \le n)$ have to be processed on m *machines* M_ℓ $(1 \le \ell \le m)$;

- job J_i $(1 \le i \le n)$ consists of m_i *operations* O_{ik} $(1 \le k \le m_i)$;
- the machine order of J_i $(1 \le i \le n)$ is given by $\mu_i = (\mu_i(1),\ldots,\mu_i(m_i))$, i.e. the k-th operation O_{ik} of J_i has to be performed on $M_{\mu_i(k)}$;
- the processing time of O_{ik} $(1 \le i \le n,\ 1 \le k \le m_i)$ is given by p_{ik};
- the total processing time has to be minimized under the conditions, mentioned in section 5.1.

We define

$$P_i[k_1,k_2] = \textstyle\sum_{k=k_1}^{k=k_2} p_{ik} \qquad \text{for } 1 \le i \le n,\ 1 \le k_1 \le k_2 \le m_i;$$

$$\left.\begin{array}{l} k_i'(\ell) = \min\{k \mid \mu_i(k) = \ell,\ 1 \le k \le m_i\} \\ k_i''(\ell) = \max\{k \mid \mu_i(k) = \ell,\ 1 \le k \le m_i\} \end{array}\right\} \text{ for } 1 \le i \le n,\ 1 \le \ell \le m.$$

$O_{ik'_i(\ell)}$ and $O_{ik''_i(\ell)}$ are the first and last operations of J_i on M_ℓ; their existence is ensured by condition (b).

For each pair of jobs (J_i,J_j), we will calculate a coefficient c_{ij}, representing the minimum difference between the starting times of O_{i1} and O_{j1} if J_j is scheduled directly after J_i. By condition (c), $O_{ik''_i(\ell)}$ has to precede $O_{jk'_j(\ell)}$ on M_ℓ, for $1 \le \ell \le m$. We introduce a directed graph G_{ij} with vertex set N_{ij} and arc set A_{ij}, defined by

$$N_{ij} = \{O_{hk} \mid h = i,j,\ 1 \le k \le m_h\};$$

$$A_{ij} = \{(O_{hk},O_{h\,k+1}) \mid h=i,j,\ 1 \le k \le m_h-1\} \cup \{(O_{ik''_i(\ell)},O_{ik'_j(\ell)}) \mid 1 \le \ell \le m\};$$

a weight p_{hk} is attached to each vertex $O_{hk} \in N_{ij}$. For an example with m = 3, $\mu_i = (2,1,2,3,2)$ and $\mu_j = (1,2,3,1)$, the graph G_{ij} is given in Figure 10.

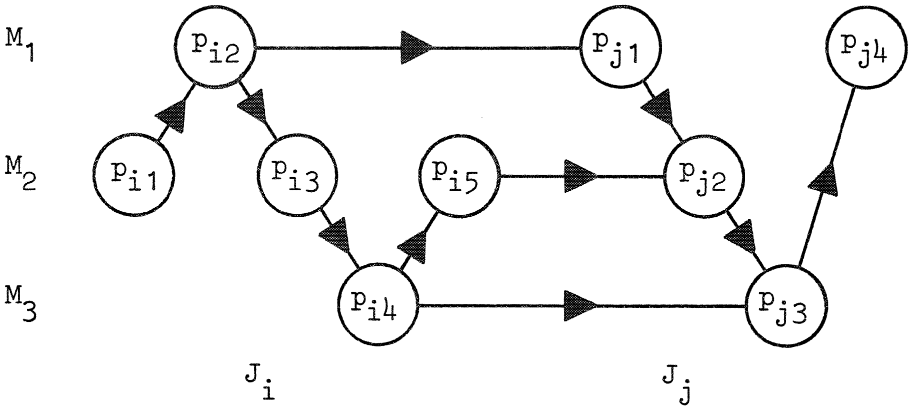


Figure 10 An example of the graph G_{ij}.

As to the *path of maximum weight* (also called *longest* or *critical path*) in G_{ij}, it is clear that

(5) it starts from O_{i1} and ends in O_{jm_j};

(6) it contains exactly one arc $(O_{ik''_i(\ell)}, O_{jk'_j(\ell)})$.

Condition (d) implies that c_{ij} is equal to the latest possible starting time of O_{j1} in G_{ij} if O_{i1} starts at time zero and O_{jm_j} finishes as early as possible. It follows from (5) and (6) that

$$(7) \qquad c_{ij} = \max_\ell \big(P_i[1,k''_i(\ell)] + P_j[k'_j(\ell),m_j]\big) - P_j[1,m_j] = \\ = \max_\ell \big(P_i[1,k''_i(\ell)] - P_j[1,k'_j(\ell)-1]\big).$$

The minimum total processing time is now given by

$$(8) \qquad \min_\nu \big(\textstyle\sum_{i=1}^{i=n-1} c_{\nu(i)\nu(i+1)} + P_{\nu(n)}[1,m_{\nu(n)}]\big)$$

where ν runs over all permutations of $\{1,\ldots,n\}$; $\nu(i)$ is the i-th job in a processing schedule.

We add a job J_* with $m_* = m$, $\mu_*(k) = k$ and $p_{*k} = 0$ for $1 \leq k \leq m$, representing beginning and end of a schedule. According to (7), its coefficients are given by $c_{*i} = 0$, $c_{i*} = P_i[1,m_i]$ for $1 \leq i \leq n$. Determination of (8) now corresponds to solving a TSP with $N = \{*\} \cup \{1,\ldots,n\}$ and (c_{ij}) defined by (7).

This TSP is asymmetric and euclidean. To prove the latter assertion we have to show that $c_{ij} + c_{jk} \geq c_{ik}$ for any $i,j,k \in N$, or, equivalently, that

$$\max_\ell \big(P_i[1,k''_i(\ell)] + P_j[k'_j(\ell),m_j]\big) + \max_\ell \big(P_j[1,k''_j(\ell)] + P_k[k'_k(\ell),m_k]\big) \geq \\ \geq \max_\ell \big(P_i[1,k''_i(\ell)] + P_k[k'_k(\ell),m_k]\big) + P_j[1,m_j].$$

This is true, since for any $\ell \in \{1,\ldots,m\}$

$$\big(P_i[1,k''_i(\ell)] + P_j[k'_j(\ell),m_j]\big) + \big(P_j[1,k''_j(\ell)] + P_k[k'_k(\ell),m_k]\big) \geq \\ \geq \big(P_i[1,k''_i(\ell)] + P_k[k'_k(\ell),m_k]\big) + P_j[1,m_j].$$

We make two final remarks on this TSP formulation.

Remark 1. In a flow-shop we know that $\mu_i = (1,2,\ldots,m)$ for $1 \leq i \leq m$, and (7) simplifies to $c_{ij} = \max_\ell \big(P_i[1,\ell] - P_j[1,\ell-1]\big)$, which corresponds to the results given in [27;29;8].

Remark 2. So far, distances have been defined as differences between the starting times of the first operations of jobs. More generally, one might arbitrarily select any two operations $O_{ik_i^*}$ and $O_{ik_i^{**}}$ for each job J_i and define c_{ij} as the minimum difference between the starting times of $O_{ik_i^*}$ and $O_{jk_j^{**}}$ if J_i precedes J_j directly. This will lead to modifications in (7) and (8), but to an equivalent TSP (cf. [7;30]).

5.3. TSP equivalence

We will now show that any TSP can be formulated as a job-shop scheduling problem with no intermediate storage. This will establish the equivalence of these problems (cf. [14]).

First, the TSP with $N = \{1,\ldots,n\}$ and (c_{ij}) $(i,j \in N)$ is converted into a minimum hamiltonian path problem on the complete directed graph G' with vertex set $N' = \{0\} \cup N$ and weights c'_{ij} on the arcs, defined by

$$\begin{aligned}
&c'_{01} = c'_{10} = \lambda, \\
&c'_{0i} = c'_{i1} = \lambda && \text{for } 2 \le i \le n, \\
&c'_{i0} \quad\;\; = c_{i1} && \text{for } 2 \le i \le n, \\
&c'_{1i} \quad\;\; = c_{1i} && \text{for } 2 \le i \le n, \\
&c'_{ij} \quad\;\; = c_{ij} && \text{for } 2 \le i,j \le n,
\end{aligned}$$

where λ is appropriately large (see below) and all c_{ij} may be assumed to be positive. A minimum hamiltonian path will have vertex 1 in the first and vertex 0 in the last position.

It is convenient to be able to assume that no two coefficients appearing in the same row or column are equal. Hence we add $i\varepsilon$ to row i and $j\varepsilon$ to column j of (c'_{ij}), where

$$0 < \varepsilon \le \frac{1}{n+1} \min\{|c'_{ij}-c'_{k\ell}| \,|\, c'_{ij} \ne c'_{k\ell},\ i = k \text{ or } j = \ell,\ i,j,k,\ell \in N\}.$$

This leads to an equivalent problem with weights $c''_{ij} = c'_{ij} + (i+j)\varepsilon$. If $c'_{ik} = c'_{i\ell}$, then $|c''_{ik}-c''_{i\ell}| = |k-\ell|\varepsilon > 0$; if $c'_{ik} \ne c'_{i\ell}$, then $|c''_{ik}-c''_{i\ell}| = |c'_{ik}-c'_{i\ell}+(k-\ell)\varepsilon| \ge |c'_{ik}-c'_{i\ell}| - |k-\ell|\varepsilon > 0$. Hence no row, and, similarly, no column of (c''_{ij}) contains two equal numbers. For each $i \in N'$, there exist two unique permutations $\alpha_i = (\alpha_i(1),\ldots,\alpha_i(n))$ and $\beta_i = (\beta_i(1),\ldots,\beta_i(n))$ of $N' - \{i\}$ such that

$$c''_{i\alpha_i(1)} < c''_{i\alpha_i(2)} < \dots < c''_{i\alpha_i(n)};$$
$$c''_{\beta_i(1)i} > c''_{\beta_i(2)i} > \dots > c''_{\beta_i(n)i}.$$

Now consider the following job-shop scheduling problem.

- $n+1$ jobs J_i ($i \in N'$) have to be processed on $n(n+1)$ machines M_{ij} ($i,j \in N'$, $i \neq j$);
- job J_i ($i \in N'$) consists of $m = n(n+3)$ operations O_{ik} ($1 \le k \le m$);
- the machine order of J_i ($i \in N'$) is given by
  $M_{\beta_i(1)i}, M_{\beta_i(2)i}, \dots, M_{\beta_i(n)i}$,
  $M_{01}, M_{02}, \dots, M_{0n}, M_{10}, M_{12}, \dots, M_{1n}, \dots\dots, M_{n0}, M_{n1}, \dots, M_{n\ n-1}$,
  $M_{i\alpha_i(1)}, M_{i\alpha_i(2)}, \dots, M_{i\alpha_i(n)}$;
- the processing times of the operations O_{ik} ($i \in N'$) are given by

$$\begin{array}{lll} p_{ik} & = c''_{\beta_i(k)i} - c''_{\beta_i(k+1)i} & \text{for } 1 \le k \le n-1; \\ p_{in} & = c''_{\beta_i(n)i} + \lambda; & \\ p_{ik} & = 1 & \text{for } n+1 \le k \le n(n+2); \\ p_{i\ n(n+2)+1} & = \lambda + c''_{i\alpha_i(1)}; & \\ p_{i\ n(n+2)+k} & = c''_{i\alpha_i(k)} - c''_{i\alpha_i(k-1)} & \text{for } 2 \le k \le n; \end{array}$$

- the total processing time has to be minimized under the conditions of no passing and no intermediate storage.

We will need the following equalities.

$$\begin{array}{lll} P_i[k,n] & = c''_{\beta_i(k)i} + \lambda & \text{for } i \in N', k \in N; \\ P_i[n+1,n(n+2)] & = n(n+1) & \text{for } i \in N'; \\ P_i[n(n+2)+1,n(n+2)+k] & = \lambda + c''_{i\alpha_i(k)} & \text{for } i \in N', k \in N; \end{array}$$

$$c''_{0\alpha_0(n)} = c''_{0n} = \lambda + n\varepsilon;$$
$$c''_{1\alpha_1(n)} = c''_{10} = \lambda + \varepsilon;$$
$$c''_{i\alpha_i(n)} = c''_{i1} = \lambda + (i+1)\varepsilon \quad \text{for } i \in N' - \{0,1\};$$

$$c''_{\beta_0(1)0} = c''_{10} = \lambda + \varepsilon;$$
$$c''_{\beta_1(1)1} = c''_{n1} = \lambda + (n+1)\varepsilon;$$
$$c''_{\beta_i(1)i} = c''_{0i} = \lambda + i\varepsilon \quad \text{for } i \in N' - \{0,1\}.$$

Analogously to section 5.2, we define

$$\left.\begin{aligned} k_i'(g,h) &= \min\{k \mid O_{ik} \text{ is processed on } M_{gh}\} \\ k_i''(g,h) &= \max\{k \mid O_{ik} \text{ is processed on } M_{gh}\} \end{aligned}\right\} \text{ for } i,g,h \in N',\ g \neq h,$$

so that

$$\begin{aligned} &k_i'(\beta_i(k),i) = k && \text{for } i \in N',\ k \in N; \\ &k_i''(i,\alpha_i(k)) = n(n+2) + k && \text{for } i \in N',\ k \in N; \\ &k_i'(g,h) \geq n+1 && \text{for } i,g,h \in N',\ i \neq h,\ g \neq h; \\ &k_i''(g,h) \leq n(n+2) && \text{for } i,g,h \in N',\ i \neq g,\ g \neq h. \end{aligned}$$

Approaching this job-shop problem in the way, described in section 5.2, we construct the weighted directed graph G_{ij} $(i,j \in N')$. We claim that there is a longest path in G_{ij} that contains the arc $(O_{ik''_i(i,j)}, O_{jk'_j(i,j)})$. To prove this, note that each path in G_{ij} from O_{i1} to O_{jm} contains exactly one arc $(O_{ik''_i(g,h)}, O_{jk'_j(g,h)})$ (see (5) and (6)); the length L_{gh} of such a path is equal to

$$L_{gh} = P_i[1,k_i''(g,h)] + P_j[k_j'(g,h),m].$$

We have to show that

(9) $$L_{ij} = \max_{g \neq h} L_{gh}.$$

First, we calculate L_{ij}. There exist two numbers $d,e \in N$ such that $j = \alpha_i(d)$, $i = \beta_j(e)$, and it follows that

(10) $$\begin{aligned} L_{ij} &= P_i[1,n(n+2)+d] + P_j[e,m] = \\ &= P_i[1,n(n+2)] + \lambda + c''_{i\alpha_i(d)} + c''_{\beta_j(e)j} + \lambda + P_j[n+1,m] = \\ &= P_i[1,n(n+2)] + 2\lambda + 2c''_{ij} + P_j[n+1,m]. \end{aligned}$$

If $g = i$, $h \neq j$, then we can find an $f \in N$ such that $h = \alpha_i(f)$, and we have

$$\begin{aligned} L_{ih} &\leq P_i[1,n(n+2)+f] + P_j[n+1,m] = \\ &= P_i[1,n(n+2)] + \lambda + c''_{i\alpha_i(f)} + P_j[n+1,m] \leq \\ &\leq L_{ij} - \lambda - 2c''_{ij} + c''_{i\alpha_i(n)} \leq L_{ij}, \end{aligned}$$

where the latter inequality is proved by

$$\lambda + 2c''_{0j} - c''_{0\alpha_0(n)} \geq \lambda + 2\lambda \qquad - (\lambda + n\varepsilon) \qquad > 0 \quad \text{if } i = 0;$$

$$\lambda + 2c''_{ij} - c''_{i\alpha_i(n)} \geq \lambda + 2(i+j)\varepsilon - (\lambda + (i+1)\varepsilon) \geq 0 \quad \text{if } i \neq 0.$$

If $g \neq i$, $h = j$, then we can show in a similar way that

$$L_{gj} \leq L_{ij}.$$

If $g \neq i$, $h \neq j$, then we have

$$L_{gh} < P_i[1,n(n+2)] + P_j[n+1,m] < L_{ij},$$

which completes the proof of (9).

Let d_{ij} $(i,j \in N')$ denote the minimum difference between the starting times of $O_{i\ n+1}$ and $O_{j\ n+1}$ if J_i precedes J_j directly (cf. *Remark 2* in section 5.2). It follows from (9) and (10) that

$$d_{ij} = L_{ij} - \big(P_i[1,n] + P_j[n+1,m]\big) = 2c''_{ij} + 2\lambda + n(n+1).$$

The total processing time $T(\nu)$ of a schedule $\nu = (\nu(0),\nu(1),\ldots,\nu(n))$ is equal to

$$T(\nu) = P_{\nu(0)}[1,n] + \textstyle\sum_{i=0}^{i=n-1} d_{\nu(i)\nu(i+1)} + n(n+1) + P_{\nu(n)}[n(n+2)+1,m].$$

We claim that $\nu(0) = 1$ and $\nu(n) = 0$ in any optimal schedule ν. If $\nu(0) = 1$ and $\nu(n) = 0$, then

$$(11)\ T(\nu) = \lambda + (n+1)\varepsilon + \lambda + 2\textstyle\sum_{i=0}^{i=n-1} c''_{\nu(i)\nu(i+1)} + 2n\lambda + n(n+1)^2 + \lambda + n\varepsilon + \lambda =$$
$$= (2n+4)\lambda + (2n+1)\varepsilon + n(n+1)^2 + 2\textstyle\sum_{i=0}^{i=n-1} c''_{\nu(i)\nu(i+1)} \leq$$
$$\leq (2n+5)\lambda + n(n+1)^2$$

if λ is sufficiently large. However, if $\nu(j) = 1$ with $j > 0$, then $c''_{\nu(j-1)\nu(j)} > \lambda$ and

$$T(\nu) > 2\lambda + 2\textstyle\sum_{i\neq j-1} c''_{\nu(i)\nu(i+1)} + 2\lambda + 2n\lambda + n(n+1)^2 + 2\lambda >$$
$$> (2n+6)\lambda + n(n+1)^2;$$

the same inequality holds if $\nu(j) = 0$ with $j < n$.

Thus we have proved that any permutation ν minimizing $T(\nu)$ has $\nu(0) = 1$ and $\nu(n) = 0$; by (11), it minimizes $\sum_{i=0}^{i=n-1} c''_{\nu(i)\nu(i+1)}$, i.e. the weight of the hamiltonian path $(\nu(0),\nu(1),\ldots,\nu(n))$ in G'.

5.4. Results

To illustrate the consequences of the no intermediate storage condition, we solved the three job-shop scheduling problems from [26,pp.236-237] under this restriction, using Little's TSP algorithm. In Table III the solution values are compared with the lengths of the schedules when infinite intermediate storage is allowed. Figure 11 illustrates the optimal schedules for one of these problems; the unrestricted schedule was found by a method of Florian et al. [6]. In general, the conditions of no intermediate storage and no passing can be expected to lead to large amounts of idle time on the machines.

TABLE III. EFFECT OF NO INTERMEDIATE STORAGE

number of jobs	number of machines	value without intermediate storage	value with intermediate storage
6	6	120	55
10	10	2433	972*
20	5	2132	1165

* indicates that the optimality has not been proved.

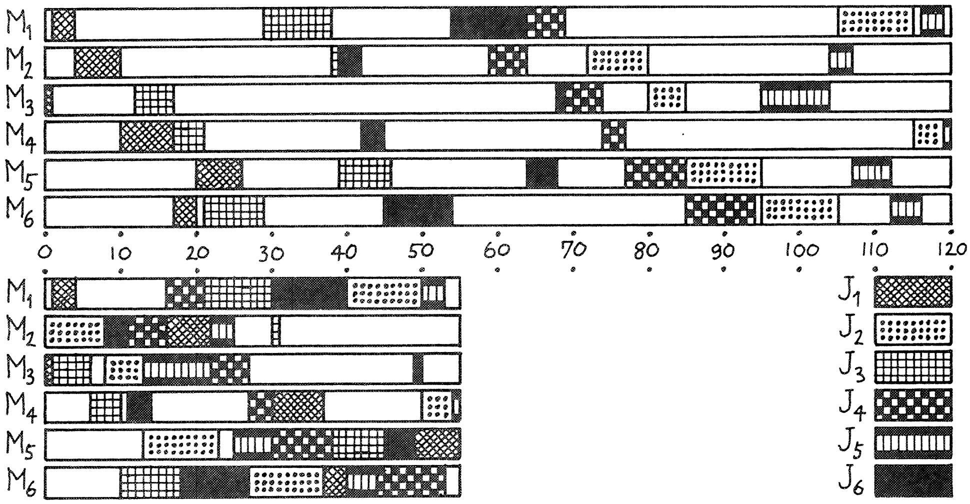


Figure 11 Optimal schedules for a 6×6 problem without and with intermediate storage.

ACKNOWLEDGEMENTS

We would like to thank B.J. Lageweg (Mathematisch Centrum) for useful comments and programming advice, J. Visschers and P. ten Kate (Instituut voor Kernphysisch Onderzoek) and J. Berendse and J.H. Kuiper (PTT) for implementing the applications described in sections 2 and 3 respectively, and A.W. Roes and T.J. Wansbeek (Instituut voor Verkeers- en Vervoers-economie) for making available the data of the import-export example, described in section 4.

REFERENCES

1. M. BELLMORE, J.C. MALONE, Pathology of Traveling-Salesman Subtour Elimination Algorithms, *Operations Res.* 19(1971)278-307,1766.
2. M. BELLMORE, G.L. NEMHAUSER, The Traveling Salesman Problem: a Survey, *Operations Res.* 16(1968)538-558.
3. N. CHRISTOFIDES, S. EILON, Algorithms for Large-Scale Travelling Salesman Problems, *Operational Res. Quart.* 23(1972)511-518.
4. E.W. DIJKSTRA, A Note on Two Problems in Connexion with Graphs, *Numer. Math.* 1(1959)269-271.
5. S. EILON, C.D.T. WATSON-GANDY, N. CHRISTOFIDES, *Distribution Management: Mathematical Modelling and Practical Analysis*, Griffin, London, 1971.
6. M. FLORIAN, P. TREPANT, G. McMAHON, An Implicit Enumeration Algorithm for the Machine Sequencing Problem, *Management Sci.* 17(1971)B782-792.
7. S.K. GOYAL, A Note on the Paper: On the Flow-Shop Sequencing Problem with No Wait in Process, *Operational Res. Quart.* 24(1973)130-133.
8. J. GRABOWSKI, M.M. SYSLO, On Some Machine Sequencing Problems (I), *Zastos. Mat.* 13(1973)339-345.
9. M. HANAN, J.M. KURTZBERG, A Review of the Placement and Quadratic Assignment Problems, *SIAM Rev.* 14(1972)324-342.
10. K. HELBIG HANSEN, J. KRARUP, Improvements of the Held-Karp Algorithm for the Symmetric Traveling-Salesman Problem, *Math. Programming* 7(1974)87-96.
11. M. HELD, R.M. KARP, The Traveling-Salesman Problem and Minimum Spanning Trees, *Operations Res.* 18(1970)1138-1162.
12. M. HELD, R.M. KARP, The Traveling-Salesman Problem and Minimum Spanning Trees: Part II, *Math. Programming* 1(1971)6-25.
13. A.M. ISAAC, E. TURBAN, Some Comments on the Traveling Salesman Problem, *Operations Res.* 17(1969)543-546.
14. R.M. KARP, Reducibility among Combinatorial Problems, pp.85-103 in R.E. MILLER, J.W. THATCHER (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972.
15. J. KRARUP, Private Communication, 1974.
16. J.B. KRUSKAL, On the Shortest Spanning Subtree of a Graph and the Traveling-Salesman Problem, *Proc. Amer. Math. Soc.* 2(1956)48-50.

17. J.H. KUIPER, "Hoe een PTT-er handelsreiziger werd" - een Routing Probleem, Unpublished Report, 1973.

18. J.K. LENSTRA, Branch-and-Bound Algorithmen voor het Handelsreizigersprobleem, Report BN 16, Mathematisch Centrum, Amsterdam, 1972.

19. J.K. LENSTRA, Clustering a Data Array and the Traveling-Salesman Problem, *Operations Res.* 22(1974)413-414.

20. G. LIESEGANG, M. RÜGER, Letter, *Operational Res. Quart.* 23(1972)591.

21. S. LIN, Computer Solutions of the Traveling Salesman Problem, *Bell System Tech. J.* 44(1965)2245-2269.

22. S. LIN, B.W. KERNIGHAN, An Effective Heuristic Algorithm for the Traveling-Salesman Problem, *Operations Res.* 21(1973)498-516.

23. J.D.C. LITTLE, K.G. MURTY, D.W. SWEENEY, C. KAREL, An Algorithm for the Traveling Salesman Problem, *Operations Res.* 11(1963)972-989.

24. W.T. McCORMICK, Jr., P.J. SCHWEITZER, T.W. WHITE, Problem Decomposition and Data Reorganization by a Clustering Technique, *Operations Res.* 20 (1972)993-1009.

25. H. MÜLLER-MERBACH, *Optimale Reihenfolgen*, Springer, Berlin etc., 1970.

26. J.F. MUTH, G.L. THOMPSON (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, 1963.

27. J. PIEHLER, Ein Beitrag zum Reihenfolgeproblem, *Unternehmensforschung* 4(1960)138-142.

28. R.C. PRIM, Shortest Connection Networks and Some Generalizations, *Bell System Tech. J.* 36(1957)1389-1401.

29. S.S. REDDI, C.V. RAMAMOORTHY, On the Flow-Shop Sequencing Problem with No Wait in Process, *Operational Res. Quart.* 23(1972)323-331.

30. S.S. REDDI, C.V. RAMAMOORTHY, Reply to Dr. Goyal's Comments, *Operational Res. Quart.* 24(1973)133-134.

31. S.S. REDDI, C.V. RAMAMOORTHY, A Scheduling Problem, *Operational Res. Quart.* 24(1973)441-446.

32. A.H.G. RINNOOY KAN, The Machine Scheduling Problem, Report BW 27, Mathematisch Centrum, Amsterdam, 1973; Report R/73/4, Graduate School of Management, Delft, 1973.

33. A.W. ROES, Enige Methoden ter Verkrijging van een Routeschema voor de Interinsulaire Scheepvaart in Indonesië, Unpublished Report, 1973.

34. M.M. SYSLO, On Some Machine Sequencing Problems (II), *Zastos. Mat.* 14 (1974)93-97.

35. J.M. VAN DEMAN, K.R. BAKER, Minimizing Mean Flowtime in the Flow Shop with No Intermediate Queues, *AIIE Transactions* 6(1974)28-34.

36. J. VISSCHERS, P. TEN KATE, BAKALG, een Bedradingsprogramma met Optimalisatie van Draadlengte, Report SO-1, Instituut voor Kernphysisch Onderzoek, Amsterdam, 1973.

37. D.A. WISMER, Solution of the Flowshop-Scheduling Problem with No Intermediate Queues, *Operations Res.* 20(1972)689-697.