PAPERS IN REGIONAL SCIENCE: The Journal of the RSAI 71, 3: 331–52

ALGORITHMIC AND TECHNICAL IMPROVEMENTS: OPTIMAL SOLUTIONS TO THE (GENERALIZED) MULTI-WEBER PROBLEM

Kenneth E. Rosing
Economisch-Geografisch Instituut
Erasmus Universiteit Rotterdam
Postbus 1738
3000 DR Rotterdam
The Netherlands

Britton Harris
Department of City and Regional Planning
University of Pennsylvania
Philadelphia, Pennsylvania 19104
USA

ABSTRACT Rosing has recently demonstrated a new method for obtaining optimal solutions to the (Generalized) Multi-Weber Problem and proved the optimality of the results. The method develops all convex hulls and then covers the destinations with disjoint convex hulls. This paper seeks to improve implementation of the algorithm to make such solutions economically attractive. Four areas are considered: sharper decision rules to eliminate unnecessary searching, bit pattern matching as a method of recording a history and eliminating duplication, vector intrinsic functions to speed up comparisons, and profiling a program to maximize operating efficiency. Computational experience is also presented.

1. INTRODUCTION

The Generalized Multi-Weber Problem consists of finding the locations of some specified number of centers (two or more) such that the total distance of each from the group of weighted fixed points assigned to it (fixed points are assigned to their closest center) is at a minimum and such that the total aggregate distance, fixed points to centers, is at an absolute minimum. It is analogous to the p-median problem; however, because it is not on a network, the theorem of Hakimi (1964, 1965; Levy 1967) is not applicable and the Multi-Weber Problem cannot be reduced to a combinatorial problem in this way.

This paper analyzes some practical difficulties and their solutions in finding optimal (and potentially optimal) solutions for the (Generalized) Multi-Weber Problem (Rosing 1992a).

Motivation

The model form of the (Generalized) Multi-Weber Problem is applicable to problems such as regionalization and the districting of territories. For example, it was proposed by Rosing (1992b) as the most appropriate model for assigning

An earlier version of this paper was presented at the 37th North American Meetings, Boston, November 1990.

injection wells to steam generators (to be located and built) in a heavy oil field. Several methods for finding good solutions are known. However, some of these depend on a restatement of the problem while others depend on the use of heuristics. This paper presents the first potentially usable method for finding optimal solutions for strictly defined problems of moderate size that meet the original terms of reference of the problem. Such a capability provides an opportunity to examine in depth the performance of heuristics by measuring their approach to optimality in solving Multi-Weber problems of realistic size.

The success of this approach depends upon an original formulation (discussed in Rosing 1992a), on the increasing availability of very fast computers, and on a number of programming innovations. In most practical applications, the method converts a very difficult nonlinear mixed integer programming problem (Garey and Johnson 1979) into a relatively easy integer programming problem, but one that retains the type of high-order complexity that is not feasible to explore without a great deal of computing power. At the same time, fast computers (such as many desktops) are not effective unless continued attention is given to the overall structure and details of the programming. This paper discusses the application of this dictum to arrive at a practical and workable approach to a previously theoretical approach to solution.

Definitions of the Problem, Method, and Approaches to Programming

Consider a set of points fixed on a plane. Following Cooper (1963) these fixed points are called *destinations* and are indexed $k = 1, 2, 3, \ldots, n$. Other points, whose location is to be determined, are called sources and are indexed $j = 1, 2, 3, \ldots, p$; $p < n$.

The problem of locating a single source to minimize (in Euclidean space) the distance to some number of destinations is generally referred to as the Fermat problem or the Weber problem. While the problem initially involved three destinations, it was soon extended to n destinations. Later it was extended to the consideration of n weighted destinations and termed the Generalized Weber Problem. This problem was solved by Weiszfeld (1937) and others. Elements of the history of this problem can be found in Kuhn and Kuenne (1962), Cooper (1963), Kuhn (1967), Ostresh (1978a, 1978b), Love, Morris, and Wesolowsky (1988), and Rosing (1991, 1992a).

Miehle (1958) first stated and Cooper (1963) first formalized the Multi-Weber Problem. In the extension of the Weber Problem to multiple sources, an optimal partitioning of the destinations must first be found and then the location of the one single source for each set must be found that minimizes the sum of the distances from each destination to its assigned source. When the n destinations are weighted, the problem becomes the Generalized Multi-Weber Problem. Kuenne and Soland (1970, 1972) and Ostresh (1973a) have published implicit enumeration algorithms for this problem. While both fail to converge in any except small and trivial problems (due to lack of computer time and space), their performance would undoubtedly improve in a modern computing environment.

The solution method used here for the (Generalized) Multi-Weber Problem depends on a theorem, first stated by Harris, Farhi, and Dufour (1970, 1972),

that requires any optimal solution to consist solely of disjoint convex sets (or hulls), each with an optimally located center, where the convex sets have been chosen from an exhaustive partitioning of the destinations.

A *convex hull* is a set of boundary lines around one or more points with the property that the line segment connecting any two points in the hull lies entirely within the hull. Each destination is, by itself, a separate convex hull. Each pair of destinations has a convex hull consisting of a straight line connecting the two points. Any set of three or more non-collinear destinations have a convex hull that encloses an area. This convex hull is composed of straight lines connecting the outermost points of the set in such a way that no interior angle is greater than 180° (that is, it is a convex polygon). *Non-overlapping* or *disjoint* convex hulls have distinctly different memberships where no member of one convex hull lies on or within another convex hull, so that no destination has membership in two or more convex hulls.

In any feasible solution to the Multi-Weber Problem, including the optimal solution, all destinations must be contained in non-overlapping convex hulls. One source will correspond to each of the convex hulls (Galvani 1933) and will also be inside that convex hull (Kuhn 1967). Convex hulls are indexed $i = 1, 2, 3, \ldots, m$.

2. THE OPTIMAL SOLUTION METHOD

Rosing's (1992a) method is to identify all possible convex hulls (which could, geometrically, form an element of the optimal solution), find the (Generalized) Single-Weber point of each, and calculate the partial objective function associated with each such hull. Lists of memberships of convex hulls, and their associated functional value, are kept and used to write a constraint set for a Set Covering Problem (SCP). A linear program (LP) is then used to cover the destinations with convex hulls, minimizing the sum of distances from destinations to sources (the objective function).

The algorithm for the identification of all convex hulls is fully described in Rosing (1992a). Briefly, it is to divide and divide again. Given a set of destinations in a plane, a single straight line, passing through the set, will divide it into two subsets, each contained within one of two disjoint (non-overlapping) convex hulls (Harris, Farhi, and Dufour 1970, 1972; Ostresh 1975; Drezner 1984). For a set of n destinations there are $n(n-1)/2$ possible such lines, and thus possible pairs of convex hulls. Enumerating the objective function of all such pairs (and choosing the minimum value) provides the optimal solution to the (Generalized) Multi-Weber Problem when the number of sources equals 2 (Harris, Farhi, and Dufour 1970, 1972; Ostresh 1975; Drezner 1984). Take one such line (and thus two convex hulls); call the hulls *I* and *II*. Considering sub-set *I* first consisting of n_I members, obviously there are $n_I(n_I - 1)/2$ such lines and half-lines (half-lines are now introduced because some lines terminate at the line between *I* and *II*); there are also a like number of pairs of non-overlapping sub-sub–sets, each contained within its own convex hull. This first set of divisions comprises level 1. Again take one such line (or half-line) and the two associated convex hulls and call them *A* and *B*. This is division level 2 and is referred to as *deeper* than level 1 (because this second level of division takes place within the former).

Considering first sub-sub–set *A*, composed of n_{IA} members, there are now $n_{IA}(n_{IA} - 1)/2$ pairs of convex hulls within a sub-sub-sub–set that can be created by passing a line, a half-line, or a line segment (line segments are now introduced because a dividing line can terminate on one end at the line between *I* and *II* and on the other at the line between *A* and *B*) through the points contained in convex hull *A*. Name an arbitrary pair of these *i* and *ii*. Taking sub-sub-sub-sub–set *i*, there are $n_{IAi}(n_{IAi} - 1)/2$ pairs again; this is level 3. The process could be continued in the same manner to deeper and deeper levels.

At level 1, complete enumeration of all possible lines creates all possible pairs of convex hulls and the optimal solution for the Two-Weber Problem. At level 2, complete enumeration of all possible lines and half-lines within *I* and then within *II* gives all possible convex hulls within *I* and *II*. Choosing a new *I* and *II* and repeating the procedure at level 2 until all possible lines at level 1 have been employed gives a list of all possible convex hulls that could form an element of the solution for the Three-Weber Problem. Similarly, complete enumeration within *IA* of all possible pairs *i* and *ii*, followed by complete enumeration of all possible pairs *i* and *ii* within IB, followed by *IIA* and *IIB*, the choosing of a new dividing line between *A* and *B* until all pairs in them have been enumerated (and the pairs at level 3 each time as well) will, upon exhaustion of possible pairs *I* and *II* (and each time exhaustion of all possible pairs at level 2 and level 3), create all possible convex hulls that could be a member of the solution for the Four-Weber Problem. Each time a new convex hull is identified, its Single-Weber Point must be found and recorded, its partial objective function must be found and recorded, and the membership must be recorded.

The number of different disjoint convex hulls is a function of the geometry (spatial arrangement) of the destinations, the number of destinations, and the number of sources to be located (since the more sources there are, the more levels of division exist and hence the more complex the identified geometry is). As has been frequently noted (for example, Fisher 1969; Cooper 1963; Ostresh 1975; Keane 1975), the combinatorial solution space of the Multi-Weber Problem is a Sterling Number of Type II, symbolized $S_{(n,p)}$. While the number of convex hulls is large, it is much smaller than the corresponding Sterling Number. For example, $S_{(25,7)}$ = 227,832,482,998,716,310 (Abramowitz and Stegun 1972) while the number of distinct convex hulls for $n = 25$, $p = 7$ is approximately 84,000 (the exact number is arrangement-dependent).

The three proven methods for the identification of pairs of convex hulls (Harris, Farhi, and Dufour 1970, 1972; Ostresh 1973b, 1975; and Drezner 1984) differ in technical approach. Their results, however, are the same: they repeatedly divide the full set of sources into two disjoint convex hulls, enumerate all $n(n - 1)/2$ pairs, and declare the pair with the minimum functional value to be the optimum solution to a Weber problem with two sources. Rosing (1992a) proves that by dividing each part of a partition in two, dividing each of these again, then again dividing each of these, etc., one will find, with $p - 1$ divisions, all feasible convex hulls for p sources. Any of the three methods listed above may be used in this division process. The TWAIN algorithm (Ostresh 1973b, 1975) was used in this experiment.

This new method for finding the optimal partitioning of the destinations into p parts now consists of two steps. First, all feasible convex hulls (feasible for a given number of sources) in a set of n points are found; in the process, all duplicates are discarded. Second, this large set of convex hulls is used to solve the problem with a set covering formulation in a discrete linear program. Two stages are involved.

A *preparation* program first generates all possible convex hulls, for a given value of p, discarding the duplicates found. As each unique (new) convex hull is found its Single-Weber Point is found, using the improved (Kuhn and Kuenne 1962; Ostresh 1978a, 1978b; Weiszfeld 1937) algorithm. The potential contribution to the objective function and the appropriate set of constraints are then written to disk in Mathematical Programming Standard (MPS) format.

The completed constraint set is then submitted to a set covering formulation and solved by a standard LP package. This set covering formulation, given by Rosing (1992a), is so well known that a discussion of it is not necessary, except to say that the full set of points must be covered by selected convex hulls. The formulation is

$$\text{Minimize:} \quad Z = \sum_{i=1}^{m} c_i X_i \tag{1}$$

$$\text{subject to:} \quad \sum_{i=i}^{m} X_i = p^* \tag{2}$$

$$\sum_{i \in M_k} X_i \leq 1 \qquad \forall\, k \tag{3}$$

$$X_i = (0, 1)$$

where

i = the index number of the convex hull and of the functional value of that convex hull;
m = the number of convex hulls;
k = the index number of the fixed points;
X_i = is a decision variable that equals one if the i^{th} convex hull is chosen, and zero otherwise;
c_i = the summed weighted distances of all fixed points to the one Single-Weber point of that convex hull (the point of minimum aggregate travel of that convex hull — that is, the cost associated with each hull); and
M_k = $\{i \mid$ the set of convex hulls of which k is a member$\}$.

The problem of improving the efficiency of the programs that perform these tasks was approached using four main methods. First, various bounds were greatly sharpened, thus reducing the amount of work in both parts of the solution method (see Section 3). Second, bit coding and bit pattern matching of lists of points (destinations) in various subsets of the destinations were employed to very striking effect (see Section 4). Third, the capabilities of advanced computers were used to great advantage for vector processing (see

Section 5). And fourth, modern methods of program profiling were used as a guide to increasing the efficiency of the code produced (see Section 6).

All results presented here were obtained using the Linear and Mathematical Programming System (LAMPS) (Advanced Mathematical Software 1986) on a Convex 210 computer with a UNIX 7.1.0.1 operating system. LAMPS is a new, modern, and very powerful implementation of the revised simplex algorithm. Fortran programs were compiled using Convex (1988a) UNIX Fortran version 5.0 with optimization level two (vector processing).

The 15-point data set is that published by Cooper (1964). The 20- and 25-point data sets are Cooper's 15-point set, each time augmented by an additional 5 random points. These later data sets were published by Rosing (1992a).

3. SHARPER DECISION RULES

In the preprocessing program two resulting diagnostic numbers are important for measures of speed and thus for the efficiency of the convex hull algorithm and the speed and efficiency of the LP. The first number is the total number of convex hulls identified (which affects the efficiency of the preparation program) and the other is the number of unique convex hulls (which affects the efficiency of the LP). Good decision rules are rules that are simple in the sense that they have a low level of complexity of calculation and thus will significantly increase the efficiency of the total process. If a decision rule is very complex to compute, it can easily take more time to use than the time saved by employing it. (See Table 1 for solutions to problems not given in Rosing 1992a.)

Reducing Total Number of Convex Hulls Identified

Each time a convex hull is identified, its membership list (list of destinations) must be compared with all previously identified unique convex hulls to determine whether it is a new, unrecorded convex hull; that is, whether it is a unique

TABLE 1. Solutions to Problems 15,7; 20,7; 25,7

Problem *n p*	Objective Function	Group Memberships
15/7	70.633	a. 1, 4 b. 2, 5 c. 3 d. 7 e. 13, 14, 15 f. 10, 11, 12 g. 6, 8, 9
20/7	100.033	a. 3 b. 16 c. 6, 7, 8, 9 d. 10, 11, 12, 18 e. 13, 14, 15 f. 2, 5, 17, 19 g. 1, 4, 20
25/7	136.513	a. 3 b. 16 c. 10, 11, 18, 22 d. 6, 7, 8, 9, 24 e. 12, 13, 14, 15, 21, 23 f. 2, 17, 19 g. 1, 4, 5, 20, 25

convex hull or a duplicate. Duplicates need not be recorded, their Single-Weber point need not be found, and they do not need to be included in constraints. For unique convex hulls, all these steps must be performed. Although comparisons are very fast (see below, Sections 4 and 5) significant savings are made by reducing the number of duplicates found.

Single-member convex hulls are referred to hereafter as *one-hulls*, just as two-member convex hulls are referred to as *two-hulls*, etc. One-hulls can be found simply by listing the points 1 to n. Two-hulls can be found nearly as simply. Each combination of two points forms a two-hull; there are $n(n-1)/2$ such two-hulls. Simply listing these hulls saves many later comparisons to identify unique and non-unique hulls. At any level of division a three-hull must be compared, and possibly be recorded (if unique), but it need not be further subdivided. Since no new two-hulls and one-hulls will be found, their identity need not be held in the main storage array. This increases the speed and efficiency of the preparation program but can have a small deleterious influence on the LP. This is because *all* one-hulls and two-hulls are listed including, probably, ones which could not be generated at the value of p being used.

The removal from the search of the one- and two-hulls leads, however, to the addition of a corollary improvement. At level 1 a number of different convex hulls of all different sizes will be found, some small, some large. When a large hull is passed down (that is, subjected to further subdivision), a number of hulls already found at level 1 will be found again, and then again at each lower level. Large hulls, but not small hulls, must be divided or passed down. At level 1 the smallest hull that must be divided has p members, and the smallest hull that must be passed down is of size $p+1$. At level 2 the smallest hull investigated is size $p-1$, and size p and larger groups are passed down, etc. This is because any hull smaller than p (at level 1) will be found at one of the lower levels. This saves the repetition of finding small hulls over and over again. Table 2 compares the results with and without the elimination of the one- and two-hulls in the preparation program. In each of these tables all other improvements, discussed below, are operational unless some interrelationship prevents it; in such cases what has been disabled is noted. The column under each value of n labeled "Disabled" contains the number of hulls found and the total CPU time for the preparation program without elimination. The column labeled "Enabled" gives the equivalent values with the elimination of one- and two-hulls and a halt in the checking of small hulls at high levels (which will be enumerated at lower levels). In the rows after each value of p on the line labeled "Hulls," the total number of hulls enumerated is given. (Note that, in the "Disabled" column, it is the total number of hulls generated; in the "Enabled" column, it is the number that passed the size test and continued to the history comparison step.) The line labeled "Time" contains the total CPU time in seconds. The lines labeled "%red" give the percent reduction in hulls and time, respectively. As Table 2 shows, although the number of hulls found is dramatically reduced, this has less effect on total time than had been hoped. However, even a 2 to 10 percent reduction of the total time is still significant. In certain cases (for very small problems), the improvement is negative. This is because the work of writing out to disk, in MPS format (the most time-consuming portion

TABLE 2. Savings From Screening to Remove One- and Two-Member Convex Hulls

		n = 15		n = 20		n = 25	
p		Disabled	Enabled	Disabled	Enabled	Disabled	Enabled
3	Hulls	3684	3226	9424	8719	16720	15979
	%red		12.43%		7.48%		4.43%
	Time	2.44	3.78	11.94	12.21	25.35	26.55
	%red		−54.92%		−2.26%		−5.15%
4	Hulls	18140	14672	59586	52612	121706	112792
	%red		19.12%		11.70%		7.32%
	Time	8.70	8.50	37.07	36.47	101.58	97.26
	%red		2.30%		1.69%		4.25%
5	Hulls	49350	37004	202788	172958	512572	466645
	%red		25.02%		14.71%		8.96%
	Time	13.23	12.81	71.00	69.92	248.06	247.84
	%red		3.17%		1.50%		0.02%
6	Hulls	92594	63345	470870	384529	1463272	1304992
	%red		31.16%		18.34%		10.87%
	Time	17.08	15.64	112.98	110.68	513.91	498.47
	%red		8.43%		2.04%		3.00%
7	Hulls	134838	81763	864848	666508	3201198	2777167
	%red		39.36%		22.93%		13.25%
	Time	19.49	17.53	161.17	150.51	924.90	902.67
	%red		10.06%		6.61%		2.40%

Note: All times are given in seconds.

of the program), all one- and two-hulls is greater than finding a much smaller number of such hulls.

Another series, not summarized here, was also run in which only one- and two-hulls were eliminated and no minimum pass-down size criterion was imposed. Of the improvement shown in Table 2, the majority (up to approximately 75 percent) is a result of eliminating one- and two-hulls. This is probably due to the imposition of a size criterion requiring an additional logical decision for each hull developed (see Table 2, "Hulls," "Enabled" column). The additional time required nearly cancels the improvement yielded.

Using Upper Bounds on Cost to Determine Eligible Convex Hulls

The highest-cost convex hull (cost being the total weighted distance from destinations to their source) in an optimal solution must be less than the objective value of the best solution that a heuristic has obtained for the full problem. Take the example where n = 15, p = 4, and a functional value, from a heuristic, is 100. If any single convex hull has a functional value higher than 100 it cannot be a component of the optimal solution. If a hull has a functional value of exactly 100 it can be a component only if the other three hulls are one-hulls and thus do not contribute to the functional value. Cooper's (1964) heuristic for the Multi-Weber Problem, ALTERNATE, still seems to be one of the most efficient algorithms for this problem. In our implementation ALTERNATE, as

coded by Ostresh (1973c), and modified to permit random starts, was used to find heuristic solutions to each problem. The heuristic was run 200 times on each problem and the best objective function value obtained was used as the upper bound for removal of any hull.

As discussed above, each hull developed must be checked for size to see if it should be passed down. Hulls surviving this test must be checked against the history matrix of already developed hulls, and, if unique, recorded for future reference. For the unique hulls the Single-Weber Point must then be calculated, and the cost must be tested against the cost criteria from the heuristic. The savings in the preparation program consist only of the elimination of writing to disk one column of the simplex tableau for the set covering formulation. Each column, corresponding to one unique convex hull, has an entry in Objective Function (1), an entry in one row for the summation of p (Constraint 2), and one entry per member of that convex hull in that members row (Constraint 3). Table 3 shows the comparison of problems in which a bound was employed and in which no bound was employed. The columns headed "No Bound" come from the problems reported in Table 2. Times remain the same but the "Hull" lines now show the number of unique hulls, whereas the Table 2 "Enabled" column reports the total number of hulls found. For each value of p, the row labeled "Bound" gives the functional value of the best solution found by

TABLE 3. Savings From Employing a Heuristic Upper Bound Preparation Program

		n = 15		n = 20		n = 25	
p		No Bound	Up. Bound	No Bound	Up. Bound	No Bound	Up. Bound
3	Bound		143.196		210.196		252.245
	Hulls	1370	837	3764	2392	7118	4586
	%red		38.91%		36.45%		35.57%
	Time	3.78	2.38	12.21	7.44	26.55	15.85
	%red		37.04%		39.07%		40.30%
4	Bound		113.567		169.433		207.411
	Hulls	2930	1303	10601	5035	23982	11635
	%red		55.53%		52.50%		51.15%
	Time	8.50	4.47	36.11	18.86	97.26	49.64
	%red		47.41%		47.77%		48.96%
5	Bound		97.289		134.262		169.841
	Hulls	4042	1436	17611	5948	50316	18334
	%red		64.47%		66.23%		63.56%
	Time	12.81	6.82	69.92	36.02	247.84	131.76
	%red		46.76%		48.48%		46.84%
6	Bound		81.263		116.312		152.672
	Hulls	4398	1113	21659	5789	71982	22935
	%red		74.68%		73.27%		68.14%
	Time	15.64	8.50	110.68	64.01	498.47	331.44
	%red		45.65%		42.17%		33.51%
7	Bound		70.633		102.734[a]		137.156[b]
	Hulls	4482	876	23507	4940	84111	22553
	%red		80.46%		78.98%		73.19%
	Time	17.53	9.82	150.51	101.40	902.67	700.37
	%red		43.98%		32.63%		22.41%

Note: All times are given in seconds.
[a] ALTERNATE non-optimal; FV = 100.033 (2.70% error).
[b] ALTERNATE non-optimal; FV = 136.513 (0.47% error).

ALTERNATE. (Non-optimal heuristic solutions have a footnote giving the optimal value and show the error as a percentage.) It should be remarked that in only two cases, the two largest problems ($p = 7$, $n = 20$, 25), was the best of 200 runs of ALTERNATE non-optimal. This may imply a significant degradation of the power of ALTERNATE as n and p increase. The remaining lines have the same meaning as the corresponding lines in Table 2.

Looking at the "No Bound" column of $n = 15$ as p increases, it can be seen that the rate of increase in the number of unique hulls decreases. Obviously there is a finite number of unique hulls. Even though as one goes to deeper and deeper levels in the division process the number of total hulls continues to grow, initially at an increasing rate, the number of undiscovered unique hulls begins to be depleted. The same can be seen in the "No Bound" columns $n = 20$ and $n = 25$, although the effect occurs at higher values of p. Looking now at the corresponding "Up. Bound" columns, the bound, of course, becomes tighter with higher values of p. More and more hulls are eliminated (those that are above the upper bound) at higher values of p.

It had been expected that the major savings of bounding would be on the LP which, with fewer columns in the tableau, would solve in less time; however, the savings in writing out the constraint set are 20 percent to 50 percent, certainly not inconsiderable.

Table 4 shows the comparison between problems in which a bound was employed and those in which no bound was employed. Each cell shows the results of solving a problem using the constraint set generated as described in Table 3. In each case the tableau has $n + 2$ rows, and the number of unique

TABLE 4. Savings From Using a Heuristic Upper Bound Linear Program

		$n = 15$		$n = 20$		$n = 25$	
p		No Bound	Up. Bound	No Bound	Up. Bound	No Bound	Up. Bound
3	Convt	6.1	3.0	21	10	47	23
	Priml	1.4	0.75	3.8	2.0	4.8	4.8
	Total	8.94	4.76	27.80	14.45	61.41	31.79
	%red		46.00%		48.02%	n.c.	29.62%
4	Convt	13	4.4	59	21	164	58
	Priml	2.5	0.98	8.9	3.5	28	19
	Total	17.77	6.56	75.87	27.95	212.07	85.83
	%red		63.08%		63.16%	n.c.	59.53%
5	Convt	17	4.5	96	23		85
	Priml	3.1	0.99	15	4.3	n.c.	19
	Total	23.21	6.79	124.47	30.67		116.89
	%red		70.75%		75.36%	n.c.	n.c.
6	Convt	18	3.3	115	20		102
	Priml	3.6	0.74	17	3.8	n.c.	20
	Total	25.11	5.06	148.36	27.97		137.33
	%red		79.85%		81.15%	n.c.	n.c.
7	Convt	19	3.3	127	16		96
	Priml	3.4	0.74	20	3	n.c.	19
	Total	24.27	5.07	160.23	22.62		129.46
	%red		79.94%		85.88%		n.c.

Note: All times are given in seconds. n.c. = not calculated, generally because the comparative problem or this problem was not or could not be solved because of technical difficulties.

hulls (see Table 3) is the number of columns. The time required to read in and translate the constraint set is given on the line "Convrt" of Table 4 and the time required for the actual solution is given on the line "Priml." Total time is indicated on the line "Total" and the difference between the sum of "Convrt" time plus "Priml" and "Total" time is roughly one-half system overhead and time to set the problem up. As expected, large savings are made in the reading and converting step. However, the savings in solution time are also notable. In general, total time is reduced by 50 percent to 85 percent in addition to the savings reported in Table 3. It is also encouraging that the largest savings are made in the larger problems.

Reducing Still Further the Number of Unique Convex Hulls

For any given n, the objective function is a monotonic decreasing function with increases in p. By solving a series of problems, one can develop a trade-off curve showing the effectiveness of each additional investment in facilities. Since, for a fixed n, as p increases the average number of destinations assigned to each source decreases, a problem solved for p facilities can provide information for the problem with $p + 1$ facilities. Moving to a higher level of p does not involve extensive reorganization of the clustering pattern. Rather, as inspection of the solutions in detail reveals, generally one large (expensive) convex hull is split into two cheaper hulls, sometimes with the addition of one or two destinations from another hull. The partial functional values, and memberships, of most convex hulls remain the same. The partial functional value of one or two convex hulls are reduced. For the problems solved here, the partial functional value did not increase for any convex hull; that is, no hull increased in total weighted distance. This is not surprising since the functional value must decrease (or remain the same) when moving from $p - 1$ to p sources. Given these observations, the partial functional value of the most expensive hull of the solution of $p - 1$ can be used as the upper bound on the cost/size of convex hulls of p. This is, generally, a much tighter bound than that described for using upper bounds on cost (see above). The results from using this bound for the preparation program are shown in Table 5.

At this time we are unable to prove that a single convex hull will *never* increase in partial functional value as p increases, but we firmly believe this to be the case. Thus, we can no longer guarantee optimality when this bounding method is used, since we cannot prove that one convex hull could not increase in partial functional value in some particular problem. (If a partial functional value did increase, the convex hull(s) that compose a portion of the optimal solution would be rejected by the preparation program and would not be available to the set covering step, perhaps resulting in a non-optimal solution.) The bounding methods discussed above do, however, guarantee optimality.

Column headings in Table 5 have the same interpretation as in Table 3, as do the labels of the lines, except that "%red1" is the time improvement calculated with the times in the Table 3 "Up. Bound" column (thus giving the relative improvement between the two upper bounds), while "%red2" is calculated from the Table 3 "No Bound" column (thus giving the total improvement over the unbounded program).

TABLE 5. Savings From Costliest Hull Upper Bound Preparation Program

p		n = 15 Lrgst Hull	n = 20 Lrgst Hull	n = 25 Lrgst Hull
3	Bound	120.361	196.197	245.499
	Hulls	671	2183	4435
	%red	19.83%	8.74%	3.29%
	Time	2.05	6.81	15.63
	%red1	15.55%	8.47%	1.39%
	%red2	45.77%	44.23%	37.69%
4	Bound	65.207	91.412	109.839
	Hulls	483	1667	3979
	%red	62.93%	66.89%	65.80%
	Time	3.03	11.32	29.01
	%red1	32.21%	39.98%	41.56%
	%red2	62.35%	68.65%	70.17%
5	Bound	43.313	45.415	80.415
	Hulls	296	630	3987
	%red	79.39%	89.41%	78.25%
	Time	4.96	25.85	97.02
	%red1	27.27%	28.23%	26.37%
	%red2	61.28%	63.03%	60.85%
6	Bound	34.680	43.404	56.650
	Hulls	226	659	2382
	%red	79.69%	82.62%	21.13%
	Time	7.22	54.87	286.14
	%red1	15.06%	14.28%	13.67%
	%red2	53.38%	50.42%	42.60%
7	Bound	27.034	43.313	56.650
	Hulls	178	698	2621
	%red	79.68%	85.87%	88.38%
	Time	8.90	94.18	658.42
	%red1	9.35%	7.12%	5.99%
	%red2	49.23%	37.43%	27.05%

Note: All times are given in seconds. %red, %red1 were calculated from "Up. Bound" column of Table 3; %red2 was calculated from "No Bound" column of Table 3.

Percentage improvements in time over the previous bound vary widely because the largest hull of the solution for $p - 1$ provides, in some cases, a relatively loose bound on the current problem, while in other cases the strength of the bound increases considerably.

Table 6 records, in the same manner as Table 4, the improvement of the LP. These improvements are, again, significantly greater than those of the preparation program. Again, the rapid lowering of the bound as p increases results in the greatest time reduction in the larger problems, enabling the solution of problems which were, with other steps discussed thus far, unsolvable.

Summary

From this work it is obvious that both the preparation program and the LP are input/output bound. An upper bound eliminates only the writing-to-disk step in the preparation program. The main steps of the program are accomplished very quickly. The estimated 8,000 hulls per second handled in the $n = 15$ problem was confirmed by analysis of the profile of execution of the program.

TABLE 6. Savings From Costliest Hull Upper Bound Linear Program

p		$n = 15$ Lrgst Hull	$n = 20$ Lrgst Hull	$n = 25$ Lrgst Hull
3	Convrt	2.2	9.3	22
	Primal	0.69	1.8	4.4
	Total	3.85	12.86	30.24
	%red1	19.12%	11.00%	4.88%
	%red2	56.94%	53.74%	50.76%
4	Convrt	1.3	5.3	14
	Primal	0.4	1.4	3.6
	Total	2.56	8.12	20.73
	%red1	60.98%	70.95%	75.58%
	%red2	85.59%	89.30%	90.22%
5	Convrt	0.70	1.6	13
	Primal	0.24	0.39	3.1
	Total	1.73	2.90	18.50
	%red1	74.52%	90.54%	84.17%
	%red2	92.25%	97.67%	n.c.
6	Convrt	0.51	1.6	6.8
	Primal	0.13	0.45	1.6
	Total	1.45	3.13	10.16
	%red1	71.34%	88.81%	92.60%
	%red2	94.23%	97.89%	n.c.
7	Convrt	0.39	1.7	7.4
	Primal	0.13	0.40	1.8
	Total	1.27	3.11	10.94
	%red1	74.95%	86.25%	91.55%
	%red2	94.77%	98.06%	n.c.

Note: All times are given in seconds. %red1 was calculated from "Up. Bound" column of Table 4; %red2 was calculated from "No Bound" column of Table 4. n.c. = not calculated, generally because the comparative problem or this problem was not or could not be solved because of technical difficulties in using LAMP.

Several ways to improve the time come immediately to mind. The first is to modify the setup of the operating system so that a single user is using all the available disk drives; then several, in our case four, controllers are employed instead of one. The reduction is close to 75 percent. The second way is to write the constraint set in the binary output format of the data input routine (convert) instead of writing it as ASCII card images. This would eliminate the convert step of the LP completely, which, as shown in Tables 2 through 6, is the most time-consuming portion of the LP. Work on this problem is continuing. The third way is to attack the number of hulls that must be checked (Table 2). Three- and four-hulls are relatively simple to identify and then screen out. Work on this algorithm is also in process. At this time, we see no way to further tighten the upper bound.

4. BIT-CODED LISTS

It is sometimes desired, as in this problem, to repeatedly represent classes that are subsets of a fixed and ordered set of items, such as a list of n points. One convenient method of doing this is to designate one or more computer words as potentially defining the whole original set; this would be done by setting the n bits all to 1 in order to represent the entire given set. Subsets

could then be represented by setting the bit corresponding to each element to 1 if it is a member of the subset or to 0 if it is not.

This procedure has the benefit of greatly reducing storage requirements; for instance, any subset of 32 elements can be represented (in a 32-bit computer) in one computer word, while other representations would require either 32 words or a variable number of words equal to the cardinality of the subset. There are many situations in which this representation presents great computational advantages. A few examples are discussed below, including three that occur in the programming for the problem presented in this paper.

The utility of bit representation depends on the operations that are available. Most modern computers, including AT-class machines with the Microsoft Fortran Compiler, have a suite of logical operations that are very effective for these purposes. If each word is regarded as containing a set of bits that are either "true" (with a value of 1), or "false" (with a value of 0), three of these operations perform logical operations of pairs of sets. Within Convex (1988a) Fortran version 5.0, JIAND is an intrinsic function that produces a new word representing a set which itself is the intersection of two sets (words). JIOR produces the union. JIEOR, which is the "exclusive or," produces the disjunction — i.e., the union minus the intersection of the two original sets. The unitary operator JNOT produces the complement of the original set (but also complements all the bits that are not used in representing the given universe). Two of these functions (JIAND and JIEOR), plus bit testing, are used when this step is incorporated. However, the most important application does not require them, and depends on the simple fact that identical subsets have identical representations that are, therefore, numerically equal. Care must be exercised in setting the sign since -0 is not numerically well-defined. Since, as described above, one-hulls and two-hulls have been removed, all hulls can be represented by some positive or negative number: a positive number if destination number 1 is not in the set and a negative number if destination 1 is in the set.

Bit Pattern Matching

Harris (1978) used bit-coded lists to represent a tree in the solution of the transportation problem of linear programming, the Hitchcock Problem. Most representations of this tree look toward the root node; the few that look away from the root node have been somewhat awkward to use. This new representation mode (bit-coded lists) proved very efficient and economical of storage.

In the problem discussed here, most of the original work revolved around checking hulls as they were discovered to determine whether they had been previously discovered or were new, unique hulls. This is a non-trivial problem for large n and p: for example with $n = 30$ and $p = 5$, one would explore about 1,125,000 hulls to retain about 130,000 unique hulls.

For this problem, the original description of these unique hulls were kept in lists of 30 words each. Each size of hull (number of members) was identified. The entire list of unique hulls had to be searched to find hulls of the same size. The length of the list grew as new members were added so that, on average, each of the 1.125 million candidates called for 65,000 queries —a total of 72 billion logical comparisons. These queries discovered, on average, about 2.3

thousand (65/29 thousand) hulls of the same size. For hulls of the same size, membership lists had to be compared to determine if they were identical. On average, slightly more than half of the identically sized hulls had to be compared point by point, since about half the new hulls would be rejected, on average, about halfway through the list of unique hulls. The testing required far less than 30 comparisons since one mismatch would reject identity. Conservatively, the average number of comparisons may have been about 10. Thus, these comparisons for matching hulls involved another 1.5 billion tests.

The membership list of each hull was bit-coded into a single-integer word. The integer value of this number was unique to one particular hull membership. This reduced the storage by a factor of 30 since a single word of storage replaced 30 required in the old method. Comparisons were reduced by a minimum factor of 30 since the old method required at least two comparisons, and now no more and no less than one comparison was always required.

The organization of the history was also revised. As discussed above, one-hulls and two-hulls were enumerated rather than searched for. Therefore it was not necessary in this step to record them. The history at this point was an array of 20,000 by 30. Column 1 held all hulls of size 3, column 2 all hulls of size 4, column 3 all hulls of size 5, etc. Now only one column vector of integers (this was also a contiguous piece of hardware computer space), one integer per unique hull, had to be compared to the one integer representing one new hull. Comparisons were here again reduced by an average factor of 30 since only correct-sized hulls were tested.

Geometric Applications of Bit-Coded Lists

Our experience with the aforementioned processes prepared us for the discovery of some interesting geometric applications of bit-coding. These arose when we decided to pre-calculate all of the feasible hulls with three or four members for later inclusion in the LP. In each significant case we undertook considerable development and experimentation before we arrived at the present results.

Three destinations define a triangle, and to be feasible they must not enclose any other destination. Four-hulls can be of two types, with either three or four destinations defining their convex hulls; but neither type may enclose a fifth destination. The methods to be described depend for their efficiency entirely upon bit-coded lists.

The computational complexity of finding both feasible triplets and feasible quadruplets is proportional to n^3 for many practical problems. The method for triplets involves examining all of the $O(n^3)$ possible triplets. For feasible triplets, the number of operations is proportional to n, while for non-feasible triplets it is fixed. The method for quadruplets examines all triplets and finds feasible quadruplets from them. The computational complexity then depends on the number of feasible triplets, which we estimate from empirical experience to be proportional to n^2.

Of course this conclusion is arrangement dependent. We can identify the bizarre extreme case, where every node is on a circle, or on any other convex closed curve. In this case, any subset of n points is a feasible hull, and the

number of feasible quadruplets is on the order of n^4, while the total number of feasible hulls is 2^{n-1}. In this case the linear program for a 30-element problem would have to deal with over a billion columns.

There are $[n(n-1)(n-2)/6]$ triplets that can be found using n given points, no three of which are collinear. Each triplet defines three lines, segments of which make a triangle. A feasible triplet has no interior points from the original set. At first glance, it might appear that it can be easily determined whether there is an interior point by applying a "point-in-polygon" routine to the $(n-3)$ points not defining each triangle. This involves a sizeable multiple of $(n-3)$ operations for each triplet and is not economical.

A different procedure, described here, is followed, but first an important notational convention must be described: if i, j, k, and l stand for destinations or points, then for pairs, $i<j$, for triplets, $i<j<k$, and for quadruplets, $i<j<k<l$.

The equations of the lines joining the order n^2 pairs of points are calculated, each requiring a fixed number of operations. Substituting the coordinates of every point into this equation gives quantities proportional to the distance of each point from the line, and of opposite sign for locations on opposite sides of the line. This requires a small multiple of n operations for each line. The original pair of points are at zero distance, but it is quicker to calculate their distances than to skip them by testing every point twice.

As the distances from a given line are calculated, two bit-coded lists are built, one for the points on each side of the line. These bit-coded lists are then used to determine whether a given triplet has an interior point. For each of the three lines determined by two of the three points, it is determined which side of the line the third point is on and the bit-coded list for this side of this line is selected. In two operations, using JIAND, it is determined whether there is any point which is on all three lists. If the intersection of words is not equal to zero there is an included point. The triplet is rejected.

This procedure is repeated for all triplets, and the successful ones are recorded, once again in a bit-coded list. Each word of this list corresponds to an (i, j) pair and each of n bits is 1 for a k belonging to a feasible triplet, and 0 otherwise. This list is used further in finding feasible four-hulls.

A feasible hull with four members may be described as composed of four triangles. If there are four members of its convex hull, so that none is interior, the four triangles decompose the quadrilateral in two ways (see Figure 1). If one point is interior, so that the group has three points on its convex hull, the group is defined by one triangle, which is decomposed into three smaller ones (see Figure 2). These situations enable analysis of two cases that arise in examining all triplets using the lists made in the previous step. In Case I, (i, j, k) does not form a feasible triplet. It is now necessary to determine if the infeasibility consists of one, or more than one, interior point(s). If it consists of one point, l, it is a feasible quadruplet; it is infeasible otherwise. To determine this, the three words that denumerate all the points l with the lines defined by the point pairs (i, j), (i, k), (j, k) are considered. The intersection of these words is formed with two JIAND operations. If the resulting list is zero, there is no l that forms a feasible triplet with each of the three lines; thus there is more than

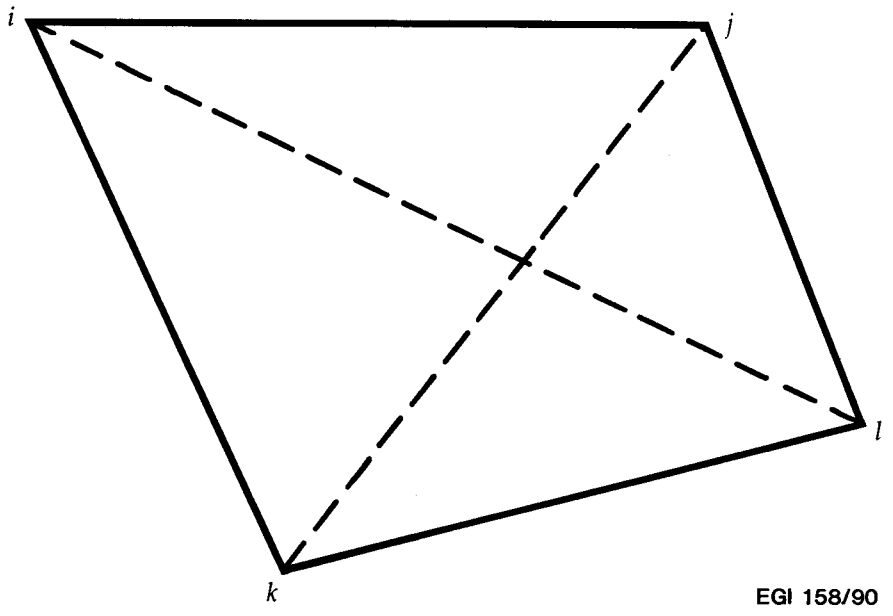

FIGURE 1. A Quadrilateral Defined by the Sum of Either of Two Pairs of Triangles: *ijk* + *jkl* or *ijl* + *ikl*

one interior point in (*i, j, k*) and this triplet is abandoned as a potential part of a feasible quadruplet. If it is non-zero, the locus of the single 1-bit is found and recorded as *l* in the quadruplet (*i, j, k, l*). Locating this 1-bit can be done with masks in a binary search that is proportional to (log *n*); this, for our purposes, is fixed at five steps.

Case I (described above) is more common than case II, where (*i, j, k*) is a feasible triplet, in the ratio of about *n*:1, given the assumption that the number of feasible triplets is less than n^2. The subsidiary case where an infeasible triplet is the hull of a feasible quadruplet is also relatively rare, so that this part of the computation is small. Case II involves more work, which is offset by the fact that it is less common.

In case II, any given (*i, j, k*) forms a feasible triplet. In this case there are two sub-cases:

(a) Any exterior point that forms a feasible triplet with at least one pair out of (*i, j, k*), and that forms a convex quadrilateral, is part of a feasible quadruplet.
(b) Any point that would not form a convex quadrilateral, if it forms a feasible triplet with *two* pairs of (*i, j, k*), makes a feasible quadruplet. Applying the rule from case (b) does not exclude any valid examples from case (a) (see Figure 3).

Three new words are now formed by intersection, using the JIAND function on the sets of valid triplets as given by the bit-lists for the pairs of pairs [(*i, j*), (*i, k*)], [(*i, j*), (*k, j*)], and [(*i, k*), (*j, k*)]. Each of the new words contains a list of the valid *l* indicators, but there may be duplication. The bit-wise union of these three words is then formed, using JIOR. This new word is a list of exactly those

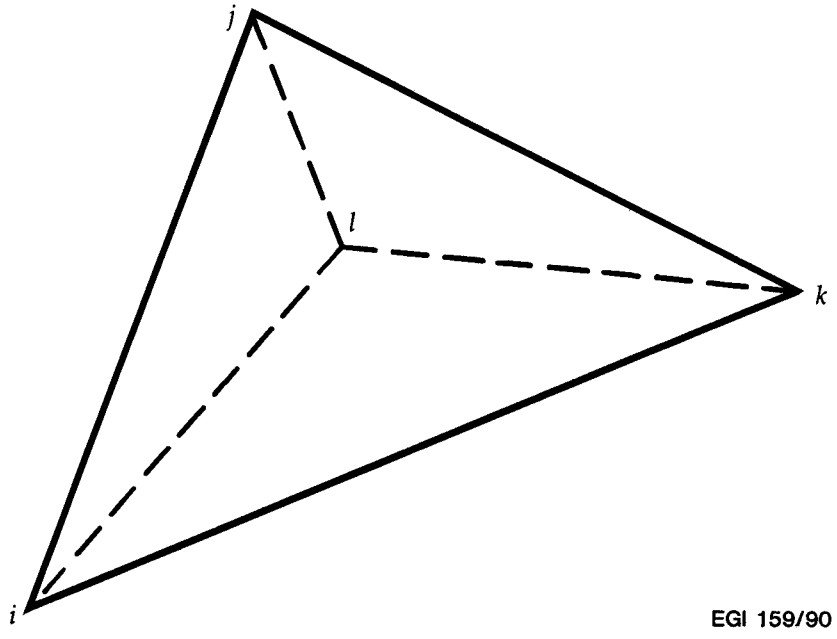

FIGURE 2. Four Points in Four Triangles as a Feasible Quadruplet

l-values that form feasible quadruplets with (i, j, k). Finally the resulting word is decoded, finding the loci of all non-zero bits.

Case II thus arises in about $1/n$ of all triplets, but requires five operations for comparisons and n operations for decoding, so that the work is of the order n^3. Case I arises in most triplets $O(n^3)$, but requires, on average, a fixed number of operations per case so that it is also of $O(n^3)$.

Summary and Extensions

This experience in finding very fast methods for analyzing what at first seemed to be difficult or intractable problems suggests that there may be many other geometric problems that could benefit from a similar approach. These may include problems involving networks and trees, which lead into the realm of graph theory. Of course, these methods are relatively more efficient for large rather than small problems, so that they are, in general, applied, rather than theoretical, considerations. At the same time, the underlying structure of the problems, which for success must be discovered in every case, may have some theoretical content, and perhaps may even be non-trivial.

5. VECTOR FORTRAN CODING

The Convex 240 is capable of vector processing but not of parallel processing. In many cases the current problem could profit from parallel processing, for example in investigating two complementary hulls at the same time. In any case, using vector processing significantly improves the overall operation of the program.

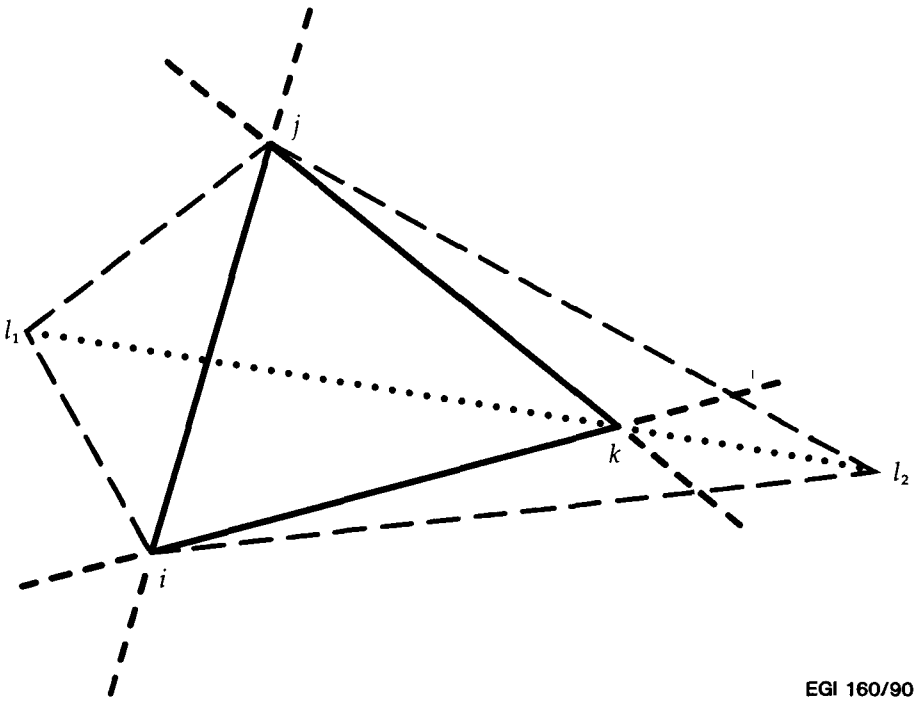

FIGURE 3. Other Configurations

Note: Triangle *ijk* is a feasible triplet if ijl_1 is a feasible triplet and $ijkl_1$ is a feasible quadruplet (ikl_1 and jkl_1 are also feasible). If jkl_2 and ikl_2 are feasible triplets, $ijkl_2$ is a feasible quadruplet.

Veclib

Veclib (Convex 1988b) is a library of vector processing function subroutines callable in a Fortran program. One of them, IISVEQ, compares a vector of integers with an integer scalar. Upon equality it returns the position of the integer in the vector. If no equality is found, it returns zero. After being called, the result is tested, and if it is greater than zero, the current hull is non-unique. Thus each 128 elements (each element representing a complete hull) are compared in the same or less time than one size-of-hull test in the old method. After the full length of the vector has been investigated (or as soon as equality is found) IISVEQ returns. IISVEQ's result must be compared only once. The number of operations is thus 1/128 of the reduced number of operations required in bit pattern matching (see above).

Profiling

Writing an effective Fortran program for vector processing entails good judgment in structuring loops and placing logicals. Software to profile the program is indispensable. Profiling a program entails obtaining counts of how many times a subroutine is called and how much time, and what percent of the total time, is spent in each subroutine during execution. More detailed profiling provides the same information at the level of every loop in the program. Profiling the subroutines first indicates which subroutines might be significantly

improved. Profiling the loops of such subroutines concentrates one's attention on the least efficient portions of the program. Relatively simple adjustments can then result in marked improvements in operation. For example, sometimes one large loop can be replaced by a combination of one slightly smaller loop and a very small loop containing a logical. Since an internal exit from a loop generally prevents it from being handled in a vector mode, total execution time is considerably diminished. Time improvements may be very significant. In the preparation program, one such modification of TWAIN (Ostresh 1973b) reduced its execution time by more than 10 percent.

Vector Programming

In order to compare the vector program with a program that did not include vector processing, the program was recompiled with a normal Fortran routine to replace IISVEQ and optimization was turned off. As an economy measure not all 20 programs were resolved. Rather, the preparation program was run for $n = 20$ with $p = 3, 4, 5, 6, 7$ and for $p = 3$, $n = 15, 20, 25, 30$. With $p = 3$ and $n = 15$ the program took 1.28 times as long, $n = 20$ took 1.36 times as long, $n = 25$ took 1.37 times as long, and $n = 30$ took 1.67 times as long. When $n = 20$ and $p = 4$, it took 3.45 times as long, $p = 5$ took 6.89 times as long, $p = 6$ took 9.36 times as long, and when $p = 7$, it took 10.71 times as long as the times reported in Table 5. In addition, several linear programs were solved with the old non-vector processing version of LAMPS. The values of n and p had no effect on the time. On average, in the unvectorized version of Convert takes 1.27 times as long; Primal takes 1.73 times as long, and total time is 1.42. Vector coding demonstrates very effective improvements, particularly as the problem grows larger.

6. OVERALL IMPROVEMENT

The overall comparison of the improvements discussed here for the preparation program was done by comparing results with the times reported by Rosing (1992a) for the same problems. Based on the speed of generating and investigating convex hulls, the improvement is on the order of 15- to 30-fold, depending on problem size.

The improvement of the LP is the improvement discussed for vector programming (Section 5, above), plus the improvement caused by the exclusion of convex hulls that cannot be in the optimal solution at the preparation step. This drastically reduces both the Convert and Primal times.

7. FUTURE DIRECTIONS OF RESEARCH

This method provides a way to solve the (Generalized) Multi-Weber Problem in at least small and moderate cases. The screening out of three-hulls and four-hulls, mentioned above, must still be fully implemented. This must greatly reduce the total number of hulls that will found by TWAIN (or any of the three available algorithms). There will be a small increase in time from adding this step since it increases the complexity of calculation of the initial stage of the preparation program. This step must be carried out only once for all values of p (for any given data set) since it finds all one-, two-, three-, and four-hulls.

Later, however, there is no increase in the amount of computation to screen out one-, two-, three-, and four-hulls instead of only one-hulls and two-hulls.

In any particular data set, this screening step need be done only once; the information can be permanently held and reused for various values of p. It is also possible to develop the constraint set for $p + 1$ from p by increasing the amount of data written to disk. The history of unique hulls would have to be written out and an identification process would have to be developed for determining which level of the program developed each hull. Whether this would save time overall is, however, an uninvestigated point.

Implementation and use of the new veclib (Convex 1989), which permits double precision integers, will then permit the solution of problems up to a size of 64. A genuine test of the effectiveness of heuristic solutions to the Multi-Weber Problem will then be possible.

ACKNOWLEDGMENTS

The authors would like to thank the two guest editors and the editor of *Papers*. Their work has considerably improved the readability of this paper.

This paper was prepared while the first author was in the Department of Geography, University of Manitoba, and a Fellow, G.W.C. Whiting School of Engineering, Johns Hopkins University, Baltimore, Maryland, USA.

REFERENCES

Abramowitz, M., and Stegun, I. A. 1972. *Handbook of mathematical functions with graphs and mathematical tables.* 10th ed., with corrections. New York: Wiley Table 24.4, 835.

Advanced Mathematical Software. 1986. *Lamps users guide.* Version 1.56. London: Advanced Mathematical Software Ltd.

Convex. 1988a. *Convex Fortran User's Guide.* 8th ed. Houston, Texas: Convex Corp. ug-C-10.

Convex. 1988b. *Convex Veclib User's Guide.* 3rd ed., revision 2. Houston, Texas: Convex Corp. 2-15, 2-16.

Convex. 1989. *Convex Veclib User's Guide.* 4th ed. Houston, Texas: Convex Corp. ug-C-17.

Cooper, L. 1963. Location-allocation problems. *Operations Research* 11: 331–43.

Cooper, L. 1964. Heuristic methods for location-allocation problems. *SIAM Review* 6: 37–53.

Drezner, Z. 1984. The planar two-center and two-median problems. *Transportation Science* 18: 351–61.

Fisher, W. D. 1969. *Clustering and aggregation in economics.* Baltimore, Maryland: Johns Hopkins University Press.

Galvani, L. 1933. Sulla determinaxione del centro di gravità e del centro mediano de una popolazione con applicazioni alla popolazione Italina censita il 1° dicembre 1921. *Metron* 11: 17–28.

Garey, M. R., and Johnson, D. S. 1979. *Computers and intractability: a guide to the theory of NP-completeness.* New York: W. H. Freeman.

Hakimi, S. L. 1964. Optimum distribution of switching centers and the absolute centers and medians of a graph. *Operations Research* 12: 450–59.

Hakimi, S. L. 1965. Optimum distribution of switching centers in a communication network, and some related graph theoretic problems. *Operations Research* 12: 462–75.

Harris, B., Farhi, A., and Dufour, J. 1970. *Experimental investigations of a combinatorial problem.* Philadelphia, Pennsylvania: University of Pennsylvania, Institute for Environmental Studies Discussion Paper.

Harris, B., Farhi, A., and Dufour, J. 1972. *Aspects of a problem in clustering.* Philadelphia, Pennsylvania: University of Pennsylvania, Institute for Environmental Studies Discussion Paper.

Harris, B. 1978. A new algorithm for tree modification in the primal transportation problem. *Transportation Science* 12: 271–76.

Keane, M. 1975. The size of the region-building problem. *Environment and Planning A* 7: 575–77.

Kuenne, R. E., and Soland, R. M. 1970. *The multisource Weber problem: exact solutions by branch and bound.* Institute for Defense Analysis, Economic Series.

Kuenne, R. E., and Soland, R. M. 1972. Exact and approximate solutions to the multisource Weber problem. *Mathematical Programming* 3: 193–209.

Kuhn, H. W. 1967. On a pair of dual nonlinear programs. In *Non-linear programming*, ed. J. Abadie, pp. 37–54. Amsterdam: North Holland Press.

Kuhn, H. W., and Kuenne, R. E. 1962. An efficient algorithm for the generalized Weber problem in spatial economics. *Journal of Regional Science* 4: 21–33.

Levy, J. 1967. An extended theorem for location on a network. *Operational Research Quarterly* 18: 433–522.

Love, R. S., Morris, J. G., and Wesolowsky, G. O. 1988. *Facilities location: models and methods.* Amsterdam: North Holland.

Miehle, W. 1958. Link-length minimisation in networks. *Operations Research* 6: 232–43.

Ostresh Jr., L. M. 1973a. MULTI. In *Computer programs for location-allocation problems*, eds. G. Rushton and J. A. Kohler, pp. 29–53. Iowa City, Iowa: University of Iowa, Department of Geography Monograph No. 6.

Ostresh Jr., L. M. 1973b. TWAIN. In *Computer programs for location-allocation problems*, eds. G. Rushton and J. A. Kohler, pp. 15–28. Iowa City, Iowa: University of Iowa, Department of Geography Monograph No. 6.

Ostresh Jr., L. M. 1973c. ALTERN. In *Computer programs for location-allocation problems*, eds. G. Rushton, and J. A. Kohler, pp. 55–66. Iowa City, Iowa: University of Iowa, Department of Geography Monograph No. 6.

Ostresh Jr., L. M. 1975. An efficient algorithm for solving the two center location-allocation problem. *Journal of Regional Science* 15: 209–16.

Ostresh Jr., L. M. 1978a. Convergence and descent in the Fermat location problem. *Transportation Science* 12: 152–64.

Ostresh Jr., L. M. 1978b. The multifacility location problem: application and descent theorems. *Journal of Regional Science* 17: 409–19.

Rosing, K. E. 1991. Classical location theory: solutions to the generalized multi-Weber problem. In *The Dauphin papers: research by prairie geographers*, eds. J. Welsted and J. Everitt, pp. 119–29. Brandon, Manitoba: University of Brandon.

Rosing, K. E. 1992a. A method for optimal solutions to the (generalized) multi-Weber problem. *European Journal of Operations Research* 57 (forthcoming).

Rosing, K. E. 1992b. The optimum location of steam generators in large heavy oil fields. *American Journal of Mathematical and Management Sciences* (forthcoming).

Weiszfeld, E. 1937. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tôkohu Journal of Mathematics* 43: 355–86.