Mathematical Programming 6 (1974) 62–88. North-Holland Publishing Company

VALIDATION OF SUBGRADIENT OPTIMIZATION

Michael HELD
IBM Systems Research Institute, New York, U.S.A.

Philip WOLFE, Harlan P. CROWDER
IBM Watson Research Center, Yorktown Heights, New York, U.S.A.

Received 6 August 1973
Revised manuscript received 26 November 1973

The "relaxation" procedure introduced by Held and Karp for approximately solving a large linear programming problem related to the traveling-salesman problem is refined and studied experimentally on several classes of specially structured large-scale linear programming problems, and results on the use of the procedure for obtaining exact solutions are given. It is concluded that the method shows promise for large-scale linear programming

1. Introduction

A wide variety of mathematical programming problems can be given the form

$$\text{maximize } w(\pi), \ \ \pi \in S\,, \tag{1.1}$$

where

$$w(\pi) = \min\{c_k + \pi \cdot v_k \colon k = 1, ..., K\}\ , \tag{1.2}$$

c_k is a scalar, $v_k = (v_{k1}, ..., v_{kn})$ is a real n-vector for all k, and S is a closed convex subset of E^n. ($\pi \cdot v_k$ above denotes the inner product.) If K is finite and S is a polyhedron, then the problem is alinear programming problem; otherwise it is the general problem of maximizing a concave function over S, for any concave function has such a representation [22, p. 102]. This paper reports some experiments testing an unusual and surprisingly successful procedure for the numerical solution of certain problems of this kind. We call this procedure a *subgradient method.*

If the constraint set S can be specified by a simple set of linear inequalities (as it is in the problems we consider) and K is small, then our problem is just an ordinary linear programming problem, and we think it is best solved by the simplex method for linear programming. In the problems we are concerned with, however, K is vast-finite, but easily of the order of $n!$ with n around 100. We therefore make no effort to write out the data c_k, v_k explicitly for all k, but rather suppose them so defined that, *given* π, the task of then finding c_k, v_k for which the minimum of (1.2) is assumed is reasonably convenient. That task, called "column generation", is central to most schemes for the solution of very large linear programming problems (a good account of which is given by Lasdon [15, Chapter 4]). As we have discussed elsewhere [24], all problems amenable to treatment by the Dantzig–Wolfe decomposition procedure can be cast in the form (1.1), (1.2).

A first encounter with the kind of method reported here was described by Held and Karp [11] in their work on the traveling-salesman problem, for which they devised a function w of the type (1.2) constituting a kind of "pseudo-dual" for that problem; its maximum value provided excellent bounds for their branch-and-bound algorithm. Having tried both a steepest-ascent procedure and a simplex method using column generation for maximizing w, and finding them dishearteningly slow, they invented a "subgradient" method which turned out to be highly effective. Subsequently, A.J. Hoffman pointed out that their procedure could be viewed as an application of a method discussed by Agmon [2] and Motzkin and Schoenberg [17] for the solution of linear inequality systems, and Held and Karp presented the procedure from that point of view. We have recently found a sizable body of Russian literature dealing with the theory of subgradient methods, which apparently originated with Shor [23], and has perhaps been best expounded by Poljak [20, 21]; we recommend his work for a discussion of the algorithm in more general spaces and for theoretical results on the rate of convergence.

Despite the mathematical interest of the Agmon–Motzkin–Schoenberg work and that of the Russians, there appears to be little computational experience with the method beyond that reported by Held and Karp. References in the Soviet literature to computational experience are at best cryptic. For example, quoting in full one of the two such references we have found [21]: "The effectiveness of the methods described here has also been confirmed by practical experience in their application".

In the next section we present the subgradient method as we have been using it. The next three sections give the formulations and computational results for the three types of problems we have tried: the assignment problem, the "noninteger" traveling salesman problem, and a multicommodity network flow problem. In each case the method proved to be effective: convergence seemed fairly rapid. We have not tried to compare it with other methods for overall efficiency as measured, say, by operation counts or running time. Our aim was simply to determine whether subgradient optimization would work at all on some readily accessible problems.

For all of the problems we were able, in one way or another, to establish the provably optimal solutions. In Section 6 we show how this was done, and give some general results connecting the approximate solutions the subgradient procedure obtains with the exact solutions. The last section presents what we think our computational experience adds up to, and what we hope it portends for other kinds of important mathematical programming problems.

2. The subgradient method

Let us first take $S = \mathrm{E}^n$, so the problem (1.1) is just

$$\max w(\pi)\,, \tag{2.1}$$

where, as before,

$$w(\pi) = \min\{c_k + \pi\cdot v_k\colon k = 1, ..., K\}\,. \tag{2.2}$$

Throughout this paper we suppose w to be bounded above. Since it is piecewise linear, there then exists at least one point π^* such that $w(\pi^*) = w^* = \max w$.

For any π, the minimum of (2.2) is assumed for at least one value of the index k. Denote by $V(\pi)$ the set of all n-vectors v_k such that the minimum is assumed for the index k:

$$V(\pi) = \{v_k\colon c_k + \pi\cdot v_k = w(\pi),\ k = 1, ..., K\}\,. \tag{2.3}$$

$V(\pi)$ will usually be a singleton; when it is, the function w is differentiable at π, and the single member of $V(\pi)$ is the gradient $\nabla w(\pi)$. If $V(\pi)$ is not a singleton, then w is not differentiable at π; but the notion of differentiability has been extended to give us what we need. (The material of the next paragraph is taken from [22, Section 23], with the appropriate changes made for concave, rather than convex, functions.)

The n-vector u is a *subgradient at* $x \in \mathrm{E}^n$ of the concave function f on E^n if

$$f(y) - f(x) \leq u \cdot (y - x) \tag{2.4}$$

for all $y \in \mathrm{E}^n$. The set of all subgradients at x is the compact, convex set $\partial f(x)$ called the *subdifferential.* The function f has the directional derivative

$$f'(x;d) = \lim_{t \to 0^+} \frac{f(x + td) - f(x)}{t}$$

for any x, $d \in \mathrm{E}^n$, and

$$f'(x;d) = \min\{d \cdot u\colon u \in \partial f(x)\} .$$

A necessary and sufficient condition that x maximize f is that $f'(x;d) \leq 0$ for all d, or equivalently that $0 \in \partial f(x)$. For the concave function w of (2.2), $\partial w(\pi)$ is the convex hull of $V(\pi)$, and

$$w'(\pi;d) = \min\{d \cdot v\colon v \in V(\pi)\} = \min\{d \cdot v_k\colon c_k + \pi \cdot v_k = w(\pi)\} .$$

Fig. 1 sketches contours of the function

$$w(\pi) = \min\{-\pi_1, \pi_1 - 2\pi_2, \pi_1 + 2\pi_2\} .$$

At the point $(2, 1\frac{1}{4})$, $V(\pi)$ is the singleton $\{(-1, 0)\}$; at any point on the ray $\pi_1 < 0$, $\pi_2 = 0$ it is the doubleton $\{(1, 2), (1, -2)\}$; at the origin, which maximizes w, it has three members. The function fails to be differentiable only on three rays issuing from the origin – but that is just where we want to be, so we cannot escape the lack of differentiability.

It is certainly possible to use ∂w to define a direction of steepest ascent for any point which does not solve the problem, and develop a close analogue of the method of steepest ascent for our problem; several such proposals have been made [3, 5, 8, 9]. We take the view, however, that finding the entire set $V(\pi)$ in order to get a locally "best" direction

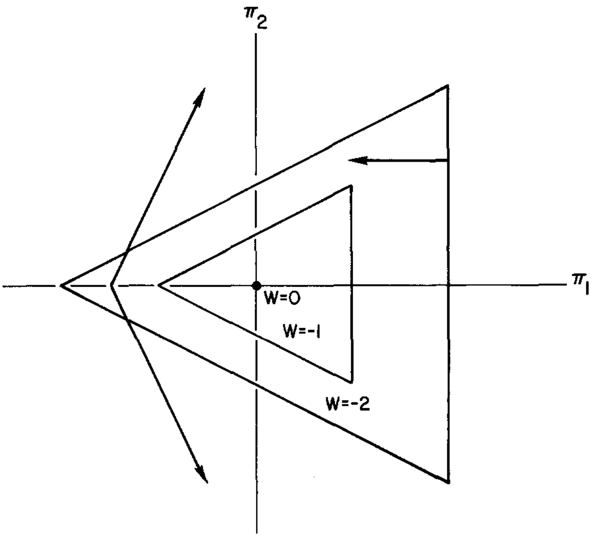

Fig. 1.

is too heavy a computational requirement for the problems of interest to us, and we suppose that, for any π, we obtain only a unique

$$v(\pi) = v_k \in V(\pi) \tag{2.5}$$

as a result of the evaluation (2.2) of $w(\pi)$. This restriction has the result that at a point like $\pi = (1, 1)$ in the above example we cannot obtain a direction of ascent; $w'(\pi; d) < 0$ for each $d \in V(\pi)$. We will, courageously, use the direction $v(\pi)$ nevertheless.

Another respect in which the algorithm below differs from the usual steepest ascent prescription is in making no effort to maximize w in the chosen direction. First, because that would entail a very heavy computational burden; secondly, because the method of choice of direction hardly makes that advisable. Our choice of step length depends only very modestly on the behavior of the function.

The subgradient algorithm

Given $\pi^0 \in E^n$ and the sequence $\{t_j\}$ of positive scalars, called *step sizes,* define the sequence $\{\pi^j\}$ by

$$\pi^{j+1} = \pi^j + t_j v(\pi^j) \quad \text{for } j = 0, 1, \dots . \tag{2.6}$$

Some theoretical results relating to an appropriate choice of step size for the algorithm have been given. The most general result [20] is that $w(\pi^j)$ will converge to its maximum w^* under merely the conditions

$$t^j \to 0, \quad \sum_{j=0}^{\infty} t_j = \infty . \tag{2.7}$$

Alternatively, using the choice

$$t_j = \lambda_j \frac{\hat{w} - w(\pi^j)}{|v(\pi^j)|^2} \tag{2.8}$$

(where $|\cdot|$ denotes the Euclidean norm) with $\hat{w} < \max w(\pi)$, and for each j, $\epsilon < \lambda_j \leq 2$ for some fixed $\epsilon > 0$, the sequence $w(\pi^j)$ either converges to $\hat{w}$, or a point π^j is obtained such that $w(\pi^j) \geq \hat{w}$ (cf. [11, 21]). This choice is almost a direct application of the Agmon–Motzkin–Schoenberg procedure for the determination of a point π such that $w(\pi) \geq \hat{w}$, that is, a point satisfying the inequalities

$$\pi \cdot v_k \geq \hat{w} - c_k , \qquad k = 1, ..., K \tag{2.9}$$

The main idea of the proofs of convergence is this: Let π^* be any point in the optimal set, i.e., $w(\pi^*) = w^*$. Since $v(\pi^j) \in \partial w(\pi^j)$, relation (2.4) says

$$w^* - w(\pi^j) \leq v(\pi^j) \cdot (\pi^* - \pi^j) , \tag{2.10}$$

so that if π^j is not optimal, then the direction $v(\pi^j)$ makes an acute angle with the ray from π^j through π^*. Consequently, if t_j were sufficiently small, the point π^{j+1} would be closer to π^* than was π^j. Thus the sequence approaches the optimal set, although the objective values $w(\pi^j)$ are not monotonic.

There seems to have been no theoretical study of the rate of convergence of the general algorithm determined by (2.7), but Agmon [2] has shown linear convergence of his procedure to a solution of the inequalities (2.9). Poljak [21] has shown that if (2.8) is used with the underestimate $\hat{w}$ replaced by w^*, the (unknown) maximum value of w, then geometric convergence obtains. Oettli [19] has shown geometric convergence of an iterative method based upon subgradient directions for simultaneously approximating both the primal and dual solutions by solving a system of inequalities.

Our practical work has been guided by (2.8); however, an overestimate $\bar{w}$ of the maximum of w was used instead of the recommended underestimate $\hat{w}$, which would be hard to find. We observed that the results did not seem to depend critically on the exact value of $\bar{w}$. Of course, it is necessary that $t_j \to 0$, but (2.8) will not accomplish this if used with an overestimate $\bar{w}$ unless we choose a sequence λ_j which tends toward zero. We have tried a number of ways to choose such a sequence and the results are reported on in Sections 3, 4 and 5. Generally (but not always) a good rule is to set $\lambda = 2$ for $2n$ iterations (where n is a measure of the problem size), and then successively halve both the value of λ and the number of iterations until the number of iterations reaches some threshold value z; λ is then halved every z iterations until the resulting t_j is sufficiently small. We note that this procedure violates the condition $\Sigma t_j = \infty$. It is thus possible to converge to a point not in the cptimal set, although in our work that almost never happened. We would particularly point out choice of step size as an area which is imperfectly understood.

In many cases we found that the algorithm terminated in a *finite* number of iterations with an optimum solution (cf. Sections 3 and 5). This seems to occur whenever the optimal set $\{\pi\colon w(\pi) = w^*\}$ is of full dimension. Since π^j approaches the optimal set and does not stop moving until $v(\pi^j) = 0$, and $v(\pi) = 0$ is the unique subgradient in the interior of the optimal set finite termination is then quite likely; to avoid it you would have to be unlucky enough to hit the boundary of the optimal set infinitely often!

When the constraint set S is a proper subset of E^n, the step (2.6) should be modified to

$$\pi^{j+1} = P_S(\pi^j + t_j v(\pi^j)) \,, \tag{2.11}$$

where P_S is the operator projecting E^n onto S, that is, for any $x \in \mathrm{E}^n$, the point $P_S(x)$ is the unique point of S nearest x. Using the convexity of S it is easy to show that when the point π^{j+1} given by (2.6) is closer to a solution π^* than is π^j, the same is true of the point given by (2.11), and that the convergence results given above hold equally well here. The problem studied in Section 5 requires such a projection. Note that here, as is obvious for the unconstrained problem, if $\pi^{j+1} = \pi^j$, then π^j is optimal, for if $v(\pi^j) \neq 0$, then the hyperplane $c_j + v(\pi^j)\cdot\pi = w(\pi^j)$ separates S from the set $\{\pi\colon w(\pi) \geq w(\pi^j)\}$, and those sets meet at π_j.

3. The assignment problem

Given n jobs, n men, and the n-by-n matrix $A = \{a_{ij}\}$ giving the cost for which man i will do job j, the assignment problem is that of finding a one-to-one assignment of men to jobs that will minimize the total cost. A solution is evidently that set of integers x_{ir} achieving

$$\min \left\{ \sum_{i,r} a_{ir} x_{ir} : \sum_i x_{ir} = 1,\ \sum_r x_{ir} = 1,\ \text{all } x_{ir} \geq 0 \right\}. \tag{3.1}$$

It is well known [18] that among the solutions to the above linear programming problem without the integer requirement will be found one that is integral, so the optimal value sought is also that of the dual of the linear program, namely

$$\max \left\{ \sum_i \pi_i + \sum_r \rho_r : \pi_i + \rho_r \leq a_{ir} \right\} \quad \text{for all } i, r. \tag{3.2}$$

Given π, the ρ which maximizes the dual objective is evidently given by $\rho_r = \min_s [a_{sr} - \pi_s]$, so the dual objective may be written

$$w(\pi) = \sum_i \pi_i + \sum_r \min_s [a_{sr} - \pi_s] , \tag{3.3}$$

a piecewise linear concave function. In order to exhibit w in the form (2.2), define an *assignment* as a function from $1, ..., n$ to $1, ..., n$: $A(r)$ is the man assigned to job r (at this point we do not require that each job be assigned to a different man). Enumerating all possible assignments $A_1, A_2, ..., A_K$ $(K = n^n)$, set the cost of an assignment

$$c_k = \sum_{r=1}^{n} a_{A_k(r)r} ,$$

and

$$(v_k)_i = 1 - [\text{number of indices } r \text{ for which } A_k(r) = i] . \tag{3.4}$$

Then (2.2) holds.

The iterative step (2.6) from π to $\pi + t\,v(\pi)$ has this description: Given π, assign each job r to a man i who will do it cheapest, as measured by the modified costs $a_{ir} - \pi_i$. Then $1 - v_i(\pi)$ is the number of jobs to which man i has been assigned. If he has been assigned to no job, then π_i is increased; if he has been assigned to more than one job, then π_i is decreased. If $v(\pi^j) = 0$, then A_k assigns exactly one man to each job. A_k then solves the assignment problem, π^j solves its dual, and the iteration terminates.

In our first experiments, using problems of order $n \leq 30$ and randomly generated a_{ij} (uniformly distributed over 1, 2, ..., 200), we were startled to find that the procedure usually terminated in $v(\pi^j) = 0$. The explanation lay in the fact that such problems usually have a unique optimal assignment. The theorem below shows that then the set of optimal π is of full dimension, so that, as argued in Section 2, termination is nearly certain.

Theorem 3.1. *If an optimal assignment problem of order n has a unique solution, then the optimal set for its associated problem* (3.3) *is of dimension n.*

Proof. By the strict complementary slackness theorem, there exist $\bar{x}$ optimal for (3.1) and $\bar{x}$, $\bar{p}$ optimal for (3.2) such that $\bar{x}_{ir} = 0$ implies $\bar{\pi}_i + \bar{p}_r < a_{ir}$ for all i, r. If the assignment problem has the unique optimal assignment A, then (3.1) has the unique solution $\bar{x}_{ir} = \delta_{i,A(r)}$, so that

$$\text{for each } r,\ \min_s [a_{sr} - \bar{\pi}_s] \text{ is assumed uniquely at } s = A(r)\,. \tag{3.5}$$

Let Π be the set of all π such that

$$a_{ir} - \pi_i > a_{A(r),r} - \pi_{A(r)} \quad \text{for all } r, i \neq r\,. \tag{3.6}$$

By (3.4), $v(\pi) = 0$ for $\pi \in \Pi$. Π is clearly convex and open, and a subset of the optimal set (indeed, it is easy to show that it is precisely the interior of the optimal set). Since (3.5) shows Π to be nonempty, the theorem is proven.

On the other hand, suppose that the optimal assignment is not unique. To simplify a rather complicated discussion, suppose that there are just two optimal assignments, assigning men 1 and 2 indifferently to jobs 1 and 2, so that $a_{11} + a_{22} = a_{21} + a_{12}$. When π is sufficiently close to the optimal set, $\min[a_{sr} - \pi_s]$ will be assumed for $s = 1$ or 2 when $r = 1$ or 2. Indeed, for both $r = 1$ and $r = 2$, the minimum is assumed for $s = 1$ if $\pi_2 - \pi_1 < a_{21} - a_{11} = \Delta$, and for $s = 2$ if $\pi_2 - \pi_1 > \Delta$. Thus $v(\pi)$ has either the form $(-1, 1, ...)$ or $(1, -1, ...)$ in these cases, and also when $\pi_2 - \pi_1 = \Delta$ if ties are settled in the same way for $r = 1$ as for $r = 2$.

The procedure drives the sequence π^j toward approximate satisfaction of $\pi_2 - \pi_1 = \Delta$, but $v(\pi^j) = 0$ never happens. The examples reported below do not have unique optimal assignments, and $v(\pi^j) = 0$ was never encountered.

There are many ways to perturb the data of an assignment problem so that the perturbed problem has a unique solution which is an optimal assignment for the unperturbed problem, but even if there were no better way to solve an assignment problem (and we think there is [7, Chapter III]), we would not recommend that approach to ensure termination. The general approach to determining optimality given in Section 6 has worked quite well in our tests and, we think, terminates the algorithm in fewer iterations.

We tested the subgradient method on three large symmetric assignment problems. These problems were obtained by treating certain traveling-salesman problems, the data for which is in the literature, as assignment problems. Problem A42 was derived from the 42-city problem of Dantzig, Fulkerson and Johnson [4]; problem A48 from the 48-city problem of Held and Karp [12]; and problem A57 from the 57-city problem of Karp and Thompson [13]. (The only alteration consisted in changing the diagonal entries from zero to a number so large that no diagonal entry appeared in an assignment.)

The results are summarized in Table 1. There $\bar{w}$ is the upper estimate used in (2.8) to compute the step size, and w^*, as before, is the optimal value. Figs. 3.1 and 3.2 plot, every ten iterations, the least (symbol □) and greatest (symbol ●) value found during the previous ten iterations of the quantities $w(\pi^j)$, $\log_{10}(w^* - w(\pi^j))$, for the problem A42. The last plot clearly shows the linear convergence of $w(\pi^j)$ to w^*. It should be noted that linear convergence is virtually assured by our step-size choices, provided convergence happens at all.

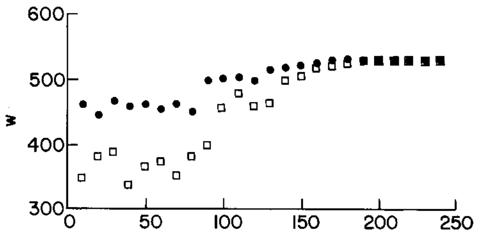

Fig. 3.1. (Problem A42).

Table 1
Assignment problems

	λ	iterations	best w
Problem A42	2	84	454.00
$\bar{w}$ = 692	1	42	470.05
w^* = 532	1/2	21	512.01
	1/4	10	516.75
	1/8	5	526.78
		162	
	(then $\lambda \leftarrow \lambda/2$ every 5 iterations; w = 531.994 at iteration 252)		
Problem A48	1	96	9069.14
$\bar{w}$ = 12196	1/2	48	9413.79
w^* = 9870	1/4	24	9738.26
	1/8	12	9795.85
	1/16	6	9832.83
	1/32	5	9831.57
		191	
	(then $\lambda \leftarrow \lambda/2$ every 5 iterations; w = 9869.91 at iteration 307)		
Problem A57	1	114	9185.62
$\bar{w}$ = 13696	1/2	57	10121.73
w^* = 10553	1/4	28	10345.09
	1/8	14	10427.58
	1/16	10	10520.43
		223	
	(then $\lambda \leftarrow \lambda/2$ every 10 iterations; w = 10552.91 at iteration 405)		

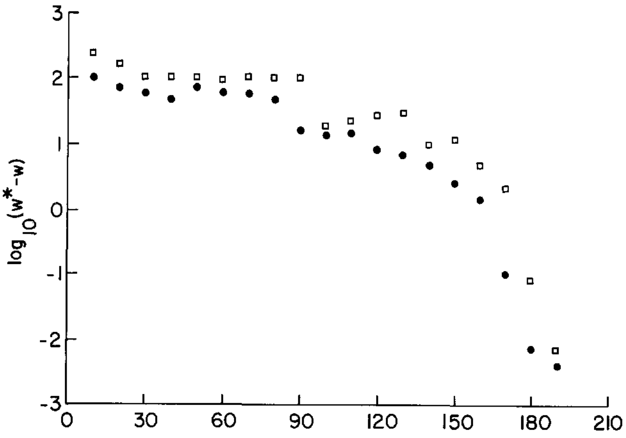

Fig. 3.2. (Problem A42).

4. The non-integer symmetric traveling-salesman problem

Let the $n \times n$ symmetric matrix (c_{ij}) specify the weights assigned to the edges of the complete undirected graph with vertex set $\{1, 2, ..., n\}$. Any subgraph G may be represented by 0-1 variables x_{ij}, $i = 1, 2, ..., j-1$, $j = 2, 3, ..., n$, where $x_{ij} = 1$ if the arc (ij) is present, and $x_{ij} = 0$ otherwise. The weight of any subgraph G is $\Sigma_{i,j}\, c_{ij} x_{ij}$. The traveling-salesman problem seeks a *tour* (a cycle passing through each vertex exactly once) of minimum weight. Dantzig, Fulkerson and Johnson [4] have given an integer linear programming formulation for this problem involving a combinatorial number $O(2^{n-1})$ of constraints. By shrewd choice of some of those, and ad hoc combinatorial arguments, they found the first known solution of the famous "42-city" traveling-salesman problem. Held and Karp [10, 11] considered the same formulation with the 0-1 constraints deleted and replaced by $0 \leq x_{ij} \leq 1$. The resulting program is an ordinary linear program (still, of course, with $O(2^{n-1})$ constraints) the solution of which provides lower bounds on the solution of the traveling-salesman problem. They showed that this linear program could be cast in the form (2.1), (2.2): Define a *1-tree* as a tree (a cycleless connected subgraph spanning the vertex set $\{2, 3, ..., n\}$, together with two distinct edges incident on vertex 1. If the 1-trees are numbered $1, 2, ..., K$ $(= \frac{1}{2}(n-2)(n-1)^{n-2})$, then in (2.2), c_k is the weight of the k^{th} 1-tree and $(v_k)_i = d_{ki} - 2$, where d_{ki} is the degree of vertex i in 1-tree k.

The results given by Held and Karp [11] include the three traveling-salesman problems we report on here. They used the subgradient algorithm simply with $t_j = 1$, apparently not a bad choice for these problems, since the number of steps they required to reach a reasonable value of w was as low as twice that taken by the present method.

Our results, using the rules given in Section 2, are summarized in Table 2 and Figs. 4.1, 4.2. The data for the three problems, T42, T48 and T57, are those for the assignment problems A42, A48 and A57, cited in Section 3, and the displays are constructed in the same way.

5. The multicommodity maximum flow problem

A directed network consists of a set $N = \{1, 2, ..., n\}$ of nodes, a subset A of ordered pairs (i, j) of elements taken from N, called the *arcs* of the network, and non-negative numbers c_{ij} associated with each

Table 2
Traveling-salesman problems

	λ	iterations	best w
Problem T42	2	84	642.62
$\bar{w}$ = 720	1	42	689.35
w^* = 697	1/2	21	693.95
	1/4	10	695.22
	1/8	5	696.23
		162	
	(then $\lambda \leftarrow \lambda/2$ every 5 iterations; w = 696.99 at iteration 230)		
Problem T48	2	96	9599.14
$\bar{w}$ = 12363	1	48	11143.48
w^* = 11444.5	1/2	24	11299.14
	1/4	12	11375.72
	1/8	6	11410.43
		186	
	(then $\lambda \leftarrow \lambda/2$ every 6 iterations; w = 11443.2539 at iteration 282)		
Problem T57	2	114	12505.78
$\bar{w}$ = 13151	1	57	12789.22
w^* = 12907.5	1/2	28	12876.62
	1/4	14	12898.65
	1/8	7	12899.14
		220	
	(then $\lambda \leftarrow \lambda/2$ every 7 iterations; w = 12907.4297 at iteration 304)		

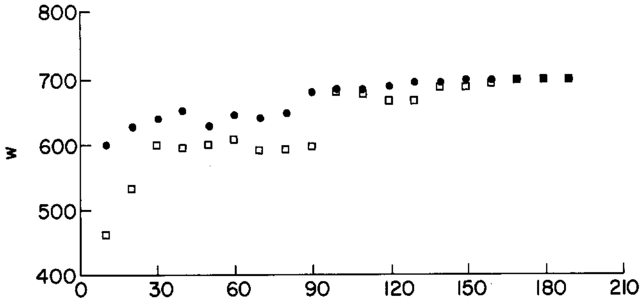

Fig. 4.1. (Problem T42).

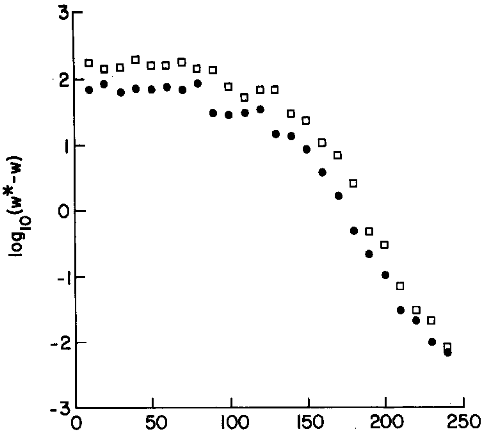

Fig. 4.2. (Problem T42).

arc (i, j), called the *capacity* of arc (i, j). There are $2D$ special nodes in the network: $s^1, s^2, ..., s^D$ and $t^1, t^2, ..., t^D$. The node s^d is called the *source* for commodity d, and t^d is called the *sink* for commodity d. It is not required that these $2D$ special nodes be distinct except that $s^d \neq t^d$ for each $d = 1, ..., D$.

A set of non-negative numbers x_{ij} is called a *simultaneous flow* of commodities if the following are satisfied:

$$\text{for every } (i, j) \in A, \quad \sum_{d=1}^{D} x_{ij}^d \leq c_{ij} \, , \tag{5.1}$$

and for every node i, and for every commodity $d = 1, 2, ..., D$,

$$\sum_{\{j:(i,j)\in A\}} x_{ij}^d - \sum_{\{j:(j,i)\in A\}} x_{ji}^d = \begin{cases} v^d & \text{if } i = s^d \\ 0 & \text{if } i \neq s^d, t^d \\ -v^d & \text{If } i = t^d \ . \end{cases} \tag{5.2}$$

For each arc (i, j) and for each d, x_{ij}^d represents the amount of flow of commodity d in arc (i, j). The constraints (5.1) express the fact that the different commodities share a scarce resource, arc capacity. We call v^d the *value* of the flow of commodity d. The multicommodity maximum flow problem seeks a simultaneous flow which maximizes $\Sigma_{d=1}^D v^d$. This particular statement of the problem is sometimes called the *node-arc* formulation [7].

In order to exhibit the multicommodity maximum flow problem in the form (1.1), (1.2), we apply a *resource-allocation* decomposition procedure [1]. In this approach the "scarce resources", arc capacities, are allocated among the competing commodities. Let the network have R arcs and let c_r denote the capacity of arc r. Letting π_{rd} be the amount of the capacity of arc r dedicated to the flow of commodity d, we require

$$\begin{cases} \sum_{d=1}^{D} \pi_{rd} = c_r & \text{for all } r, \\ \pi_{rd} \geq 0 & \text{for all } r, d. \end{cases} \tag{5.3}$$

Any RD dimensional vector $\pi = \{\pi_{rd}\}$ satisfying (5.3) is called an *allocation.* Clearly, given any allocation, the multicommodity flow problem decouples into D single-commodity flow problems: Let $F_d(\pi)$ be the maximum flow value for commodity d, given the allocation π, and let

$$w(\pi) = \sum_{d=1}^{D} F_d(\pi) . \tag{5.4}$$

Then we seek an allocation π which maximizes $w(\pi)$.

We proceed to show how (5.4) can be written in the form (1.2). A *cut* q_d for commodity d is a set of arcs of the network whose deletion breaks all paths from s^d to t^d. Relative to the allocation π, the capacity included in q_d is $\Sigma_{r\in q_d} \pi_{rd}$. Letting Q_d be the set of all cuts for commodity d, the max-flow, min-cut theorem for single commodity flows [7] asserts that

$$F_d(\pi) = \min_{q_d \in Q_d} \sum_{r \in q_d} \pi_{rd} ,$$

so that

$$w(\pi) = \min_{q_1 \in Q_1, \ldots, q_D \in Q_D} \sum_{d} \sum_{r \in q_d} \pi_{rd} .$$

Each selection $(q_1, \ldots, q_D)$ of cuts from $Q_1 \times \ldots \times Q_D$ defines an RD-dimensional vector $v = \{v_{rd}\}$, where $v_{rd} = 1$ if $r \in q_d$, and is zero otherwise; we suppose the (probably astronomically large) set of all such v

[1] This was suggested to us by R.M. Karp. Subsequently we found G.B. Dantzig had proposed this approach [1].

numbered $v_1, ..., v_K$ in any fashion. The multicommodity flow problem becomes

$$\max_{\pi} w(\pi) ,$$

where

$$w(\pi) = \min\{\pi \cdot v_k : k = 1, ..., K\} \tag{5.5}$$

and

$$\left\{\begin{array}{ll} \sum_{d=1}^{D} \pi_{rd} = c_r & (r = 1, 2, ..., R) \\ \pi_{rd} \geq 0 & (r = 1, 2, ..., R; d = 1, 2, ..., D). \end{array}\right. \tag{5.6}$$

This problem is clearly in the form (1.1), (1.2).

Because of the simple nature of the constraints (5.6), the projection operator required for the subgradient procedure (cf. eq. (2.11)) has a particularly simple characterization. Given some $\bar{\pi} = \{\bar{\pi}_{rd}\} \in \mathrm{E}^{RD}$, we must find $\pi = \{\pi_{rd}\}$, minimizing $|\pi - \bar{\pi}|^2$ and satisfying (5.6). Noting that $|\pi - \bar{\pi}|^2 = \Sigma_r \Sigma_d (\pi_{rd} - \bar{\pi}_{rd})^2$, we see that we need only find, for each r, the point $(\pi_{r1}, ..., \pi_{rd})$, minimizing $\Sigma_d (\pi_{rd} - \bar{\pi}_{rd})^2$ under the constraints $\Sigma_d \pi_{rd} = c_r$, $\pi_{rd} \geq 0$ for all d. In general, we want to solve the quadratic programming problem:

$$\min\{\tfrac{1}{2}|x - \bar{x}|^2 : \Sigma_j x_j = c, \text{ all } x_j \geq 0, j = 1, ..., n\} .$$

By the Kuhn–Tucker theorem [14], necessary and sufficient conditions that x solve the last-stated problem are the existence of multipliers λ and $v_1, ..., v_n$ such that

$$x_j - \bar{x}_j = v_j - \lambda, \qquad v_j \geq 0, \qquad x_j v_j = 0 \quad \text{for all } j.$$

If we set

$$x_j(\lambda) = \max\{\bar{x}_j - \lambda, 0\} , \qquad v_j(\lambda) = \max\{-(\bar{x}_j - \lambda), 0\} ,$$

then the Kuhn–Tucker conditions are satisfied, for $x = x(\lambda)$, $v = v(\lambda)$, and we have $x \geq 0$. It is then only necessary to find that value, λ^* of λ for which $\Sigma_j x_j(\lambda) = c$, and we have the solution. Since $\Sigma_j x_j(\lambda)$ is nonincreasing as a function of λ, piecewise linear, and attains every nonnegative value, this is readily done. Suppose the coordinates of $\bar{x}$

ordered $\bar{x}_1 \leq \bar{x}_2 \leq ... \leq \bar{x}_n$. The "breakpoint" values of λ between, and outside of, which $\Sigma_j x_j(\lambda)$ is linear, are $\bar{x}_1, \bar{x}_2, ..., \bar{x}_n$, and its value for $\lambda = \bar{x}_J$ $(J = 1, 2, ..., n)$ is $\Sigma_{j=J+1}^n (\bar{x}_j - \bar{x}_J)$.

Since $\Sigma_{j=1}^n x_j(\lambda^*) = c$ for any $\bar{x}_J < \lambda^*$, $x_J(\lambda^*) = 0$ and $\Sigma_{j=J+1}^n (\bar{x}_j - \bar{x}_J) > c$. Thus $x_J(\lambda^*) = 0$ for $J = 1, 2, ..., J_0$, where

$$J_0 = \max \left\{ J \colon \frac{\bar{x}_{J+1} + ... + \bar{x}_n - c}{n - J} > \bar{x}_J \right\} ,$$

and it is merely necessary to choose λ^* such that

$$\sum_{j=J_0+1}^{n} x_j(\lambda^*) = \sum_{j=J_0+1}^{n} (\bar{x}_j - \lambda^*) = c ,$$

i.e.,

$$\lambda^* = \frac{-c + \sum_{j=J_0+1}^{n} \bar{x}_j}{n - J_0} .$$

We tried the subgradient procedure on a variety of multicommodity flow problems. For each problem the capacities were randomly chosen integers. The problems "Complete 1" and "Complete 2" represent complete networks (all possible arcs present). "Bipartite" is a network with an essentially bipartite structure. Problem "Test" was especially constructed for testing the program and algorithm. "Random" is a randomly chosen subnetwork of the complete network. "Circle" was constructed so that the arc-chain linear programming formulation [7] would be small – there are only two chains from source to sink for each commodity; thus the total number of columns in the linear program is only 40.

In every case the algorithm terminated because the optimal set of π was fully dimensional. This need not always be the case: we have constructed the problem sketched in Fig. 5.1 whose optimal set contains no open set. The algorithm converged to a very accurate solution, but showed no signs of terminating.

Results on the six problems we studied are summarized in Table 3, and a plot of the value of w for the even-numbered iterations for problem Complete 1 in Fig. 5.2. The data for the first three of these problems will be found in the Appendix to this paper.

Table 3
Multi-commodity flow problems

	λ	iterations	best w
Problem Random			
$w^* = 361$	2	7	361
25 nodes			
172 arcs			
4 commod.			
Problem Complete 1			
$w^* = 1924$	2	28	1888.45
14 nodes	1	14	1916.47
182 arcs	1/2	3	1924
4 commod.		45	
Problem Bipartite			
$w^* = 4935$	2	60	4911.81
30 nodes	1	30	4931.99
285 arcs	1/2	6	4935
15 commod.		96	
Problem Test			
$w^* = 172$	2	29	172
17 nodes			
40 arcs			
3 commod.			
Problem Circle			
$w^* = 38$	2	40	19.94
10 nodes	1	20	30.80
20 arcs	1/2	10	34.01
20 commod.		70	
	(then $\lambda \leftarrow \lambda/2$ every 10 iterations; w terminated at 38 at iteration 115)		
Problem Complete 2			
$w^* = 19777$	2	60	18138.21
30 nodes	1	30	19318.71
870 arcs	1/2	15	19708.84
10 commod.	1/4	6	19777
		111	

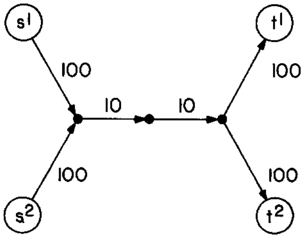

Fig. 5.1. Numbers indicate arc capacities.

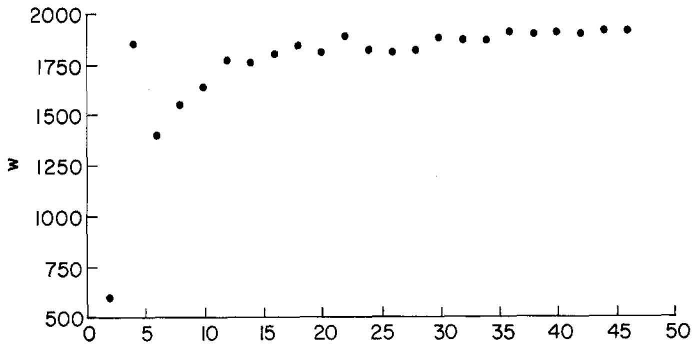

Fig. 5.2. (Problem Complete 1).

6. Establishing optimality

The method of this paper differs from almost all other optimization procedures in not containing, as part of its machinery, an effective test for the optimality of its solutions. Since the point of our experimental work was to see how well the method would approximate optimality, we were obliged to devélop such tests in order to evaluate the answers we got. The material of this section is thus a contribution to the art of obtaining the exact solution of a large linear programming problem with the assistance of subgradient optimization, but it is only a small step in that direction. We are content, at this time, to have some sufficient conditions for optimality which are satisfied by the data of the problems we have studied.

We will discuss optimality conditions here only for the unconstrained problem; the exact solutions of our one type of constrained problem – multicommodity flow – were obtained directly (see Section 5). Throughout this section we will study the sequence $\{\pi^j\}$ given by (2.6), supposing only that the step sizes satisfy conditions (2.7). Then we know [20, 21] that

$$\lim_{j\to\infty} w(\pi^j) = w^* = \max w = w(\pi^*) \text{ for some } \pi^*.$$

Our results require that the sequence $\{\pi^j\}$ be bounded. A condition ensuring this is given by

Theorem 6.1. *If for all* $\pi \neq 0$ *there exists* v_k $(k = 1, ..., K)$ *with* $\pi \cdot v_k < 0$, *then any sequence* $\{\pi^j\}$ *satisfying* (2.6), (2.8) *is bounded.*

Proof. If $\{\pi^j\}$ is not bounded, then there is a subsequence on which $|\pi^j| \to \infty$ and a point π which is the limit of $\pi^j/|\pi^j|$ on the subsequence. Now on the subsequence

$$\min_k \left[\frac{c_k}{|\pi^j|} + \frac{\pi^j}{|\pi^j|} \cdot v_k\right] = \frac{w(\pi^j)}{|\pi^j|} \to 0$$

(since $w(\pi^j) \to w^*$), so we have $\pi \cdot v_k \geq 0$ for all k, which proves the theorem.

The condition just established is readily checked for the traveling-salesman problem. The condition does not hold for the assignment problem in the formulation of Section 3, because $w(\pi)$ is unchanged if $\pi = (\pi_1, ..., \pi_n)$ is replaced by $(\pi_1 + \alpha, ..., \pi_n + \alpha)$ for any α; but if π is normalized by requiring always $\pi_1 = 0$, say, then the condition is easily checked.

To shorten notation below, we shall write

$$v^j = v(\pi^j) = v_{k(j)}, \quad c^j = c_{k(j)} ,$$

where $k(j)$ is the index chosen at step j in the calculation (2.2) of $w(\pi^j)$. Note that the infinite sequences $\{c^j\}$, $\{v^j\}$ have only finitely many distinct members.

Writing the problem (2.2) as the linear programming problem

$$\max\{z \colon z - \pi \cdot v_k \leq c_k \text{ for all } k\} ,$$

we find its dual to be the problem

$$\min\left\{\sum c_k y_k \colon \text{all } y_k \geq 0, \sum y_k = 1, \sum -v_k y_k = 0\right\} . \tag{6.1}$$

To the extent that the subgradient algorithm is clever about its choices v^j of subgradients, we can expect the important part of the above problem to be represented by the choices $v^J, v^{J+1}, ...$ for sufficiently large J. Thus for integers $J < J^*$ we define the linear programming problem $P(J, J^*)$ as:

$$\min\left\{\sum_{j=J}^{J^*} c^j y_j \colon \text{all } y_j \geq 0, \ \sum_J^{J^*} y_j = 1, \ \sum_J^{J^*} -v^j y_j = 0\right\} .$$

Lemma 6.2. *For any J there exists J^* so that $P(J, J^*)$ is feasible.*

Proof. If not, then the convex hull of the finite set $\{v^J, v^{J+1}, ...\}$ does not contain the origin. The nearest point $d \neq 0$ to the origin in that set is such that $d \cdot v^j \geq |d|^2$ for all $j \geq J$, so for $j > J$

$$d \cdot \pi^j = d \cdot \left[\pi^J + \sum_{i=J}^{j-1} t_i v^i\right] \geq d \cdot \pi^J + |d|^2 \sum_{i=J}^{j-1} t_i \,,$$

which tends to ∞ as $j \to \infty$, contradicting the boundedness assumption on $\{\pi^j\}$.

Theorem 6.3. *For any J there exists J^* so that the solution of $P(J, J^*)$ solves the linear programming problem* (6.1).

Proof. Let $\overline{K}$ be the set of all indices k actually chosen in the calculation of w in (2.2) for some j, $J \leq j \leq J^*$. For J^* sufficiently large, K contains all the indices that will ever be chosen, and $P(J, J^*)$ is feasible. Define the function $\overline{w}$ by

$$\overline{w}(\pi) = \min\{c_k + \pi \cdot v_k \colon k \in \overline{K}\} \,.$$

Then, just as for the original problem, the problem max $\overline{w}(\pi)$ is the dual of $P(J, J^*)$, and since the latter is feasible, the former has a finite solution. Thus the subgradient algorithm applied to $\overline{w}$, started at $\pi = \pi^J$, yields a sequence $\{\overline{\pi}^{J+1}, ...,\}$ such that

$$\lim_{j \to \infty} \overline{w}(\overline{\pi}^j) = \max \overline{w} = \text{Value}\,[P(J, J^*)]\,,$$

the left-hand equality expressing the convergence of the algorithm for $\overline{w}$, and the right-hand equality the duality theorem for linear programming. But the restriction to $\overline{K}$ has in no way altered the choices made by the algorithm; $\overline{\pi}^j = \pi^j$ and $\overline{w}(\overline{\pi}^j) = w(\pi^j)$ for $j \geq J$, so

$$\lim_{j \to \infty} w(\pi^j) = \text{Value}\,[P(J, J^*)]\,.$$

Since the left-hand term is a lower bound for the value of the problem (6.1) and the right-hand term is an upper bound for it, the theorem is proved.

The theorems just proved were not used in a systematic way to determine the optimal values of the functions w for the various problems reported above. The theorems were, in fact, developed later. We give two examples of the ad hoc means actually used to determine optimal solutions for those problems for which the algorithm did not terminate.

We ran the algorithm on problem A42 until we felt it could not do much more: At 319 iterations, w reached 531.9939 (the optimal value must, of course, be an integer). At that point we calculated all the reduced costs $a_{ir} - \pi_i - \rho_r$ (they are nonnegative by construction) and found that we could clearly distinguish those that were nearly zero (less than 10^{-4}) from the remainder (all greater than 0.1). There were 48 of the former, as against a minimum possible number of 42, hence only six men and six jobs not decisively assigned at this stage. It was a simple matter to find, by hand, a one-one assignment using only the nearly zero positions, which was thus a "nearly optimal" assignment. Its cost turned out to be 532, so it was indeed optimal.

Solutions of the traveling-salesman problems were found using Theorem 6.3 above, but in the special way described here for T42. We had noticed in all our runs long series of steps in which successive subgradients satisfied the relation $v^j = v^{j+1}$ in all but a few components, so we continued the algorithm to 1850 iterations and an objective value of 696.9999985, where we found $v^{1850} + v^{1851} = 0$. The corresponding costs c^j were 688 and 706. Since $\frac{1}{2}[688 + 706] = 697$, we have established the optimality of the resulting "fractional solution" $\frac{1}{2}v^{1850} + \frac{1}{2}v^{1851}$. (Dantzig, Fulkerson and Johnson [4] exhibited a fractional solution of value 698 but no optimal solution.)

7. Conclusions

Our results show that subgradient optimization is effective for approximating the maximum of certain piecewise linear concave functions; to us, it is surprisingly effective. It has performed well on large optimization problems of the kind normally encountered in applications of decomposition to combinatorics – problems in which the Dantzig–Wolfe decomposition algorithm, for example, tends to "tail off" and require an excessive number of iterations. Naturally we hope that it will perform as well when used on more general linear programming problems of decomposable structure, but at this point we cannot say. Neither do we know whether the algorithm suggested by the results of the previous section (the building up of the "core problem" $P(J, J^*)$ by the subgradient method, and the solution of the core problem by, say, the simplex method) will be generally efficient; that procedure, and variants such as the "Boxstep" procedure of Marsten and Blankenship [16], need much more exploration.

When "resource allocation" decomposition can be used, as we did with the multicommodity flow problem, then a feasible solution is always at hand; but of course, when the procedure is applied to the dual of the problem of interest – as above with the assignment and traveling-salesman problems – one has, unless the procedure "luckily" terminates, only an approximation to the values of optimal dual variables and an approximation (a lower bound) to the optimal objective value. Indeed, the procedure has shown its practical usefulness so far in just that way: in making branch-and-bound practical in the Held–Karp solution of traveling-salesman problems [11], and the work of Fisher and Shapiro in integer programming [6]. We feel, however, that in any particular problem there will be sufficient structure to permit the construction of an optimal, or nearly optimal, solution to the original problem without great difficulty.

Briefly, we think that subgradient optimization holds promise for alleviating some of the computational difficulties of large-scale optimization. It is no panacea, though, and needs careful work to make it effective, but its basic simplicity and its wide range of applicability indicate that it deserves to be more widely studied.

Appendix

An established canon of research in the physical sciences is that experiments must be designed and described so as to be reproducible. There is no reason for the mathematical sciences to be less demanding; it should be possible for the reader of a paper to duplicate every finding, right down to the least digit, provided he has access to the same systems. (An exception must be made for one source of error in computational experiments: the execution time of a program will depend on other jobs being run concurrently, and we do not think it feasible in general to eliminate or control that source.) The primary requirements are a *complete* description of the algorithm and problem data. Of these, the latter is often violated. This is regrettable, since it has led to a

```
   0  77  13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  81  69  92   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0  86  77   6  27   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  33  35  39  20  32   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  58  88   2  71  84  28   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0  68  31  39  59  95  28  97   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0  69  94  93  15  19  32  11  84   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   5  24  33  61  23  11  34  35  89   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   8  56  94  44  62  46  66  98  61  48   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  38  15  60  73  70  81  90  60  42  91   3   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  38  58  77   4  93  55  13  19  72  81  60  13   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   6   3  91  26  15  35  30  11  57  51  44  61  15
   0   0   0   0   0   0   0   0   0   0   0   0   0  19  10  20  90  95  10  12  15  68  47  26  57
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  38  85   3  49  86  78  73  44  83   9   4
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  26   8  42  20  54   6  49  68  16  11
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  22   9  47  22  76  98  62  17  31
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  72   2  41  99  45  59  53  80
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  86  46   7  91  93  66   4
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4  49  37  23  12  96
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  34  97   4  76  44
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  48  10  48  94
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  26   4  67
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  92  52
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   6
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
SOURCE NODES:     1   2   3   4
SINK NODES:      22  23  24  25
```

Table A1
Capacities for Random

```
   0  78 173 128 185  20 116 172  22 179 180  77  29 194 114 110 145  56 113  68 173  40  30 170 145 155  91  45 185  78
 158   0  60  33  59 117 187  66 139 192  68 129  22 182 157  18 128  52  34 125  64  36  57 107 123 156 107  55  82  44
 151 101   0  41  81 117  74  48  30 142 192  74  72 141 113 113  62  12 182 136  21  56  71 113  32 138  19 122  26 157
  81  99  50   0 156  48 155  90 129 119 134  86  28   4  64 112  12   1 183  87 149   8 106 114  84  24 151  34  11 143
 127  88 107 111   0  37  64 106 107  34  73  45 115 155 169  55 113  92  62  90 101 141 198  66  13  99 129 102  34  45
  35  34 190 139  21   0 187  30  76 133  33  94  43  92  35 120 172 141 121 162  64 174  31 168 125 173  19 102 103 186
  63   8 175  32  82   1   0  20  91  89 138 162  18 154  85  17   9  21 165 154  74 171 199  17  24 152 146  91 109 102
  60 134 134 136 105  95 173   0  62 130 135 121 115  92 198  26 108  55 101  68  66  56 113 182  20 148 107  25 126 140
 190 137  25 125  85 155 145 120   0 178 162  68 132 166  15 196 164 135 120  21 124   1  83 141 102 119 122 128 171 121
 199   8 146 123 166  74 108  68 135   0 159 115 153 163 110 145  13 114   8 109 149  99  77   4  35 182 166 141  84  64
  80  17  47  12  27  51 146 123  63 194   0  72 127  81 120 100 196  64 170  65 175 181 114 151  89  21 100  93 170  90
  88  26 165 165  71  35 102 126 161  66 115   0 180  20 127  98  58 157 147   1 111 191 190  66  10  89 103  88  56 165
   7   1 170  79 175 148  90  50 198  43  71 128   0 191  80  85 184 167 144  26  88 116 111  31  64 136 115  91   9 174
  49 165 130 190  12 171  23 165 137  67 191   4 199   0 139  24 183 187 145 141 130 174 151 192  86  19 166 153 146 160
  26 183  51  27  96  95 125 137  34  10 107  70 164  51   0 178  41  17 169  10 189 197 109  73 143  83 156 100 191 191
 115  10 131  47  68  32 143 135 168  55 122 184 189  52  16   0 194  17 109 188 157  16  93 108  82 128  42  15  32  99
  55  10 125  39 142 133 131 174 102  39 171  74  58  47  85 119   0 143 126  35  39 120 173 194   2  20  48 193  66  75
  90 170 170  11  38 108  97  30 101 138 120 190  44  17 162  25  31   0 177 122  21 178   3 186  67 166 106  18 111 159
  40 127  92 110 149  43   3  13  85  70 186  68 142  76   3  82  12 133   0  10 174 185  57 173 184   9   4 171  69  81
  32  26  18 157 155  88 144   1   2 175  29  97  58  94 162  62 196  62 144   0  52 106 115 162  98 158  70  59  10  52
   0  12  77 152 186 139  35  92  48  22 100 165  66  81 140  29  62 170 161  84   0 168 178  10 129  71 178  13  40 179
  84   0  98 100 121  39 109 137  66 152 138 120  73  38 183  18  10  58  66  30   4   0  30   2  89 198  36 148 150   5
  36 111   0  29  45 181 141 196 160 118 124  29  45 101  36  42 101  19 144  26 107 184   0 120 166 148  18 169 139  19
  55 136  67   0 181 139  62 138 171 136 100  95  13  64 169  84 109 190  69  84   3  51 190   0 198 104 179  81  78  71
   5  16 173 184   0   1  95  66 179 146  81 107  65  69  89 129 147 166 123  61  53  48 149 133   0  87 111 165  17  50
 128  71  31  32  46   0  14 149 135  55 182 189  75  37   9  13  97  82 109  11  83  29 180  73 160   0  52  61 197  64
 126  83  27 136 163 101   0 103 195  14 160 167  60 194  55 115  56 198 149 117  24 137 137  43 145 143   0  76  28 133
 195  60  84 129 161  32  30   0 152 187 125  99  88  33  45 172  58  45 109   1 189 157 198 194  24 165 182   0 187  15
  38   9  55 146  94  84 141 100   0 188 105  89 184  48 176 194  44  87 109 149 121 109 152 180  40  96 107 144   0  94
  70 109 171 117 111  12 174 155 164   0 104  39 109 192  49  16 142 116  76  86  22  36 123 127  19  93   2 114 158   0

SOURCE NODES:     1   2   3   4   5   6   7   8   9  10
SINK NODES:      21  22  23  24  25  26  27  28  29  30
```

Table A2

Capacities for Complete 2

0	16	7	12	88	58	59	51	93	34	19	28	49	15
71	0	35	55	69	57	58	82	62	98	5	4	60	84
44	30	0	18	14	19	11	34	30	82	54	71	17	66
29	59	81	0	67	97	45	63	18	8	80	88	64	33
72	84	36	81	0	6	64	74	41	7	91	81	31	7
73	73	81	20	16	0	80	74	80	61	26	94	5	39
86	78	64	90	61	1	0	71	8	5	33	35	47	71
57	9	49	62	20	72	67	0	28	79	66	61	40	75
62	52	16	55	67	20	16	17	0	26	93	51	11	12
60	30	55	73	21	67	93	65	97	0	64	54	6	94
0	15	88	38	20	21	10	41	49	41	0	69	37	50
30	0	29	66	28	25	23	56	19	28	67	0	62	51
20	51	0	54	89	56	40	63	70	98	35	15	0	58
50	78	24	0	77	11	36	77	45	34	7	81	78	0

SOURCE NODES:	1	2	3	4
SINK NODES:	11	12	13	14

Table A3

Capacities for Complete 1

dearth of "benchmark" problems in the literature, which in turn has made it almost impossible to compare algorithms.

We have described the algorithm used in our experiments in about as much detail as possible short of giving the computer programs. Our calculations were done variously on the IBM 360, Models 91 and 67 at the IBM Research Center, Yorktown Heights, using the FORTRAN and PL/I languages. Sources of the data for the assignment and traveling-salesman problems were cited at the end of Section 3. We have found no large multi-commodity flow problems in the literature, and offer here the data for the first three such problems we generated. For reasons of space, we have omitted the remaining three; they are available from the authors.

References

[1] J. Abadie and M. Sakarovitch, "Two methods of decomposition for linear programming", in: *Proceedings of the Princeton symposium on mathematical programming,* Ed. H.W. Kuhn (Princeton University Press, Princeton, N.J., 1970) pp 1–23.

[2] S. Agmon, "The relaxation method for linear inequalities", *Canadian Journal of Mathematics* 6 (1954) 382–392.

[3] D.P. Bertsekas and S.K. Mitter, "Steepest descent for optimization problems with non-differentiable cost functionals", in: *Proceedings of the 5th annual Princeton conference on information sciences and systems,* 1971.

[4] G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, "Solution of a large-scale traveling-salesman problem", *Operations Research* 2 (1954) 393–410.

[5] V.F. Dem'janov, "Seeking a minimax on a bounded set", *Soviet Mathematics Doklady* 11 (1970) 517–521. [Translation of: *Doklady Akademii Nauk SSSR* 191 (1970).]

[6] M.L. Fisher and J.F. Shapiro, "Constructive duality in integer programming", Working Paper OR 008-72, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Mass. (April, 1972).

[7] L.R. Ford, Jr. and D.R. Fulkerson, *Flows in networks* (Princeton University Press, Princeton, N.J., 1962).

[8] A.M. Geoffrion, "Elements of large-scale mathematical programming", *Management Science* 16 (1970) 652–691.

[9] R.C. Grinold, "Steepest ascent for large-scale linear programs", *SIAM Review* 14 (1972) 447–464.

[10] M. Held and R.M. Karp, "The traveling-salesman problem and minimum spanning trees", *Operations Research* 18 (1970) 1138–1162.

[11] M. Held and R.M. Karp, "The traveling-salesman problem and minimum spanning trees: part II", *Mathematical Programming* 1 (1971) 6–25.

[12] M. Held and R.M. Karp, "A dynamic programming approach to sequencing problems", *Journal of the Society for Industrial and Applied Mathematics* 10 (1962) 196–210.

[13] L.L. Karg and G.L. Thompson, "A heuristic approach to solving traveling-salesman problems", *Management Science* 10 (1964) 225–248.

[14] H.W. Kuhn and A.W. Tucker, "Nonlinear programming", in: *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, Ed. J. Neyman (University of California Press, Berkeley, Calif., 1951) pp. 481–492.

[15] L.S. Lasdon, *Optimization theory for large systems* (Macmillan, London, 1970).

[16] R.E. Marsten and J.W. Blankenship, "Boxstep: a new strategy for Lagrangian decomposition", Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Ill. (March, 1973).

[17] T. Motzkin and I.J. Schoenberg, "The relaxation method for linear inequalities", *Canadian Journal of Mathematics* 6 (1954) 393–404.

[18] J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem", in: *Contributions to the theory of games*, Vol. II, Eds. H.W. Kuhn and A.W. Tucker, Annals of Mathematics Study No. 28 (Princeton University Press, Princeton, N.J., 1953).

[19] W. Oettli, "An iterative method, having linear rate of convergence, for solving a pair of dual linear programs", *Mathematical Programming* 3 (1972) 302–311.

[20] B.T. Poljak, "A general method of solving extremum problems", Soviet Mathematics Doklady 8 (1967) 593–597. [Translation of *Doklady Akademii Nauk SSSR* 174 (1967).]

[21] B.T. Poljak, "Minimization of unsmooth functionals", *U.S.S.R. Computational Mathematics and Mathematical Physics* 14–29. [Translation of: Žurnal Vyčislitel'noĭ Matematiki i Matematičeskoĭ Fiziki 9 (1969) 509–521.]

[22] R.T. Rockafellar, *Convex analysis* (Princeton University Press, Princeton, N.J., 1970).

[23] N.Z. Shor, "On the structure of algorithms for the numerical solution of optimal planning and design problems", Dissertation, Cybernetics Institute, Academy of Sciences U.S.S.R. (1964).

[24] P. Wolfe, M. Held and R.M. Karp, "Large-scale optimization and the relaxation method", in: *Proceedings of the 25th national ACM meeting*, Boston, Mass. (August 1972).