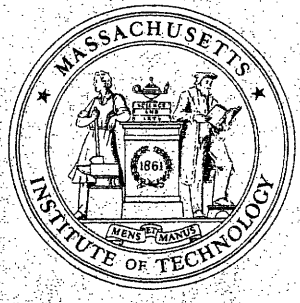
OPERATIONS RESEARCH CENTER

working paper



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A PC-Oriented Interactive Tool for Finding Efficient Solutions to Emergency Planning Problems

by

Oli B.G. Madsen and Richard C. Larson

OR 186-88 September 1988

A PC-Oriented Interactive Tool for Finding Efficient Solutions to Emergency Planning Problems

by

Oli B.G. Madsen
IMSOR
The Institute of Mathematical Statistics
and Operations Research
The Technical University of Denmark
DK-2800 Lyngby
Denmark

and

Richard C. Larson
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139
USA

September 1988

Table of Contents

Abstract

We outline an interactive, multi-objective, personal computer (PC)-oriented procedure to reposition facilities and to sectorize a district in a spatially distributed queueing system. We model the queueing system itself as an approximate hypercube queueing model. The procedure we describe is heuristic and is intended for use in the strategic or tactical planning of emergency systems such as those involving police patrols and ambulance services. It has two phases, each with different objectives. Phase 1 deals with one primary objective: to minimize the mean region-wide travel time. It results in a facility (unit) positioning and a sectoring of the district. Phase 2 deals with three, usually conflicting objectives, all concerned with equalizing performance measures such as workload and specific travel times. Both phases assume a close interactive cooperation with the user.

1. Introduction

The hypercube model, developed as a descriptive model during the early 1970s, made it possible to analyze emergency systems in a realistic way by modeling such systems as spatially distributed queueing systems. Since 1975, it has proven its applicability successfully in practice throughout the United States and in some parts of Europe. (For a more detailed description of the model, see [1], chapter 5; [2]; and [3].)

The hypercube model's function is chiefly descriptive: it does not suggest improved deployment options. It supplies the user with a voluminous output of performance measures from a given sector design, unit allocation, and priority rules. But eyeballing so much information and choosing among alternatives is not always a simple task. The user needs tools to assist him or her in interpreting and aggregating the hypercube output and choosing among alternatives that are often based on several conflicting demands.

This report describes some procedures for introducing optimization steps (or rather, steps for finding the near-optimum) in the hypercube model. We group these procedures into two phases. Phase 1 is concerned with one primary objective: to minimize the mean region-wide travel time. It results in a unit positioning and a sectoring of the district. Phase 2 deals with equalizing three additional performance measures: workloads of patrol units, average travel times to calls in each geographical sector, and average travel times to calls in each geographical atom.

Both phases are intended to be implemented in a personal computer environment, making it possible for the user to control the extent to which he or she is assisted by the computer.

Figure 1 shows the two phases and their relationship to the PC environment. The user feeds the decisional and geographical data needed for both phases into the

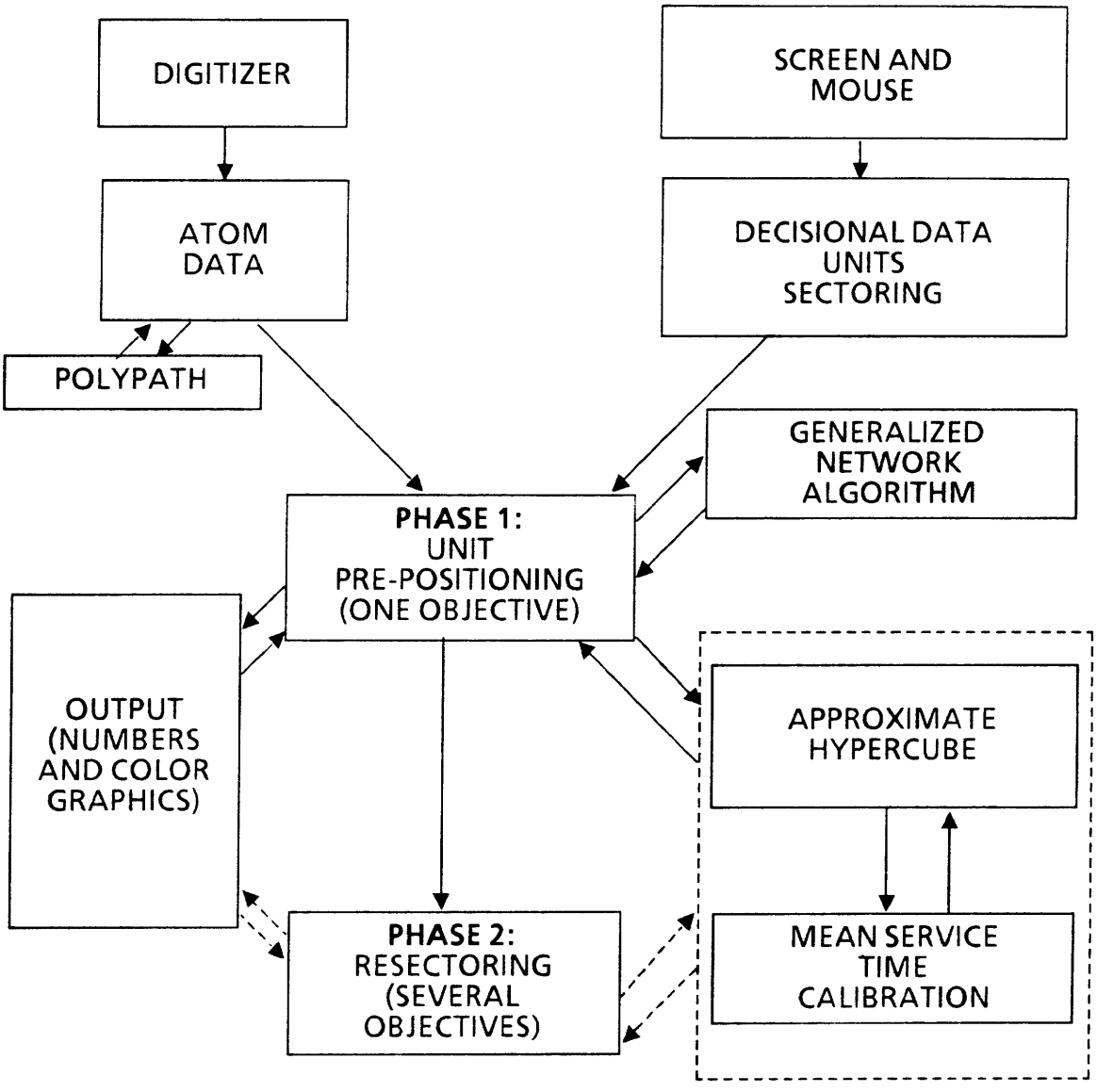


Figure 1. The unit pre-positioning and resectoring environment.

computer by means of a graphical device such as a digitizer, mouse, or colorscreen. Supporting algorithms such as Polypath and a Generalized Network Algorithm are connected to the system. Both phases use the hypercube model as a supporting subsystem. The output is displayed in a "user friendly" way by means of color graphics.

Section 2 describes Phase 1, and Section 3 describes Phase 2. Section 4 offers some concluding remarks. We assume that the reader has some prior knowledge of the hypercube model as described, for example, in [1], chapter 5,or [3]. In all the practical considerations connected with the solvability of the model, we assume that the problem size is no greater than 200 geographical atoms and 15 units.

2. Minimizing the Mean Region-Wide Travel Time in the Hypercube Model by Unit Pre-positioning

In this section, we describe an iterative procedure for optimally (or near-optimally) pre-positioning units in a hypercube model. The procedure is very similar to those described in [4] and [2], neither of which guarantee convergence to a global optimum, but only to a local one. The terminology we use is similar to that used in [1].

In short, the procedure is as follows:

1. Specify the initial locations of the units.
2. Determine the spatial response pattern.
3. Relocate the units to minimize the mean region-wide travel time.
4. Did the location of units change from the last iteration? If yes, go to Step 2; if no, then stop.

Step 2 may be performed by running the hypercube model or the approximate hypercube model, or by using some very simple approximation formulas. Step 3 may be accomplished by solving a very simple linear programming problem – so simple, in fact, that we can state the solution formula analytically.

We will now discuss the procedure in more detail.

2.1. Step 1: Specifying the Initial Location of the Units

In this step we determine an initial location of the units as well as an initial sectoring of the district.

There are two possibilities: Either the user or the computer can perform the initial sectoring of the district and locating of units. The first alternative is probably preferable, since it allows the user to specify the initial conditions, based, for example, on his knowledge of the existing system partitioning, or, if the system is new, on his experience with other systems. Alternatively, the computer may do the job – for example, by solving an N-median problem (see [1], p. 435) where N is the number of units. In either case, the demand in each node is set proportional to the call rate λ_j in geographical atom j, and the sectoring is determined by attaching the geographical atoms to the nearest unit. This procedure leads automatically to contiguous sectors.

2.2. Step 2: Determining the Spatial Response Pattern

The usual data needed for the hypercube model are as follows. For each of the N_A geographical atoms, we must have

- the coordinates of each atom center,
- the area of each atom,
- the number of calls for service, and
- dispatch preferences or a fixed dispatch policy.

For each of the N units, we need

- the spatial allocation of the unit,
- the response speed, and
- the service time for a unit.

Furthermore, we need

- the queue capacity and
- the travel times between atoms (may be calculated from the coordinates),

plus some technical details.

To proceed to Step 3, we must determine f_{nj}, the fraction of answered service requests that take server n to geographical atom j. This is also called the spatial response pattern. We could find f_{nj} by running the hypercube model, but for a large problem (e.g., $N_A = 200$ and $N = 15$) the approach is very time consuming – in

particular, because the hypercube model must be solved as many times as Step 2 is met in the unit positioning procedure. A better idea under these circumstances would be to apply the approximate hypercube formulas as follows (assuming an infinite-capacity queue). See Appendix D for a list of frequently used symbols.

$$Q(N,\rho,j)=\frac{\sum_{k=j}^{n-1}\left[\frac{(N-j-1)!\,(N-k)}{(k-j)!}\right]\frac{N^k}{N!}\rho^{k-j}}{(1-\rho)\left[\sum_{i=0}^{n-1}\frac{N^i}{i!}\rho^i\right]+\frac{N^N\rho^N}{N!}}\qquad \text{for } j=0,1,\ldots,N-1. \tag{1}$$

(See Formula 5.48, p. 328, in [1]).

We now perform the following calculation:

$$R_n^F=\sum_{a\in G_n^1}\lambda_a+\sum_{a\in G_n^2}\lambda_a Q(N,\rho,1)\rho_{m_{a1}}+\sum_{a\in G_n^3}\lambda_a Q(N,\rho,2)\rho_{m_{a1}}\cdot\rho_{m_{a2}}$$
$$+\ldots+\sum_{a\in G_n^N}\lambda_a Q(N,\rho,N-1)\rho_{m_{a1}}\rho_{m_{a2}}\ldots\rho_{m_{a(N-1)}}. \tag{2}$$

(See Formula 5.52, p. 331, in [1]).

Ultimately, we find the server utilization factors

$$\rho_n=\frac{R_n^F+\lambda_D}{1+R_n^F}=\frac{R_n^F+\lambda P_Q'}{1+R_n^F} \tag{3}$$

(see Formula 5.53, p. 331 in [1]),

where

$$P_Q'=P_Q+P_{B_2N_{-1}}.$$

P'_Q is the steady-state probability that a randomly arriving customer incurs a positive queue delay before entering service (and is thus equal to the steady-state probability that all N servers are busy). P_Q is the steady-state probability that a queue of positive length exists. Both P'_Q and P_Q can be determined from the well-known M/M/N model and will not change from iteration to iteration.

We can find ρ_n iteratively from (2) and (3) in the following way (as shown in [1], p. 322):

WORKLOAD ALGORITHM (WLA)

Step 0: Initialization

$\rho = \lambda / (N\mu)$

set i = 1

set $\rho_n(0) = \rho$, n = 1, 2, . . . , N at the outset, or

$= \rho_n$ from the previous use of this procedure;

n = 1, 2, . . . , N.

Step 1: Iteration

compute R_n^F from (2) and $\rho_n(i)$ from (3), using ρ_m (i–1) in computing R_n^F; do this for all n = 1, 2, . . . , N.

Step 2: Normalization

compute

$$\Gamma = \left[N^{-1} \sum_{n=1}^{N} \rho_n(i) / \rho \right]^{-1}$$

$$\rho_n(i) \leftarrow \Gamma \rho_n(i) .$$

Step 3: Convergence Test

$$\max_{n} |\rho_n(i) - \rho_n(i-1)| \stackrel{?}{\geq} \varepsilon$$

Yes: $i \leftarrow i + 1$, go to Step 1.

No: STOP, set $\rho_n = \rho_n(i)$; $n = 1, \ldots, N$.

For ε approximately 0.01, three iterations are usually sufficient. If we begin the procedure with good estimates of ρ_n from the previous "master iteration," we will probably need less than three iterations.

Having determined an estimate for the workloads ρ_n, we can calculate the spatial response pattern. Let

f_{nj} = fraction of answered service requests that take server n to geographical atom j;

$f_{nj}^{[1]}$ = fraction of answered service requests that take server n to geographical atom j and incur no queue delay; and

$f_{nj}^{[2]}$ = fraction of answered service requests that take server n to geographical atom j and incur a positive queue delay.

For the infinite-line-capacity case, we can approximate f_{nj} as follows (see Problem 5.11, pp. 353-354 in [1]):

$$f^{[1]}_{m_{kj}k} = \left(\frac{\lambda_k}{\lambda}\right) Q(N, \rho, j-1) \left[\prod_{l=1}^{j-1} \rho_{m_{kl}}\right] (1 - \rho_{m_{kj}}) \tag{4}$$

$$\Gamma_k = \sum_{n=1}^{N} f^{[1]}_{nk} \qquad k = 1, \ldots, N_A \tag{5}$$

$$C_k = (1 - P\{S_N\}) \frac{\lambda_k}{\lambda} \qquad k = 1, \ldots, N_A \tag{6}$$

$$f^{[1]}_{nk} \leftarrow f^{[1]}_{nk} \frac{C_k}{\Gamma_k} \qquad k = 1, \ldots, N_A \tag{7}$$

$$f_{nk}^{[2]} = \frac{\lambda_k}{\lambda} P_Q' \frac{1}{N} \tag{8}$$

$$f_{nk} = f_{nk}^{[1]} + f_{nk}^{[2]}, \tag{9}$$

where m_{kj} is the identification number of the jth preferred server for atom k, and $P\{S_N\}$ is the steady-state probability that the equivalent M/M/N system is in state N (i.e., there are N "customers" in the system.)

Appendix A shows formulas (4)-(9) in algorithmic form.

The zero-line-capacity case requires the following modifications (according to [1], p. 353):

(i) λ_D in formula (3) is zero;

(ii) at step 0,

$$\rho_n(0) = \lambda(1 - P'\{S_N\})/N$$

where

$$P'\{S_k\} = \frac{N^k \eta^k P'\{S_o\}}{k!} \qquad k = 1, \ldots, N$$

$$P'\{S_o\} = \left[\sum_{i=0}^{N} \frac{N^i \eta^i}{i!} \right]^{-1}$$

$$\eta = \frac{\lambda}{N};$$

(iii) replace Q(N, ρ, j) by

$$Q^*(N, \eta, j) \left(\frac{1}{1 - P'\{S_N\}} \right)^j \frac{1}{1 + \eta P'\{S_N\}/(1-\eta)},$$

where Q*(N, η, j) is equal to Q(N, ρ, j) but with $P\{S_o\}$ replaced by $P'\{S_o\}$.

2.3. Step 3: Relocating the Units

After we have found f_{nj} in Step 2, we can formulate a simple linear programming model to relocate the units (assuming the response pattern as reflected by f_{nj} remains fixed):

$$\text{minimize } \overline{T} = \sum_{n=1}^{N} \sum_{j=1}^{N_A} \sum_{k=1}^{N_A} f_{nj}\, \tau_{kj}\, \ell_{nk}$$

$$\text{subject to} \quad \sum_{k=1}^{N_A} \ell_{nk} = 1 \qquad n = 1, \ldots, N \tag{10}$$

$$\ell_{nk} \geq 0 \qquad n = 1, \ldots, N; k = 1, \ldots, N_A,$$

where $\overline{T}$ is the mean region-wide travel time, τ_{kj} is the minimum mean travel time from atom k to atom j, and ℓ_{nk} is the probability that response unit n is located in atom k. The solution to (10) is very simple because the constraints are independent of each other. For each n, find

$$\min_{k} \sum_{j=1}^{N_A} f_{nj}\, \tau_{kj} = Z_{nk_n^*}$$

$$\ell_{nk} = \begin{cases} 1 & \text{for } k = k_n^* \\ 0 & \text{otherwise} \end{cases} \qquad n = 1, \ldots, N.$$

We should note that ℓ_{nk} will automatically be integer, i.e., either 0 or 1, which means that each unit n will be located with probability 1 in a certain atom k_n^*. Comparing the new ℓ_{nk} values to the previous ones for a given n, we find either that ℓ_{nk} are the same or that two ℓ_{nk} values have changed from 1 to 0 – respectively, from 0 to 1 – i.e., they have exchanged values. This small change should result in a very fast recomputation of all the preferences (priorities) that are based on

$$t_{nj} = \sum_{k=1}^{N_A} \ell_{nk}\, \tau_{kj} \ , \tag{11}$$

where t_{nj} is the mean time required for unit n, when available, to travel to atom j. The change in t_{nj} is either zero or

$$\left(t^{NEW}_{k_n^j} - t^{OLD}_{k_n^j} \right).$$

New preferences give a new m_{kj} matrix (see Table 5-4, p. 305, and p. 330, in [1]) and new G_n^j sets (p. 330, and Table 5-8, in [1]), and the approximate hypercube model mentioned in Step 2 can be rerun if changes occurred in ℓ_{nk}. We can initiate the hypercube model (see WLA , Step 0, in the preceding section) by setting

$$\rho_n(0) = \sum_{j=1}^{N_A} f_{nj} .$$

In ambulance applications, it is preferable that all ℓ_{nk} are {0, 1}, as the LP model just mentioned provides. However, police car applications usually require that the unit patrols in several atoms when it is "idle" (preventive patrol). Jarvis ([4], pp. 122-124) has added additional constraints to Problem 10 to depict mobile units:

$$\sum_{n=1}^{N} (1-\rho_n)\,\ell_{nk} \geq X \, \frac{\lambda_k}{\lambda} \, N(1-\rho) \qquad k = 1, \ldots, N_A , \tag{12}$$

where X is the minimum allowed fraction of the target level for preventive patrol. The left-hand side of the equation specifies the total fraction of preventive patrol that is available for atom k. $N(1-\rho)$ is the total preventive patrol effort available from all the servers. λ_k/λ specifies that the minimum effort allocated to each atom should be roughly proportional to the fraction of calls originating from atom k. X (or, rather, 1–X) specifies how much slack is allowed from the ideal allocation. For X close to 1, no feasible solution may exist to Problem 10 with constraints (12).

Constraints (12) may result in overlapping areas of preventive patrol – i.e., for a fixed k, several ℓ_{nk} are positive. If such overlapping areas are not acceptable, additional constraints of the following type may be neccessary:

$$\left.\begin{aligned} &\ell_{nk} \le y_{nk} \quad n = 1,\ldots,N; \qquad k = 1,\ldots,N_A \\ &\sum_{n=1}^{N} y_{nk} \le 1 \qquad\qquad\qquad k = 1,\ldots,N_A \\ &y_{nk} = \begin{cases} 1 & if\ \ell_{nk} > 0 \\ 0 & if\ \ell_{nk} = 0 \end{cases} \end{aligned}\right\} \tag{13}$$

The introduction of (12) in Problem 10 will, in the general case, change the solution from a 0-1 solution to a continuous solution. The constraints will not be separable as before, and a standard linear programming (LP) routine will be necessary in order to solve the problem. For N = 15 and N_A = 200, for example, the LP problem will have 3,000 variables and 15 + 200 = 215 constraints – not a very large problem. Furthermore, it will have a matrix structure as in a classical transportation problem, with two coefficients greater than zero in each column. For this generalized transportation problem (GTP), it is possible to develop codes that take this special structure into consideration. In the following discussion, we will further examine the solution.

Adding constraints (13) to the problem makes it much more difficult to solve. Doing so turns Problems 10, 12 and 13 into mixed integer programming problems. For N = 15 and N_A = 200, we get

- 3,000 continuous variables,
- 15 + 200 + 15*200 = 3,215 constraints, and
- 3,000 0-1 variables.

Solving an integer programming problem of these dimensions to optimum would be very time consuming and could not be done on a personal computer. However, constraints (13) are so simple that it may be possible to imbed them directly in the linear programming code.

Detailed Discussion of the LP Model. We now restate the relocation model containing Equations 10 and 12.

$$\underset{\ell_{nk}}{\text{Minimize}}\ \overline{T} \quad \sum_{n=1}^{N}\sum_{j=1}^{N_A}\sum_{k=1}^{N_A} f_{nj}\,\tau_{kj}\,\ell_{nk}$$

$$\text{subject to} \quad \sum_{k=1}^{N_A} \ell_{nk} = 1 \qquad n = 1,\ldots,N$$

$$\sum_{n=1}^{N} (1-\rho_n)\,\ell_{nk} \geq X\,\frac{\lambda_k}{\lambda}\,N(1-\rho) \qquad k = 1,\ldots,N_A$$

$$\ell_{nk} \geq 0 \qquad n=1,\ldots,N; \qquad k = 1,\ldots,N_A\ .$$

Introducing the following identities,

$$c_{nk} \equiv \sum_{j=1}^{N_A} f_{nj}\,\tau_{kj}$$

$$a_n \equiv (1-\rho_n) \leq 1$$

$$b_k \equiv X\,\frac{\lambda_k}{\lambda}\,N(1-\rho),$$

we can write the LP problem as follows:

$$\underset{\ell_{nk}}{\text{minimize}} = \sum_{n=1}^{N}\sum_{k=1}^{N_A} c_{nk}\,\ell_{nk} \tag{14}$$

$$\text{subject to} \quad \sum_{k=1}^{N_A} \ell_{nk} = 1 \qquad n = 1,\ldots,N \tag{15}$$

$$\sum_{n=1}^{N} a_n \ell_{nk} \geq b_k \qquad k = 1, \ldots, N_A \tag{16}$$

$$\ell_{nk} \geq 0 \qquad n = 1, \ldots, N; \quad k = 1, \ldots, N_A . \tag{17}$$

(14) is a special case of the generalized transportation problem in which $a_{nk} \equiv a_n$, i.e., in our problem the a_{nk} coefficients are independent of k. The constraint matrix has the following structure, where each box corresponds to a fixed k:

1 1 1 ... 1	1 1 1 ... 1	1 1 1 ... 1
$a_1\ a_2\ a_3 \ldots a_N$	$a_1\ a_2\ a_3 \ldots a_N$	$a_1\ a_2\ a_3 \ldots a_N$

It is easy to see that the matrix structure is exactly the same as for the transportation problem, in which $a_n = 1, n = 1, ..., N$. We can solve the generalized transportation problem by decomposition, or by using algorithms for solving a generalized network flow, or by generalized upper bounding techniques. We now discuss each of these approaches in more detail.

Decomposition.

Decomposition is a method well suited to solving large specially structured LP problems. The problem to be solved is decomposed into both a master problem that usually includes the constraints affecting all the variables, and a series of subproblems that each include only some of the variables. A master iteration includes one iteration in the master problem and solution of all the subproblems. The iteration in the master problem results in a new set of simplex multipliers that are used to modify the objective functions of the subproblems. The solution of the subproblems indicates how the next iteration in the master problem should be done. A more detailed discussion of decomposition can be found in [5].

Considering problem (14-17), we treat constraints (15) as constraints that affect all the variables and constraints (16-17) as the subproblems. Each of the N_A subproblem will be of the form

$$\left.\begin{array}{lll} \textit{minimize} & \sum_{n=1}^{N} (c_{nk} + \pi_n)\ell_{nk} & \\ \textit{subject to} & \sum_{n=1}^{N} a_n \ell_{nk} \geq b_k & \\ & \ell_{nk} \geq 0 \qquad n = 1, \ldots, N, & \end{array}\right\} \tag{18}$$

where π_n is the nth component of vector $\underline{\pi}$ of simplex multipliers that arise from the master problem.

The solution to (18) is very similar to (10), which also had only one constraint in each subproblem.

Find

$$\min_{n} \left(\frac{c_{nk} + \pi_n}{a_n} \right) = \frac{c_{n^*k} + \pi_{n^*}}{a_{n^*}}$$

and set

$$\ell_{nk} = \begin{cases} \dfrac{b_k}{a_{n^*}} & \text{for } n = n^* \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the new ℓ_{nk} can take decimal values that may even be greater than 1. This does not "damage" the solution because on the master level the solution is composed of convex combinations of the solutions from the subproblems.

If

$$\frac{c_{n^*k} + \pi_{n^*}}{a_{n^*}} < 0,$$

the solution to this specific part of the subproblem would be $\ell_{n^*k} = \infty$. We have not yet investigated the sign of π_n, which is the only component that may be negative. Our guess at present is that it is possible to prove that $\pi_n \geq 0$ and that, therefore, infinite solutions may not occur. However, even if we are wrong, no great damage has been done.

Figure 2 shows the solution procedure for (14).

In the master problem, we find an optimal solution using the usual linear programming methods. If N = 15, the master problem has 16 constraints and only one iteration is required. This should be easy to handle on a PC. Solving the subproblem is a simple scanning of coefficients, which is a very fast procedure.

Algorithm for Solving Generalized Network Flow. In the last decade, researchers have developed a number of algorithms that can handle LP-based graph problems: GNET [6], NETG [7], GENNET [8], and GNO/PC [9]. These procedures

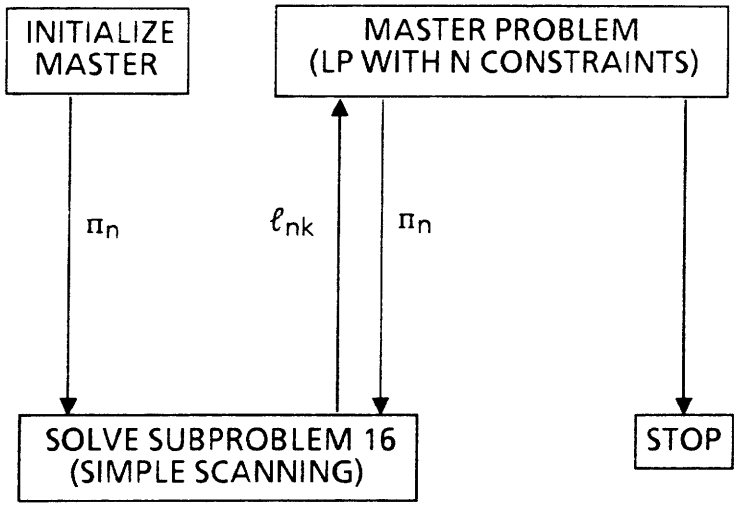


Figure 2. The Decomposition-based Solution Procedure for the Unit Relocation Problem

solve special structured linear programming problems whose coefficient matrix, ignoring simple upper- and lower- bound constraints, contains at most two non-zero entries in each column. Such problems are called generalized network problems or capacitated generalized transshipment problems. The solution method is still based on the simplex method, but instead of working with a matrix of coefficients and an inverse basis matrix, we store the necessary information as a network. A change of basis corresponds to setting the flow to the lower bound in one arc and increasing it beyond the lower bound in another. Using tree structures and efficient pointer and relabeling techniques, we can solve very large network problems in a short time. Some researchers [7] have reported that the special network code was 50 times faster than a sophisticated state-of-the-art linear programming code. This result is supported by the experiences mentioned in [9], in which networks of 50,000 arcs and 4,000 nodes were solved on an IBM PC/AT that had 512 K of memory and a mathematical coprocessor. GENNET is coded in FORTRAN IV and has been

implemented on an Apple II. GNO/PC is coded in C and can be used on several types of personal computer.

To estimate the computer time needed to solve Problem 14 with 215 nodes and 3,000 arcs on an IBM PC/AT, we will list some of the results of [8] and [9]. Brown and McBride [8] discuss three problems of a size similar to the ones we consider. Their characteristics are as follows.

Problem	Nodes	Arcs	Percentage of Capacitated Nodes	CPU time
NG 19	400	2443	20	1.04
NG 23	400	2836	40	0.91
NG 27	400	2676	80	0.79

The CPU times are in seconds on a mainframe IBM 370/168-3 computer with a FORTRAN H compiler. We estimate that the conversion factor to an IBM PC/AT with a mathematical coprocessor would be around 50, resulting in CPU times of 52, 45.4, and 39.3 seconds, respectively. Nultey and Trick [9] report 19 test runs of problems that range in size from 4,000 nodes + 50,000 arcs to 1,000 nodes + 5,000 arcs – problems all considerably larger than those we consider in this paper. For a fixed number of nodes, N, the computer time is linear in the number of arcs m:

$$T_N(m) = a_N + b_N m,$$

where the coefficients a_N and b_N are positive and $b_N < b_{N+1} < b_{N+2}$. For N = 1,000, 2,000, 3,000, and 4,000, the computer times in minutes on an IBM PC/AT with mathematical coprocessor are

$T_{1000}(m) = 6.30 + 124.0 \times 10^{-5} m$
$T_{2000}(m) = 12.50 + 181.6 \times 10^{-5} m$
$T_{3000}(m) = 10.00 + 262.0 \times 10^{-5} m$
$T_{4000}(m) = 9.20 + 363.4 \times 10^{-5} m,$

where the decimals indicate hundredths of minutes. A good estimate for b_{215} would be 87.5×10^{-5}. Estimating a_{215} is rather difficult, but a rough approximation might be $1.25 \leqq a_{215} \leqq 3.19$. Since the problem has 3,000 arcs, the expected computer time will be between 3 minutes, 53 seconds and 5 minutes, 49 seconds. Only a test run of Problem 14 will give us the actual computation time, but from the two examples just mentioned, we can expect that it will be in the range of 1 to 6 minutes on an IBM PC/AT with mathematical coprocessor. Furthermore, we can reduce the computation time considerably for the second and subsequent solutions. Most codes include an advanced start option, which allows the user to specify an initial solution. A natural choice would be the optimal solution from the previous run. The only changes will be in c_{nk} and a_n, while b_k remains constant. The changes in a_n and perhaps in c_{nk} will be rather small, so the previous optimal solution will be a good choice for an initial solution.

If the planning procedure is to be run on-line on a PC, a solution time of 6 minutes for the linear programming step alone is excessive: one minute is probably a reasonable upper limit. If numerical experiments show that the procedure takes more than one minute of computer time, the code should be speeded up or an alternative method used, as we will discuss in Section 2.4.

<u>Generalized Upper Bounds (GUB) Techniques</u>. Problem (14) may be considered an LP problem with generalized upper bounds (GUB), where

$$\sum_{k=1}^{N_A} \ell_{nk} = 1$$

are the GUBs. Some researchers [10] have suggested a special version of the simplex method in which the GUB rows are treated indirectly and only the "ordinary" rows are in the constraint matrix. In the present case, this approach would reduce the

number of rows from 215 to 200 and keep the number of variables to 3,000, resulting in at best a very modest reduction in computer time.

Evaluation of Solution Methods. We now summarize briefly the usefulness of the solution methods we have discussed so far.

Using an existing *general LP code* for solving the problem would probably be too time consuming on a PC. It could be done, for example, by setting $N = 10$ and $N_A = 150$, but the procedure would be very clumsy. If constraints (13) (no overlapping) were included, the only way to proceed would be either to determine whether a generalized network flow algorithm could handle this additional constraint or to use decomposition with indirect satisfaction of (13).

The *decomposition approach* may be feasible, but convergence may be slow (although this is not certain). The method requires parts of a simplex code and could easily be implemented on a PC.

Using existing software for *generalized network flow* would be a fast and reliable way to solve the problem. The difficulty lies in the cost and in the interface between program parts.

The *GUB techniques* require an excessive amount of coding and might gain us nothing.

Knowing either that a_n is independent of k or that $o \leq a_n \leq 1$, we do not see an advantage to any of these methods.

2.4 An Alternative Approach to the Police Planning Problem

In order to allow continuous ℓ_{nk}, we could add a new step to the procedure described at the beginning of Section 2. The idea is similar to that of constraint (12),

in which the preventive patrol is spread out proportional to the fraction of calls originating from the atoms.

Unit Pre-positioning Procedure with Preventive Patrol (UP4)

1. Specify the initial (deterministic) locations of the units.
2. Determine the spatial response pattern (i.e., run the hypercube model).
3. Holding the response patterns fixed, relocate the units (deterministically) to minimize mean region-wide travel time (i.e., run the simple LP model).
4. a. Allocate atoms to the nearest center (i.e., where $\ell_{nk} = 1$). Doing so sectors the region. Number the sectors n = 1, ..., N.

   b. Create probabilistic unit locations by setting

$$\lambda^{[n]} = \sum_{j \in \text{atoms in } n} \lambda_j \text{ and}$$

$$\ell_{nk} = \lambda_k / \lambda^{[n]}$$

5. Did the units' probabilistic location change from the last iteration? If yes, go to Step 2; if no, then stop.

In this procedure, Steps 1, 2, 3 and 5 are identical to those in the procedure described at the beginning of Section 2. Steps 4a and 4b are new. The termination criterion in Step 5 could use the integer ℓ_{nk} from Step 3. Step 3 itself corresponds to the simple case with LP model (10).

The advantage of UP4 over the previous procedure with LP model (10) and (12) is that the LP model the new procedure must solve is much simpler. The disadvantage is the uncertainty on convergence and the quality of the solution, compared to the previous procedure. Only through numerical experiments can we determine which procedure is preferable.

2.5 Mean Service Time Calibration

We define the mean service time μ_n^{-1} of server (unit) n as the travel time plus on-scene time and possibly related off-scene time. The basic hypercube model assumes

the service time to be independent of the identity of the server, the location of the customer, and the history of the system, i.e.,

$$\mu_1^{-1} = \mu_2^{-1} = \ldots = \mu_n^{-1} = \mu^{-1}.$$

This assumption is reasonable in certain applications, such as police planning, but may be too rough an approximation in such areas as ambulance planning.

In the following expression, we assume that the on-scene and off-scene time are constant, and that the mean travel time of server n may vary:

$$\sum_{j=1}^{N_A} f_{nj} t_{nj}.$$

f_{nj} is the fraction of answered service requests that takes server n to geographical atom j, and t_{nj} is the mean time for unit (server) n, when available, to travel to geographical atom j. When f_{nj} and t_{nj} change, the mean travel time changes. The problem with this formulation is that we need to know μ_n before we can determine it. Jarvis [11] has developed the mathematics for a model that includes μ_{nj}^{-1} – i.e., the mean time for unit n to service a request in atom j. However, for practical applications, Larson [12] and Jarvis [4] have suggested an approximate procedure, called mean service time calibration (MSTC).

The Mean Service Time Calibration Procedure (MSTC)

1. Guess the mean service time of each server n as $(\mu_n^{-1})_1$. Set m = 1.
2. Execute the hypercube model and observe the computed mean service time $(\mu_n^{-1})_{m+1}$ for each server n (which is the sum of the model-computed travel times and the on-scene/off-scene service time).
3. If max $[\,|(\mu_n^{-1})_{m+1} - (\mu_n^{-1})_m|] < \varepsilon$ then stop, and use $(\mu_n^{-1})_{m+1}$. If the test fails, enter $\{(\mu_n^{-1})_{m+1}\}$ for n = 1, ... N into the model, set m ← m + 1, and return to Step 2. Otherwise STOP: the calibration is completed.

3. Sectoring of Primary Responsibility Areas

In Section 2, we discussed a method to pre-position response units in order to minimize the near region-wide travel time. We chose this as our primary goal because it is usually considered the most important performance measure for an emergency system. Small mean region-wide travel times result in small response times and quick responses, which are essential in all emergency systems. The procedures in Section 2 also result in a sectoring of primary responsibility areas. However, we must also consider other objectives that focus more on efficient and equal use of available manpower and equipment. The final goal is often to obtain a good balance among several conflicting goals. The final evaluation is often not made by the computer alone, but is based on the judgment of an experienced manager of the emergency system. He or she makes an evaluation assisted by an interactive computer system which, in a systematic way, facilitates the decision process by evaluating different alternatives and by determining trade-offs between different objectives.

The objectives we will now discuss all involve equalization, meaning that we attempt to equalize a performance measure from atom to atom or from sector to sector. We have chosen the following objectives:

- to equalize workloads of patrol units;
- to equalize average travel times to calls in each sector;
- to equalize average travel times to calls in each atom.

If necessary, more goals could be added. Let us suppose that Phase 1 has provided us with a sectoring, or that this phase has been surpassed and the user has specified a sectoring. We can now restate the resectoring problem:

Within one global district having N response units, how should the region be partitioned into areas of primary responsibility (sectors) so as to best achieve one, or combinations, of the above-mentioned goals?

3.1 Conflicting Goals

As we have mentioned, the goals just listed usually conflict. Equalizing the workloads will not affect the average workload ρ which, for equal service times μ^{-1}, is equal to $\lambda/N\mu$, which is independent of the sectoring. For varying service times μ_i^{-1}, ρ is more difficult to evaluate and usually slightly dependent on the sectoring.

Equalizing the workloads typically will not noticeably affect the mean region-wide travel time. However, some of the other performance measures may change: for example, the imbalance in sector travel times will usually increase.

Balancing sector travel times will usually increase the mean region-wide travel time, but it will also increase the workload imbalance. We will elaborate on such conflicts later in this section. To solve problems with conflicting goals, it is necessary to use either trade-offs between goals or multi-attribute utility functions. We will now discuss methods for sectoring primary responsibility areas.

3.2 Solution Methods

The resectoring problem is in principle a complicated nonlinear constrained optimization problem containing, in part, a "black-box" system (the hypercube model). Attempting to solve the problem to optimality would cause several difficulties:

- Even if we could solve the problem to optimality, doing so would be very time consuming on a mainframe computer and extremely so on a PC. Interactive on-line use of the PC would be impossible.

- When conflicting goals exist, it is very difficult to state precisely what is understood by an optimal solution. The problem therefore has considerable uncertainty connected to it.

- Many constraints exist that are not well quantifiable – for example, the requirement that a sector should be contiguous and have a reasonable shape.

To avoid these difficulties, we suggest an interactive "near optimal" procedure based mainly on the work of Chelst [13 and 15] and Bodily [16 and 17]. Chelst measures the imbalances in the system as the maximum difference of a specific performance measure. He focuses on all atoms located at the borders of a sector and considers each of them eligible for a switch in sector identity to that of the neighboring sector. Considering one-at-a-time interchanges, his algorithm follows a steepest-descent path to a local minimum by successively selecting the most productive atom exchanges with respect to the specific goals mentioned at the beginning of this section. The algorithm makes sense because of its simplicity, the reported numerical results, and the special attention it gives to sector contiguity and compactness.

However, Chelst considers only one objective at a time. Bodily [16 and 17] has extended the ideas of Chelst by identifying three sets of decision makers, each with a different objective function. He proposes a multiattribute utility function such that the three groups' interests are aggregated into one nonlinear "quasi-utility" function that could be used in the Chelst procedure. The Bodily approach, however, requires information on the utility functions of the different groups. Those functions may be difficult for the potential user of the system to determine and interpret.

We have developed a compromise, using the procedures of Chelst but adding the possibility of establishing trade-offs between goals, weighting goals together, and identifying efficient solutions (i.e., solutions that are not dominated by other

solutions). Our approach results in a simple solution procedure that is well suited for an interactive environment and can be easily understood by the user.

3.3 The Resectoring Problem

We now present parts of the Chelst algorithm. We have divided the presentation into three parts, each corresponding to one of the objectives mentioned at the beginning of Section 3. For more detailed information, see [13].

Equalizing Workloads. We assume that an initial sectoring exists and that information is available concerning which atoms are neighbors.

(WS1) The first step in equalizing workloads is to find the sector with the workload farthest from the mean, either above or below it.

If the workload for sector car A is the one farthest from the average, and it is above average, then we follow procedures (WS2) and (WS3).

(WS2) A search is made of the sectors contiguous to sector A to determine which of the contiguous sectors has the sector car B with the lowest workload.

(WS3) The atom transferred from A to B is the atom in sector A that is contiguous with B, and whose center of mass is closest to the center of gravity (weighted call rates) of sector B. This atom transfer must, of course, not violate any contiguity constraint. If it does, the second closest atom is transferred.

If the worst imbalance occurs in an underutilized sector car, A, then we follow procedures (WS4) and (WS5).

(WS4) A search is made for the closest (as defined in WS2) contiguous atom, C, to sector A rather than first searching for the sector B that is contiguous to A and whose sector car B has the highest workload.

(WS5) Once the atom, C, that is to be transferred into A has been found, the sector that will lose the atom is implicitly determined.

The user may specify how many times steps (WS1-WS5) are to be carried out before a new user interaction takes place. The procedure stops at the user's command.

Appendix B presents an algorithm, coded in PL/1, to check whether a sector is contiguous.

Equalizing Sector Travel Times.

(SS1) First, sector A with the highest average travel time is located.

(SS2) The sectors contiguous to sector A are compared to determine which sector, B, has the lowest average travel time.

(SS3) Of the atoms in A contiguous to sector B, the atom farthest away from the center of gravity of A is transferred.

(SS4) If, however, it is found that the removal of a particular atom under Step (SS3) increases the average travel time for sector A, then Step (SS3) is repeated to find the second farthest atom from the center of gravity.

As we have mentioned, the termination of Steps (SS1-SS4) is left to the user.

Equalizing Atom Travel Times.

(AS1) The program begins by identifying the atom, T, with the highest travel time.

(AS2) A check is made to determine whether or not atom T is closer (and contiguous) to the center of gravity of a sector other than the one to which it is presently assigned. If the closest center of gravity is a sector C other than its own, the preferred option offered the user is to transfer atom T to sector C.

(AS3) The one (or two) atom(s), F, farthest away from atom T that is in the same sector and also contiguous to another sector is offered as a possible atom transfer. If that atom, F, is contiguous to more than one sector, it should be transferred to the sector whose center of gravity is closest.

In most cases, Step (AS2) will prove fruitless and only (AS3) will yield any potential atom transfers.

Additional Comments on the Algorithms. A series of additional constraints may be added to the system such that whenever a potential sector design violates these constraints, the user is notified. Examples of such constraints are:

- The maximum and/or minimum number of atoms in a sector, or maximum or minimum area of a sector.
- The maximum workload for any particular sector car.

It should be possible at any time to switch between the algorithms in order to focus on another objective. The results from any of the solutions should be stored so they can be recalled if necessary.

Contiguity. For all three algorithms discussed in this section, it is necessary to find out whether an atom transfer will result in noncontiguous sectors. To check the contiguity, the user must specify which atoms are neighbors, i.e., which atoms have common borders of a length of at least ε. The contiguity check algorithm in Appendix B requires this information as an atom-neighbor matrix A__A__TIG. In the case of 200 atoms, this symmetric matrix will contain 40,000 binary elements, which is a rather large matrix. The matrix will be very sparse, reflecting the fact

that usually an atom has only a few neighbors. Analyzing the case-example reported in [18] concerning 70 atoms in Police District 4 in Boston yields the following statistics:

N_3	2	3	4	5	6	7	8	9
Number of atoms with N_3 neighbors	3	11	31	14	7	3	0	1

In fact, atom 59 has 9 neighbors, and atoms 3, 17, and 70 each have 7 neighbors. Assuming that the maximum number of neighbors is 10, for example, we could store the neighbor information in a 200*10 matrix (not binary). In the present case, the 70*10 matrix will have a density of 305/700 = 43.6%, indicating that perhaps the information should be stored in strings of variable length. It is obviously more time consuming to check contiguity using the condensed matrix or strings.

Visual Check of Contiguity. The geographical data on each atom consists of the coordinates of center, the area, and the neighbor information just mentioned. Figure 3 gives a visual presentation of this information, showing Police District 4 in Boston. The centers may be differently colored according to which sector they belong to; however, it is very difficult to determine from the figure whether two atoms are neighbors. Atoms 70 and 4 are neighbors; atoms 67 and 69 are not. To clarify the relationship, we introduce a network connecting centers to neighboring atoms. Figure 4 shows the result.

Figure 3
The Atom Centers in Boston District 4

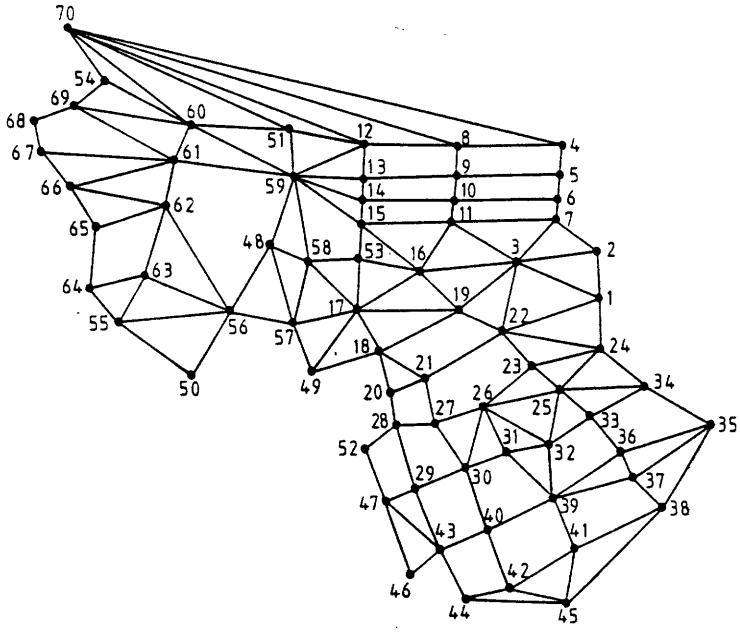


Figure 4
The Network Showing Atom-Neighborship
(Boston District 4)

3.4 Trade-off Functions and Efficient Solutions

We will use an example to illustrate the use of trade-off functions. Chelst ([13], pp. 100-104) presents a real-life example in which he equalizes the sector travel times by transferring atoms from one sector to another. The data were taken from the New Haven Department of Police Services District NORA, and involved 7 sectors and 78 atoms. In 15 iterations, the sector travel time imbalance (the highest sector travel time divided by the lowest sector travel time) was decreased from 1.993 to 1.286, an improvement of 35.5%. At the same time, the average distance traveled was increased by 5.8% and the workload imbalance was increased by 23.3% from 1.286 to 1.586 (highest/lowest). These results support our observation (see Section 3.1) that balancing sector travel time increases the imbalances in other performance measures.

Figure 5 shows the imbalances, iteration-for-iteration. A is the initial solution and P is the final solution. The first five iterations leading to point F yield improvements in both imbalances (i.e., there are no conflicting objectives), while the remaining iterations result in a 37% increase in the workload imbalance. The figure shows that solutions A, B, C, D, E, F, J, L, and N will never be preferable (comparing only these two imbalances) since we can always find solutions that dominate the solutions mentioned above: [For example, solution K dominates solution J.] Eliminating 9 out of 16 points leaves 7 to consider. We call these the efficient set of points, meaning that no point can be said to be better than another point or combination of points, relative to the two objectives. Figure 6 shows the efficient set of points. The remaining problem is how to choose among those points. Here user interaction is very important because only the user can evaluate and compare combinations of these two measures of imbalance. If reduction of travel distance imbalance is weighted very high, solution G should be chosen. If reduction

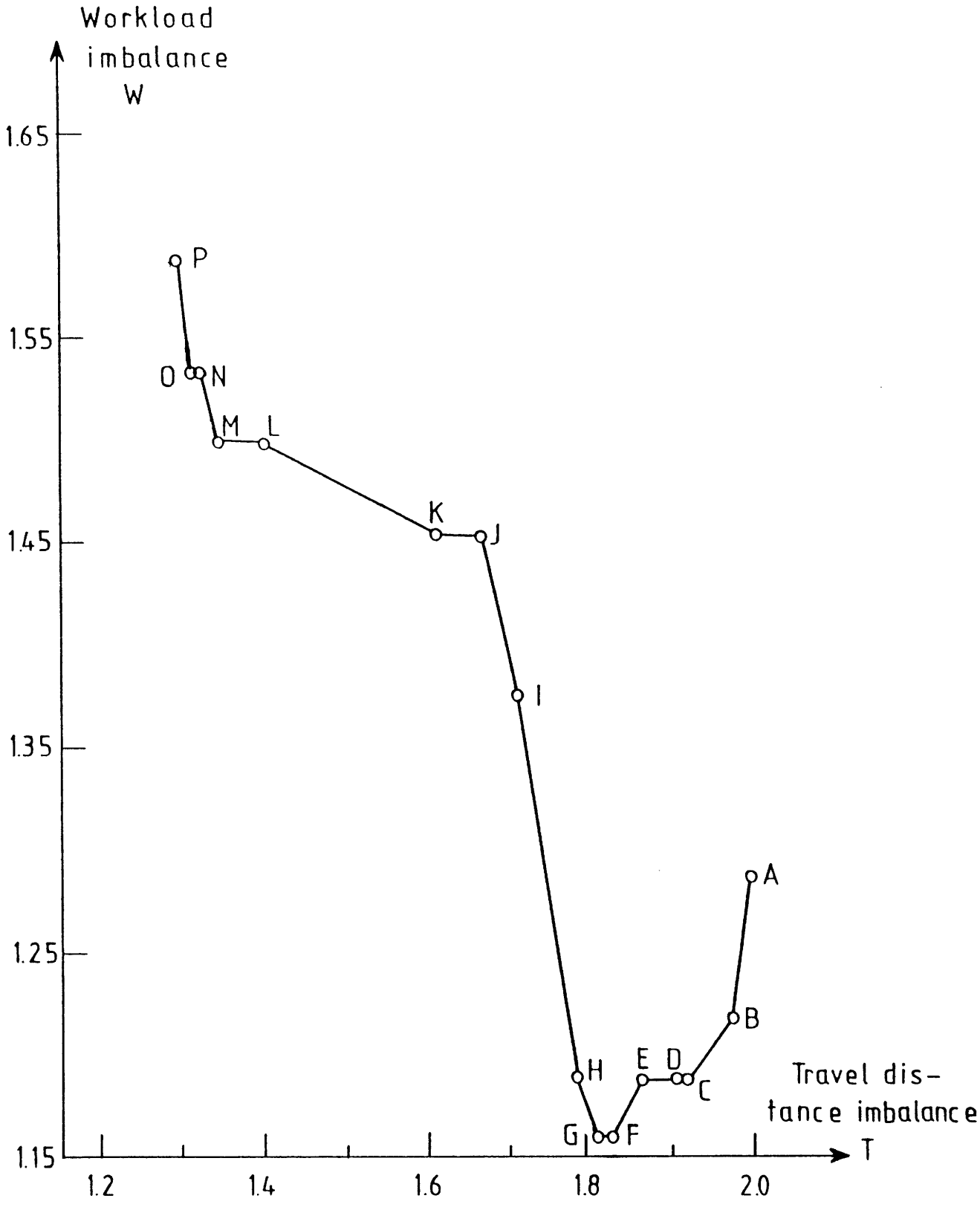


Figure 5
Equalizing Sector Travel Time Imbalances

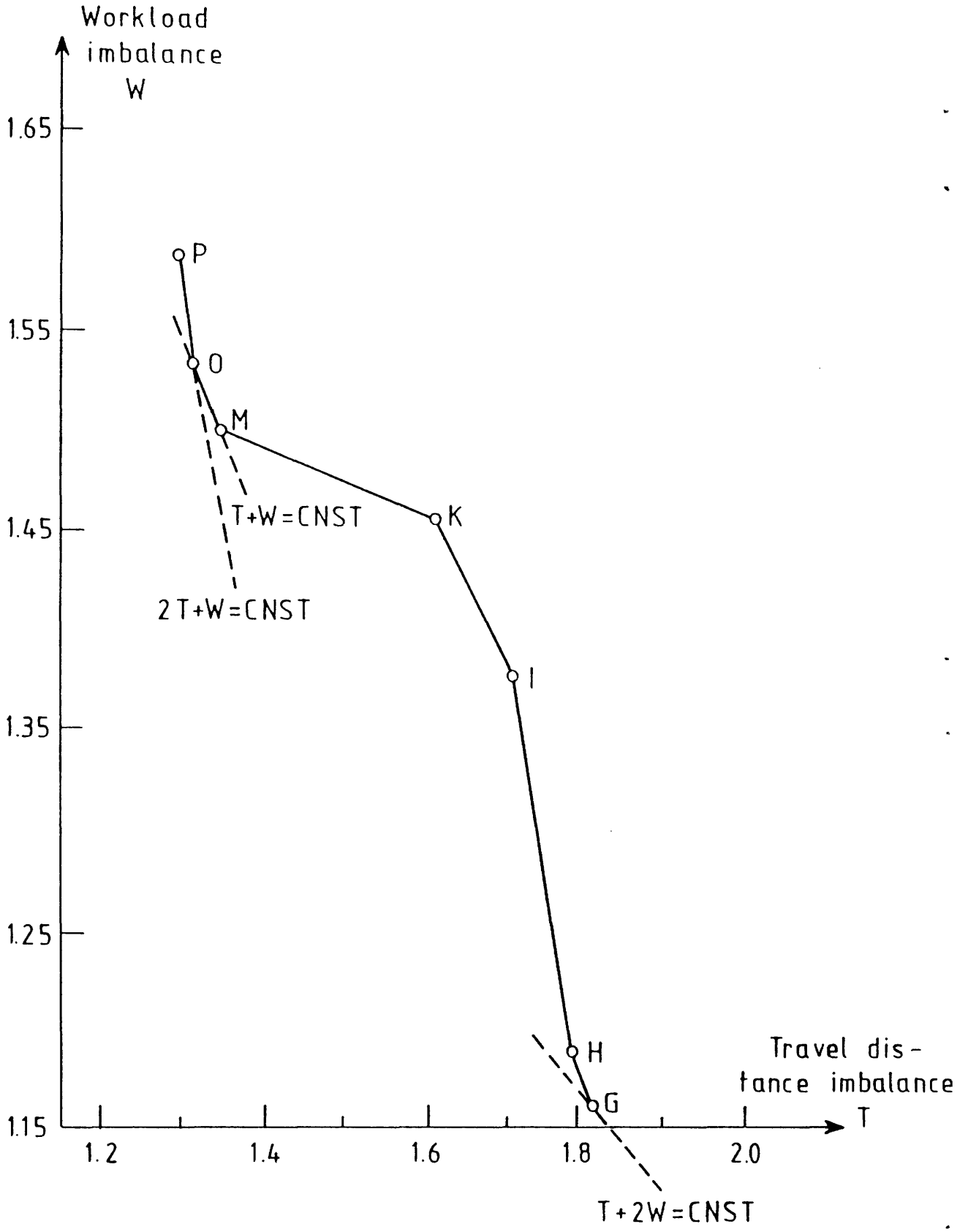


Figure 6
The Efficient Set of Solutions

of workload imbalance is weighted very high, solution P should be chosen. If the two imbalances are weighted equally, solution O is preferred, but solution M is only 0.035% worse. Use of linear combinations of the two imbalances will exclude solutions H, I and K, leaving only solutions G, M, O, and P. For example, if we call the weighting factor on the workload imbalance α, we can easily calculate the limits for α within which a specific solution is preferred if the objective is to minimize $T + \alpha W$, where T and W are the travel time and workload imbalances.

α-range	preferred solution
$0 \leqq \alpha \leqq 0.3019$	P
$0.3019 \leqq \alpha \leqq 1.0303$	O
$1.0303 \leqq \alpha \leqq 1.3655$	M
$1.3655 \leqq \alpha$	G

Table 1. Preferred Solutions for Different α- values

Of course, it is very difficult to estimate the weighting factor to several decimal places, but from Table 1 we can easily find the solution if we use some round α-estimates, as shown in Table 2. This information, if required, is obtainable from the

T-W combination	α	preferred solution
4T + W	0.2500	P
3T + W	0.3333	O
2T + W	0.5000	O
T + W	1.000	O + (M)
T + 2W	2.000	G
T + 3W	3.000	G
T + 4W	4.000	G

Table 2. Preferred Solutions for Some Round α-values

computer. In connection with each solution, the user should be able to retrieve the maps and tables that show the sectoring, as well as all the atom and sector specific results, and then jump to another solution or to the trade-off curve.

In the NORA example, the sector travel times were balanced, resulting in 16 rather arbitrary solutions, as plotted in Figure 5. To add some more points to the curve, we should now be able to balance work load, i.e., to use the procedure described in Section 3.3, starting from solution P.

The shift between objectives may now continue until no more reasonable improvements are found. Finding the efficient set of points and the range for the weighting factor can now be done exactly as shown in the NORA example.

When we include the third objective, equalizing atom travel times, the computer can still eliminate solutions and find the efficient set of points. Appendix C describes an algorithm that does so. However, it will be difficult to visualize the results using a three-dimensional trade-off surface. One approach is to set up three two-dimensional trade-off curves between any two of the three objectives and then illustrate the missing objective by choice of color or a decimal value connected to each point. Figure 7 gives an example of the screen layout, showing the three trade-off curves.

With this visual format we should be able to compare and eliminate solutions, since we can see all the points on the three curves or only the points belonging to the efficient set, as well as the detailed results connected to a particular solution.

Another way to proceed is to abandon the curves and work only with numerical information. Equalizing workload (W), sector-, or atom-travel times (ST or AT) gives a series of solutions that can be stored in the computer; one can then find the efficient set of solutions or points and eliminate the remaining part of the solution, if the user

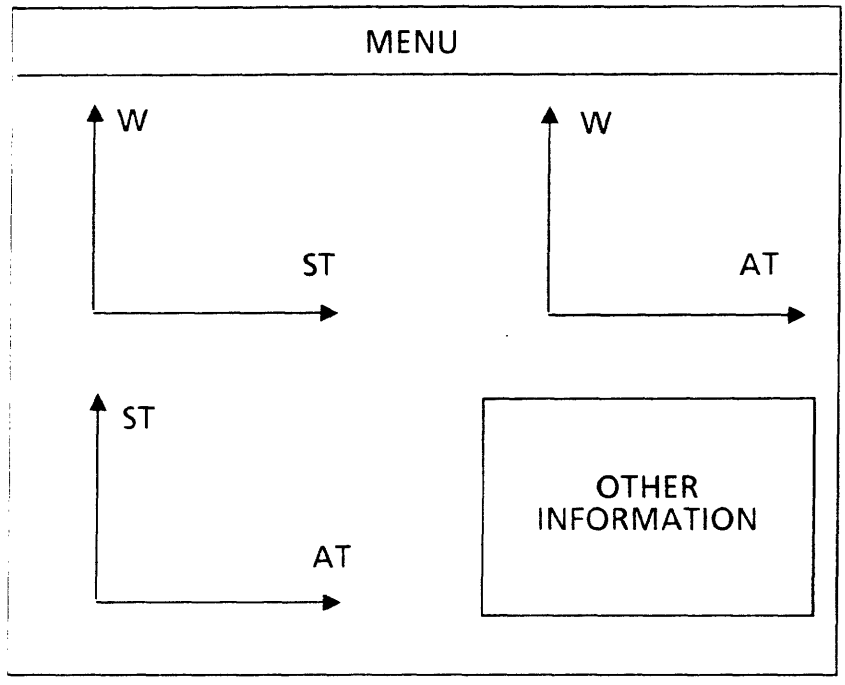


Figure 7. Example of screen layout for a three-dimensional trade-off surface. (ST = sector travel time balance; AT = atom travel time balance).

wishes. In the example shown in Figures 5 and 6, the New Haven example, this approach would decrease the number of solutions considered from 16 to 7. The user may then specify the weight he or she will put on each of the objectives, i.e., specify a_1, a_2, and a_3 in the linear expression $a_1W + a_2ST + a_3AT$. The computer can now present an ordered list either of all the solutions so far, or of all the efficient solutions, or the N_1 best of all solutions, or the N_1 best of the efficient solutions, where N_1 is a user-specified number. The screen will display only the main results in connection with each solution, such as overall mean travel time, workload imbalance, sector travel time imbalance, atom travel time imbalance, and the value of the linear expression just stated. In tabular form, these results are relatively easy to overlook. For example, taking the three most efficient solutions in the New Haven example would give the results shown in Table 3 if the linear expression was $2W + ST + 0AT$. Eventually Table 3 could be expanded to include other useful

information, such as the unit with the highest/lowest workload and the value of the workload, and the sector (atom) with the highest/lowest sector (atom) travel time. However, for ease of

Solution Number	Objective	Workload Imbalance	Sector Travel Time Imbalance	Atom Travel Time Imbalance	Mean Travel Time
G	4.119	1.158	1.803	?	.633
H	4.161	1.189	1.783	?	.634
M	4.336	1.500	1.336	?	.666

Table 3. The three most efficient solutions. Objective: $2W + ST + 0AT = OBJ$.

use, the information per solution should be kept to one or at most two lines. The user can then study the solutions in more detail, since the relevant information is easily accessible, either on the screen or as a printout.

Ordering the solutions may be done by simple complete enumeration, which is fast and easy because of the relatively small number of solutions considered.

3.5 Degrees of Interactivity

The user may now choose to split the work in some fashion with the computer. He or she may perform many of the changes, or may direct the computer to get as near to optimum as possible. We present a scenario in which the user depends heavily on the computer for assistance.

SCENARIO WITH EXTENSIVE COMPUTER ASSISTANCE

Abbreviations:		
	C	Computer
	U	User
	W(E)	Workload (equalizing)
	ST(E)	Sector travel (equalizing)
	AT(E)	Atom travel (equalizing)

Phase 1: Read data

C performs initial sectoring and locating of units

C minimizes mean region wide travel time

Phase 2:

U asks C to perform WE
U asks C to perform STE
U asks C to perform ATE
U asks C to perform WE
} Solutions from all iterations are saved in C

U asks C to find the efficient solutions

U specifies weights on W, ST and AT

C calculates an ordered list of the 5 best solutions

U specifies new weights on W, ST and AT

C calculates a new ordered list of the 5 best solutions

U asks to have 3 solutions printed out with detailed results

End of Scenario

In this situation, the computer performs a relatively large part of the work. The user suggests solutions which are then evaluated by the computer and perhaps included in the efficient set. Figure 8 shows the system from the user's point of view.

3.6 Other Measures of Equality

In the previous sections, we assumed that equality (or fairness, or balance) could be expressed as the ratio between the highest value and the lowest value of the performance measure. Furthermore, we assumed that it would be possible to construct a linear expression aggregating several measures of equality. The arguments for such an approach are that

- the method is simple;
- it is easy to understand and to interpret;
- for narrow ranges of variation, we need only consider the range and not the entire number of observations.

The workloads will usually lie within a narrow range, but the atom- and sector-travel times may vary considerably. In spite of these conditions, we believe that our approach will be useful and understandable in practice.

To illustrate how a different measure of equality and utility may alter preferences, we give a brief example from [16]. Bodily [16,17] aggregated three groups' interests into one nonlinear "quasi-utility" function that could be used in the sector design. In [16], pp. 168-180, he discusses traditional measures of imbalance of which the ratio of best to worst (which is the measure used in this report) is the simplest. All the other imbalance measures deal with all the performance values and not only with the best and worst.

Bodily's example is the same one mentioned in Section 3.4. From questionnaires, he establishes a utility function, Q_T, and depicts it as a function of the emergency response time which, furthermore, is a function of the travel time. Similarly, he arrives at a policeman's utility function Q_W, and defines it as a function of the workload. The total utility function is given by

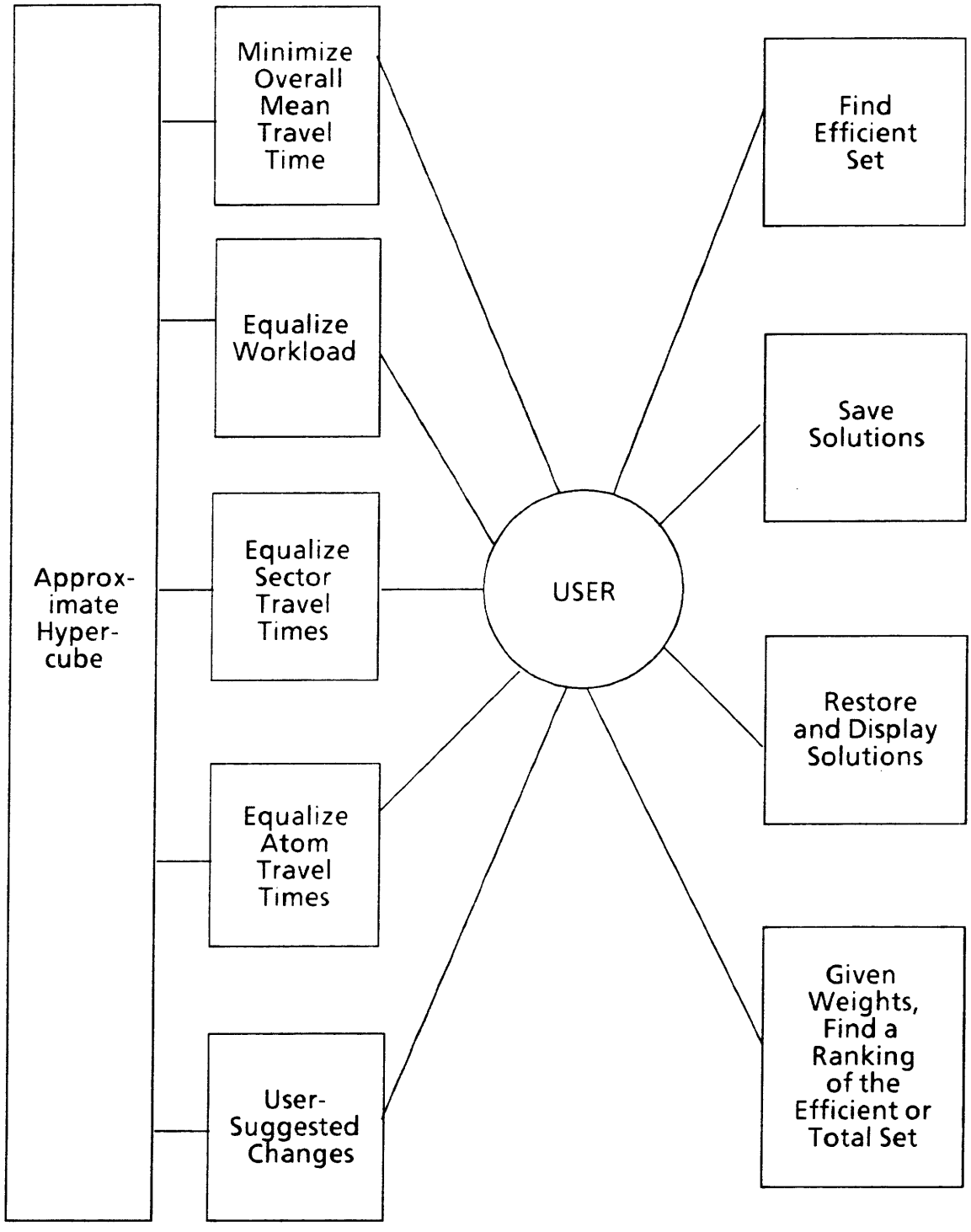


Figure 8. The system from a user's point of view.

$$Q = k_T Q_T + k_W Q_W + (1 - k_T - k_W)\, Q_T Q_W,$$

where k_T and k_W are parameters determining the weight to be put on each part of the utility function. In comparison with the ranking in Figure 5, Bodily achieves the following results in a Q_T Q_w diagram. Solution N was the "worst," preceded by O, P, M, L, K, J, and I. The efficient set not dominated by any other solution is A, B, C, (D), and F, which in our Section 3.4 clearly is made up of solutions dominated by other solutions. The parenthesis around D means that D's belonging to the efficient set depends on the exact shape of the utility function. If k_T and k_W were varied between 0 and 1, only 3 different solutions would be preferable, according to the total utility function Q – namely, solutions A, B, and F. These results show very clearly that the method by which we choose the imbalance measure and the utility functions is very important.

4. Concluding Remarks

We have outlined a two-phase procedure for including the descriptive hypercube queueing model in an optimization framework. The primary objective of the first phase is to minimize the mean region-wide travel time. We did so by means of an iterative procedure that utilizes unit pre-positioning. We found that, in the ambulance service case, the solution is very simple. In the police patrol case, we solved the problem by means of a generalized network flow model. The goals of the second phase are all concerned with equalizing performance measures. We achieved these objectives by means of an interactive local interchanging heuristic. The method generates solutions subject to user-chosen objectives. The solutions may be ranked by the computer, and non-efficient solutions eliminated. The degree of interactivity may be chosen by the user.

Throughout the paper, our emphasis has been on making methods and measures simple and easily understandable, and thus well suited to an interactive environment. Additional details of the procedure, such as stopping rules and number of iterations on different levels, will be determined when further computational results are available.

References

1. Larson, R. C., and Odoni, A.O., Urban Operations Research (Prentice-Hall, New Jersey, 1981).

2. Larson, R. C., "Structural System Models for Locational Decisions: An Example Using the Hypercube Queueing Model," K.B. Haley (ed.); OR'78, (North Holland Publishing Company, Amsterdam, 1979), 1054-1091.

3. Larson, R.C., "A Hypercube Queueing Model for Facility Location and Redistricting in Urban Emergency Services," *Computers and Operations Research,* 1, (1974) 67-95.

4. Jarvis, J. P., "Optimization in Stochastic Service Systems with Distinguishable Servers," Ph.D. thesis, Massachusetts Institute of Technology, June 1975.

5. Lasdon, L. S., Optimization Theory for Large Systems (Macmillan, New York, 1970).

6. Bradley, G., Brown, G., and Graves, G., "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, 24(1), (1977) 1-34.

7. Glover, F., Hultz, J., Klingman, D., and Stutz, J., "Generalized Networks: A Fundamental Computer-Based Planning Tool," *Management Science*, 24(12), (1978) 1209-1220.

8. Brown, G.G., and McBride, R.D, "Solving Generalized Networks," *Management Science*, 30(12), (1984) 1497-1523.

9. Nulty, W.G., and Trick, M.A., "GNO/PC - Generalized Network Optimization Systems," submitted to *OR Letters*.

10. Dantzig, G.B., and Van Slyke, R.M., "Generalized Upper Bounding Techniques for Linear Programming," *Journal of Computer and System Sciences*, 1 (1967) 213-226.

11. Jarvis, J.P., "Approximating the Equilibrium Behavior of Multi-Server Loss Systems," *Management Science*, 31(2), 1985, 235-239.

12. Larson, R.C.,"Computer Program for Computing the Performance of Urban Emergency Service Systems: User Manual," *Technical Report, TR-14-75*, Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, 1975.

13. Chelst, K.R., "Quantitative Models for Police Patrol Deployment," Ph.D. thesis, Massachusetts Institute of Technology, September 1975.

14. Larson, R.C. (ed), *Police Deployment: New Tools for Planners*, D.C. Heath, (ed.), (Lexington Books, Lexington, MA, 1978).

15. Chelst, K.R.,"An Interactive Approach to Police Sector Design," in [14]. 87-102.

16. Bodily, S.E., "Collective Choice with Multidimensional Consequences," Ph.D. thesis, Massachusetts Institute of Technology, February 1976.

17. Bodily, S.E., "Merging the Preferences of Interest Groups for Efficiency and Equality of Service in the Design of Police Sectors," in [14], 103-122.

18. Larson, R.C., "The Hypercube Queueing Model: Illustrative Police Sector Redesign," in [14], 29-74.

Appendix A

RESPONSE PATTERN ALGORITHM (RPA)

```
for k = 1 to N_A do
begin   d5 = λ_k/λ
        d4 = 0
        for j = 1 to N do
        begin   d=1
                for ℓ =1 to j-1 do
                begin d1 = m(k,ℓ)
                      d ← d * ρ(d1)
                end
                d1 = m(k,j)
                d3 = Q(N, ρ, j-1) * d * (1- ρ(d1) ) * d5
                d4 ← d4 + d3
                f(d1,k) = d3
        end
        d7 = (1 - P{S_N} ) * d5
        for n = 1 to N do
        begin f(n,k) ← f(n,k) * d7/d4 end
        f(n,k) ← f(n,k) + d5 * P_Q/N
end
```

Appendix B

CONTIGUITY CHECK ALGORITHM

```
CONTIG: Procedure Options (Main);
   Declare DIMEN FLCAT Binary;
   /* DIMEN: The Number of Atoms */
   Get List (DIMEN);
   Begin;
      Declare A__A__TIG (DIMEN, DIMEN) Fixed Binary (1,0),
                (Label (DIMEN), Scan (DIMEN)) Fixed Binary (4, 0),
                List1 Fixed Decimal;
      /* A__A__TIG: Atom Contiguity Matrix          */
      /* Scan (K) = 1: Node K is Scanned            */
      /* Label (J) = 1; Node J is Labeled           */
      /* List1: Number of Scanned Nodes Plus One    */
      Get List (A__A__TIG);
      Scan = 0;
      Scan (1) = 1;    Label (1) = 1;    List1 = 2;
      Do K = 1 to DIMEN;
         If Scan (K) = 0, Then , Do;
            Put List ('Sector is Not Contiguous');
            Go to Finish;
            END;
         I = Scan (K);
         Do J = 1 to DIMEN;
            If A__A__TIG (J, J) = 1   &   Label (J) ≠ 1,   Then, Do;
               Scan (List1) = J;
               List1 = List1 + 1;
               If List1 > DIMEN + .5, Then Go to FINE;
               Label (J) = 1;
               End;
            End; /* Loop on J */
         End;   /* Loop on K */
      FINE: Put List ('Sector is Contiguous');
      End; /* Begin Block */
Finish: End CONTIG;
```

Appendix C

ALGORITHM TO FIND THE EFFICIENT SET

N Points

EFF(j) = 0 : Point j not yet examined

EFF(j) = 1 : Point j belongs to the efficient set

EFF(j) = 2 : Point j does not belong to the efficient set

```
for i = 1 to N do begin EFF(i) = 0 end
for i = 1 to N do
begin for j = 1 to N do
      begin if j = i or EFF(j) = 2 then go to L1
            if point j dominates* point i then
            begin EFF(i) = 2
                  go to L2**
            end
         L1 : continue
      end j-loop
      EFF(i) = 1
   L2 : continue
end i-loop
```

* a point dominates if all the objectives have the same or a better value. If a tie occurs, both solutions should be declared efficient.

** It may not be allowed to jump out of a loop.

After this procedure has been run, the dominated points (EFF = 2) may be removed from the point set and the computer memory.

Appendix D

Here we list the frequently used symbols for the hypercube model.

λ	System-wide average arrival rate of (potential) customers
ρ	System-wide average utilization factor (i.e., average fraction of time servers are busy)
N	Numbers of servers or units in the system
η	System-wide average arrival rate of (potential) customers divided by total available system-wide service rate ($=\rho$ for infinite-capacity unsaturated system)
N_A	Total number of geographical atoms or cells (or, in network terms, nodes or vertices)
$\overline{T}$	System-wide average travel time
P_Q	Steady-state probability that a queue of positive length exists
P'_Q	Steady-state probability that a randomly arriving customer incurs a positive queue delay before entering service (= steady-state probability that all N servers are simultaneously busy)
$P\{S_i\}$	Steady-state probability that a particular M/M/N model is in state $S_i \equiv$ "i customers in the system"
μ^{-1}	Average service time
μ_n^{-1}	Average total service time of server (unit) n
μ_{nj}^{-1}	Average service time for unit n to service a request in atom j
ρ_n	Average utilization factor of server (unit) n
f_{nj}	Fraction of answered service requests that take server n to geographical atom j
$f_{nj}^{[1]}$	Fraction of all answered service requests that send unit n to atom j and incur *no* queue delay
$f_{nj}^{[2]}$	Fraction of all answered service requests that send unit n to atom j and incur a positive queue dealy
λ_j	Average arrival rate of (potential) customers from atom j
τ_{ij}	Mean travel time from atom i to atom j
t_{nj}	Mean time for unit n, when available, to travel to atom j

B	A system state $\{b_N, b_{N-1}, ..., b_1\}$
P_{B_k}	Steady-state probability that the system is in state B_k
ℓ_{nj}	Probability that server n is located in atom j while available or idle
$Q(N, p, j)$	Correction factor for N-server, infinite-capacity system, when we attempt to assign the $(j+1)$st preferred server
R_n^F	Rate at which server n, when free or available, is assigned to customers
λ_D	Time average rate at which a server is assigned to customers who have incurred a positive queue delay
G_n^j	Set of geographical atoms for which server n is the jth preferred dispatch alternative
m_{aj}	Identification number of the jth preferred server for atom a
ε	Tolerance, a small nonnegative constant
Γ	Normalization factor
Γ_k	Normalization factor
X	Minimum target level for preventive control