A Spatial Branch-and-Bound Algorithm for Some Unconstrained Single-Facility Location Problems

by

Richard Edahl

EDRC 05-34-89
Carnegie Mellon University

A SPATIAL BRANCH-AND-BOUND ALGORITHM FOR SOME UNCONSTRAINED SINGLE-FACILITY LOCATION PROBLEMS

Richard Edahl
Engineering Design Research Center
Carnegie Mellon
29 March 1989

Abstract

A globally convergent spatial branch-and-bound algorithm is given here which is shown to be useful on several unconstrained single-facility location problems. These include the min-sum, min-max, and max-min problems with cost functions that are continuous and non-decreasing in distance. For the special case of the min-sum problem with Euclidean metric and power cost functions, a quadratic lower-bounding function is developed that results in a convergence rate superior to that of using a simple lower bounding function from the Big Square-Small Square algorithm of Hansen et al.

1. Introduction

We examine here the utility of a general spatial branch-and-bound algorithm for a variety of unconstrained single-facility location problems. While many of these problems have efficient algorithms that converge to the globally optimal solution, there are still several problem classes for which this is not true. The exceptions arise usually for non-convex problems, possibly resulting in local optima that are not global optima. We give below several formulations of these non-convex location problems and implementations of the spatial branch-and-bound algorithm for them. These formulations are not meant to be exhaustive, but only examples.

The simplest formulation of the Weber problem is

$$\underset{x\in}{\min}\ \pounds\ w/Cx^1-*])^2 + (*^2-z,W$$

where the Wj's are positive weights and each $z^\wedge(z^1jjf)$ is a distinct point (destination, sink, resource location, etc.). The problem then is to find the location x* to minimize total transportation costs from x* to the various sinks, etc. (Transportation cost is linear in Euclidean or straight-line distance.)

The less restrictive form of this single-facility location problem that is examined here is the general min-sum location problem:

$$\text{P1)}\quad \min_{x \in R^n} \sum_{j=1}^{J} f_j(d(x,z_j))$$

where the Zj's are points in R^n, d(y) is the Euclidean metric, and each fj is a continuous non-decreasing function of the distance, d. If the fj are convex, P1 can be shown to be convex and algorithms exist that are globally convergent for it. (See e.g. Katz [8] and Cooper [3].) Weaker restrictions on fj permit P1 to have non-optimal local minima, hence the best that these algorithms can demonstrate is local convergence. (See e.g. [8].) The algorithm developed below is globally convergent for P1.

The second formulation considered here is the min-max location problem:

$$\text{P2)}\quad \min\quad maxfj(d(x_f Zj))$$

For P2, it is desired to find the location of x that minimizes the maximum cost or penalty of distance to the points Zj. This formulation may be appropriate for example for some emergency services (e.g. location of paramedic stations) where it is desired for the facility not to be too far from any of a set of locations. Forms of this problem were examined in Drezner and Wesolowsky [4] and Chen [2], among others. Using Euclidean distance for d, P2 is a convex programming problem if the f_j cost functions are convex in d. However, if the f_j functions were only continuous and non-decreasing in d, then P2 may not be convex, but quasi-convex.

The third formulation considered here is the max-min location problem:

P3) $\underset{xeS}{max}\ \underset{0 \leq j \leq J}{minfffl(xap)}$

where S is some box in R^n. For P3, it is desired to find the location for x to maximize the minimum penalty of distance to the points Zj. This formulation may be appropriate for example for problems to do with noxious facilities (e.g. toxic waste landfills) where it is desired that the facility not be too close to certain locations. One form of this problem was examined in Drezner and Wesolowsky [5]. This problem is not convex, and subject to many local optima. The optimal solution will be unbounded if the feasible region is not compact. However, due to the non-convexity of the problem, the optimal solution is not necessarily a boundary point of S.

Section 2 of the paper contains some notation and definitions and concludes with the algorithm. The convergence proof for the algorithm is in Appendix I. It is shown in Section 3 how the algorithm may be applied to P1 using as examples the Euclidean metric with power cost functions. In Section 4 is givens examples of using the algorithm on the formulations P2 and P3.

It should be kept in mind that this algorithm, while globally convergent for a wide range of location problems, cannot be called 'fast^1. One practical suggestion might be to use this algorithm until a 'good^1 approximation to the optima is found, and then to use a faster locally convergent algorithm to improve the solution. (Since at each iteration both a lower and upper bound for the optimal solution are given by the algorithm, some degree of confidence can be given to this procedure.)

2. Algorithm

The algorithm developed here can be considered either a spatial branch-and-bound algorithm (see e.g. Mitten [10]) or a variant of a sequential Lipschitz grid algorithm (see e.g. Shubert [11] and Meewella and Mayne [9]). The main difficulties in using such algorithms stem from the inability of applying the algorithm (no good Lipschitz number estimate or no good lower bounding function), from combinatorial explosion coming from the dimension of the space, or from the nature of the individual problem. For the problems considered here, it is shown that a good lower bounding function is available. The algorithm given here has some similarities to the BSSS (Big Square-Small Square) algorithm given in Hansen et al [7]. A more detailed comparison with the BSSS algorithm is given in Section 3. We also give examples of using the algorithm on problems in spaces of up to 6 dimensions.

For the problem

P) *minF(x)*

we will consider x* € S to be an e-solution if

$$F(x^*) - \varepsilon \leq F(x) \quad \forall x \in S.$$

We shall deem the problem solved if, for a given e > 0, an e-solution is found. Also let P be the optimal value of the objective function in P.

We present here an algorithm which finds at each iteration, a lower bound for F\ The lower bound is obtained by partitioning the set S into a set of cells and finding a lower bound for the minimum of the function in each cell. This lower bound is calculated by the use of lower bound functions. The cell associated with the lowest of the lower bounds is then partitioned into smaller cells and the iteration is continued.

2.1. Partitioning or Branching

The details of the partitioning are presented in this section. The general description and notation of the partitioning is taken from [9].

Let R_p be a ceH defined by

R_p *m [xe* R^n*\oP^<x^* ≤ $p^{*1'}$, /=l,...,n}

Then, a suitable partitioning is taken to be a collection of nonoverlapping cells $\{Rp\}_{p \in p}$ whose union covers S, where P is an index set. That is, $\{R_p\}_{pe P}$ satisfies the following two conditions:

(a) $V\{R_p n R_q) = 0$, Vp,q e $P,p*q$, where V is Lebesgue measure;

(b) $S=v_{peP}R_p$

For ease of notation, we shall now consider a general cell

$$R \equiv \{x \mid \alpha^i \leq x^i \leq \beta^i, \ i=1,\dots,n\}$$

The vertices of the cell are indexed as follows. Let

$$c(m) = (c^1(m),\dots,c^n(m))$$

be the binary representation of m-1, so that

$$m-1 = \sum_{i=0} 2^i c^{n-i}(m)$$

Let the 2^n vertices of the cell S be denoted by y_m,m=1,...,2^n. The vertices are numbered such that the components $\{y'_m \mid i=1,..,n\}$ of the vertex y_m are given by

$$y^i_m = \alpha^i[1-c^i(m)] + \beta^i c^i(m). \quad /=1\dots.n$$

where a' and p^j are as given in the definition of R above. In the two-dimensional case, we have the results shown in Table 2-1

m	1	2	3	4
$c(m)$	$(0,0)$	$(0,1)_2$	$(1,0)$	$(1,1)$^
y_m	(α^1,α^2)	$(\alpha^1,\beta\ )$	(β^1,α^2)	(β^1,β^2)

Table 2-1: Cell Division

Definition 1: Let R be an arbitrary cell in R^n. Then the cell dividing function $\dot{\phi} = \{\phi_1,$ <Φ2 - <M Operates 2" cells <t>$_m$(^{R}). $^{m = 1}$ »^{2}.....^{2}* defined by:

$$\_^n | \alpha^{m,i} \le x^i \le \beta^{m,i}, \quad i=1,\ldots,n\}$$

where

y˙ = {pW}/2

$\alpha^{m,}$i = ,ra$^{4'}$, if $c^q(m)=0$,
l y, otherwise,

$\beta^{m,}$i = rp', ifc«(m)=I,
ly, otherwise,

Note that

0u$_{m=1}$^»(|)$_m$(/?) = /?

«) V{<|>$_l$(/?)r^<>$_{/n}$(/?)} = 0 for *l*m.* where V is Lebesgue measure;

Notation:

$$\phi^{l+1}(R) \equiv \phi\{\phi_1(\phi^{l-1}(R)),\ldots,\phi_p(\phi^{l-1}(R))\} \equiv$$
$$\{\phi_1(\phi_1(\phi^{l-1}(R))),\ldots,\phi_1(\phi_p(\phi^{l-1}(R))),\ldots,\phi_p(\phi_1(\phi^{l-1}(R))),\ldots,\phi_p(\phi_p(\phi^{l-1}(R)))\}$$

(e.g.ifn=1,l=1,
<J)^2(/?) = •{•$_1$(J?),$\phi_2(R)\} = \{\phi_1(\phi_1(R)),\phi_1(\phi_2(R)),\phi_2(\phi_1(R)),\phi_2(\phi_2(R))\}$

It is easily seen that <J>(R) yields $(2^n)'$ sub-regions.

2.2. Determining Lower Bounds

The algorithm given here operates by first dividing the region into subcells and finding both lower bound and upper bounds for F(x) in each of the cells. It is necessary that for each subcell R_p, the lower bound F^L(Rp) and upper bound $F^u(R_p)$ satisfy

$$F^L(/y \text{ £ } \min_{xeR_p} Fix) \le\!\!\!S\ F''(/y$$

The upper bounding function ought to satisfy also:

$$F^U(R_p) \le \max_{x \in R_p} F(x)$$

A simple upper bounding function (on the minimum value of F in R) is one that evaluates F at some x in R. The most promising cell R_k (with lowest $F^L(R_p)$) is chosen and further subdivided into subcells, again computing the $F^{L>}$s and $F^{u>}$s for these new subcells.

In order for this sort of algorithm to work, some restrictions on $F^L(R)$ are necessary. For example, if $F^L(R)=K$ (F^L is the constant function), and K were sufficiently small, while F^L would give a lower bound for F, such a lower bounding function would not help in finding a good x. Instead of directly giving restrictions for F^L, we give restrictions for an associated lower bounding function whose domain consists of discs rather than cells.

Definition 2: C(x,r) = {y|d(x,r) ≤ r; y *e S).* C(x,r) is a closed disc with center x and radius r.

We now give conditions for the lower bounding function over discs:

Definition 3: F(x,r) is a d.l.b.f. (disc lower bounding function) for F if

f)F(x,r)≤F(y) **∀ y ∈ C(x,r)**

if) VxeS, given any e>0,3 5(e) such that
F(X)-F(XS)<E V0≤r≤5(e)

Note that if F satisfies a Lipschitz condition,

3Lsuchthat\F(x)-F(y)\<Ld(x,y)   Vx.yeS,

then *F(x,r)* = *F(x) -rL* is a d.l.b.f. for F.

Note also that the definition of a d.l.b.f. is similar to that for uniform lower semi-continuity. In fact, if F is uniformly continuous (or uniformly lower semi-continuous), then there will exist a disc lower bounding function. The problem is to find it.

We define a cell covering function:

Definition 4: Let h(R) be the longest side of the cell R. (h(R) - max {P^{i}-ct^{1}}). Suppose there exists functions x(R) and r(R) so that

x(R) G R

R e C(x(R),r(R))

Then x(r) and r(R) define a cell covering function if there exist a non-negative monotonic increasing function g such that

r(R)≤g(KR)) Vr≥0
and
Urn g(h) =0

Frequently, the cell covering function may be defined by x(R) = (a + p)/2, r(R) = ||p-a||/2.

We define a cell lower bounding function by comparing it with a disc lower bounding function:

Definition 5: Let x(R) and r(R) define some cell covering function for regions R. Then F^{L}(R) is a c.l.b.f. (cell lower bounding function) for F if,

F(x(R\r(R)) ≤ F^{L}(R) ^ F(y) VyeR

Note that F(x(R),r(R)) is a cell lower bounding function. In fact, for many of the location problems, this is the 'default^{1} cell lower bounding function.

2.3. The Algorithm

The algorithm which we will give below may be likened to a branch-and bound procedure (see [10], or a Lipschitz procedure (see [11] and [9]). For branch-and-bound the idea is to divide the initial set into cells, obtain bounds and then choose the most promising of these cells for further subdivision. In the notation from above, we would divide S into several subcells R_p, p=1,..,P and find F^{L}(R_p]). Then take the R_k with the lowest F^{L}, etc. In this setting, the discs are used to find bounds for the cells. For the Lipschitz procedure, discs are scattered to cover S. The disc with the smallest $F^L(R_p)$ is removed and replaced with several smaller discs. In this setting, the cells and dividing function Φ serve to enable this disc scattering and replacement to be done in a regular manner.

ALGORITHM: (Assume e>0 is given)

Step 1 Divide (or approximate) S by cells R_p so that:

(a) $S=v_{peP}R_p$

(b) $V\{R_pnR_q\}$ = 0, *Vp,q e* P_9p*q, where V is Lebesgue measure;

Put the Rp's in list L1 in non-decreasing order of the $F^L(R_p)$. Set x* equal to the solution at min $F(y(R_p))$. ($y(R_p)$ is some point in R_p determined by a method special to the problem. I.e. $y(R_p) = x(R_p)$ may be used.)

Step 2 Remove R_k from the top of L1 and test whether

$F(x^*) - \varepsilon \le F(R_k)$.

If no, go to Step 3. Otherwise, Stop (x* is then an e-solution).

Step 3 Apply Φ to R_k. Perform the following loop on m, m=1,..,2^n:

If *FiyWJRJ)* < F(**), set *x* = y^JR,))*

Insert $\$_m(R_k)$ into LI,maintaining non-decreasing order of the F^u*s.*

Go to Step 2.

The above algorithm has two basic elements -1) the bounding function, and 2) the dividing or partitioning method. The bounding function is generally independent of S and is frequently, as we shall see in Section 2, may be independent of the metric space. For example, if F is Lipschitz, F(x, r) = F(x) - rL is a valid l.b.f. for any space in which L is Lipschitz number for F. Also, if the space is R^n, the l.b.f. is generally independent of what (compact) subset comprises S.

2.4. Examples

Before going on to the location problems, we first give the results of using the spatial branch and bound algorithm on three common non-location examples. These are the same examples as used in [9]. The second example is Branin's function [1] and the third example is Goldstein's and Price's function [6]. The lower bounding function for each example used the Lipschitz number, which was taken from [9].

$A(.x,y),r)=Axy)-rL$

$f_1(x,y)=x^2+y^2$

$S=\{(x,y)| -1\leq x\leq 2; -.5\leq y\leq 1\}$

$f_1(x^*,y^*))=0$

$L=\sqrt{20}$

$f_2(x,y)=(y-5x^2/4n^2+5x/n-6)^2+l0(l-V\&K)cosx+10$

$S=\{(x,y)| -5\leq x\leq 10; 0\leq y\leq 15\}$

$/_2(x*,/)=0.397887$

$L=100$

$f_3(x)=[1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)][30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)]$

$S=\{(x,y)| -2\leq x\leq 2; -2\leq y\leq 2\}$

$f_3(x^*,y^*)=3$

$L=960000$

Results are given in Tables 2-2, 2-3, and 2-4. Note the poor performance of the algorithm on these problems. The experience here is similar to that for Meew's algorithm. In each case, even after 2000 iterations, the difference between the lower and upper bounds was significantly greater than the improvement in the objective function from the early iterations to the final ones. This would not give one a great degree of confidence that significantly better solutions did not exist (when in fact they do not). Hence, for these problems, the lower bound is not very useful.

In these tables, 'CELLS' refers to the number of cells that are active at a certain iteration. That is, it is the number of cells for which the lower bound is less than the value of the best point found so far ("UPPER BOUND'). 'LOWER BOUND[1] refers to the smallest lower bound of all the current cells. Note that for these problems that the number of active cells increases roughly linearly with the number of iterations.

ITERATION	CELLS	LOWER BOUND	UPPER BOUND
1	4	-3.6719d+00	7.8125d-02
2	7	-3.2969d+00	1.9531d-02
5	14	-1.7969d+00	4.8828d-03
15	37	-9.1797d-01	1.2207d-03
34	87	-4.6387d-01	3.0518d-04
86	201	-2.3315d-01	7.6294d-05
100	228	-2.2089d-01	7.6294d-05
500	1057	-5.3024d-02	4.7684d-06
1000	2081	-2.6889d-02	1.1921d-06
1500	3065	-1.8718d-02	1.1921d-06
2000	4073	-1.3617d-02	2.9802d-07

Table 2-2: Function 1

ITERATION	CELLS	LOWER BOUND	UPPER BOUND
1	4	-5.0787d+02	2.2456d+01
2	7	-5.0204d+02	1.3286d+00
20	59	-1.3105d+02	6.7287d-01
21	62	-1.3071d+02	5.4097d-01
65	183	-6.5620d+01	4.1767d-01
100	279	-5.7671d+01	4.1767d-01
500	1272	-1.6109d+01	3.9880d-01
1000	2365	-1.0728d+01	3.9880d-01
1500	3483	-7.1052d+00	3.9880d-01
2000	4639	-5.8062d+00	3.9880d-01

Table 2-3: Function 2

ITERATION	CELLS	LOWER BOUND	UPPER BOUND
1	4	-1.3558d+06	1.8760d+03
2	7	-1.3555d+06	8.8725d+02
3	10	-1.3505d+06	3.2688d+01
6	19	-6.7863d+05	2.7512d+01
21	63	-3.3937d+05	1.2637d+01
22	66	-3.3936d+05	1.0394d+01
81	236	-1.6969d+05	4.7795d+00
100	293	-1.6954d+05	4.7795d+00
500	1478	-8.4073d+04	3.4466d+00
1000	2958	-6.1741d+04	3.4466d+00
1500	4399	-4.2180d+04	3.1127d+00
2000	5895	-4.1523d+04	3.1127d+00

Table 2-4: Function 3

3. Min-Sum Problem

The applicability of the algorithm is restricted to problems for which a lower bounding function (as in Def 5.) can be found. However there is a class of problems, those of the continuous location variety, for which the algorithm might prove useful. Suppose one wishes to solve

P1) *min F(x) = min* £ *fffXxj))*
X€ R" X€ R" " J^{-1}

where is the Euclidean metric, z; is a point in R^n, (a resource or market location), and f_j is a non-decreasing continuous function in its argument (unit transportation cost). Then the problem is to locate a facility to minimize transportation costs. It is a simple matter to show that

$$F(x,r) = \sum_{;=i}^{J} f_j(d(x, z_j) - r) \qquad \textbf{(1)}$$

isad.l.b.f. forF(x)

Lemma 6: Let

$$F(x)=\sum_{j=1}^{J} f_j(d(x,z_j)$$

where fj is non-decreasing and uniformly continuous and d(y) is a metric. Then F(x,r) in Equation 1 is a d.l.b.f. for F(x).

Proof:

0 ***d(x,zp ≤ d(y$zp+d(x,y)*** **by Triangle inequality**
≤d(y,zp+r VyeCM

-*ffd(x,zp-r) ≤fj<d(y,zp) Vye CM

Hence, ***FM*** **= J** ***fffo*p*** **-r) <-£** ***ffdiy, zp) = F(y) Vye C(x,r)***
$J \qquad J$
$j=1 \qquad j=1$

ii) Let e>0. Then/} uniformly continuous ->
for any e//>0, 3 5<e/7) such that VO≤r<5$_4$(e//),
$f_j(d(x,z_j)) - f_j(d(x,z_j) - r) < \varepsilon/J$

Therefore, for 0 £ *r* ^ m/n 8;<ε/J),

$$\sum_{j=1}^{J} |\backslash ffd(x,zp)\ \text{-}ffdix\text{^}jhr)] = \mathbf{F(x)} - F(x,r) < \mathbf{i\varepsilon}$$

Hence *F(xfr)* is a l.b.f. for *F{x).* Q.E.D.

Note: Iff$_j$(d) = w$_j$d,

$$FM = F(x) - \mathbf{r}\text{^}\ \mathbf{w_y} = \mathbf{F(x) - rW\ is\ a\ d.l.b.f.\ for\ F(x).}$$
$$7=1$$

One may use this d.l.b.f. as the c.l.b.f. (cell lower bounding function), but there is a slightly tighter bounding function available:

$$F(R) = \sum_{ft}^{J}, f_j(d(y_j, z_j)$$

where yj is the solution to

$$min \quad d(y_{jt}zp \qquad (2)$$

The computation of $y_}$ is a relatively simple procedure. (It is the finding of the closest point in a cell to a given point Zj).

The algorithm requires that the feasible region be divided in initial cells. The feasible region being R^n does not cause difficulties here since it is well established that the optimal solution must lie in the convex hull of the points Zj. Hence, choosing as the feasible region the smallest cell that contains all of the Zj points, and choosing this cell as the initial starting cell for the algorithm will suffice.

Some example problems using the cell lower bounding function of Equation 2 were run and the results are given in Section 3.3. These examples indicate that the algorithm with this basic lower bounding function may be useful especially in finding a good starting point for some other algorithm. That is, the algorithm slows down appreciably as the upper and lower bounds come together (that is, the asymptotic rate of convergence appears to be sublinear).

The Big Square-Small Square (BSSS) Algorithm of [7], also a spatial branch-and-bound algorithm, has many similarities with the algorithm given above. For the problem P1, it uses the lower bounding function given in Equation 2. The BSSS algorithm, while described for 2-dimensional problems, also allows for the feasible region to be a union of polygons. While there is nothing preventing the extension of the algorithm given above to this case also, it is not done here. One important difference for the BSSS algorithm is that instead of dividing the cell with the smallest lower bound estimate, all cells are divided (and the subcells with a value for the lower bounding function greater than the value of the best point found so far are discarded). The main computation justification given for this was the simplicity of implementation.

While the above lower bounding function will be sufficient for the general single-facility Weber problem, there are some important variants for which an improved lower bounding function may be found. We consider two such problems, each of which uses Euclidean distance as the metric. The improved lower bounding functions given below give far superior results than the lower bounding function in Equation 2 and for the BSSS algorithm on the appropriate problems. It is the use of these special lower bounding functions that is the main difference between this algorithm and the BSSS algorithm.

3.1. Euclidean Distance, Concave Cost Functions

Consider the following variant of problem P1:

P1 a) $$\min_{xeS} F(x) = \min_{xe\ S} JT\ fj(d(x_9zp)$$

Suppose that each of $f_j(d)$ cost functions are concave in d, besides being continuous and non-decreasing in d. While efficient algorithms exist for the case of fj(d) being convex in d, they are not guaranteed to be find the global optima should fj be concave.

The idea here is to determine, for each cost function fj, a lower bounding function over a cell R that is of the form a+bd^2. Were all the fj's actually such quadratics, then the problem would be trivial to solve:

$$FQ(x) = \sum_{7=1}^{J} (dj + bjd(x,zp^2) = \sum_{7=1}^{/} (dj + bp(x-z_j)^t(x-z_j))$$

$$= \sum_{7=1} a_j + x^t x \sum_{7=1} b_j - 2x^t \sum_{7=1} b_j z_j + \sum_{7=1} b_j z_j^t z_j$$

which is a simple quadratic function in x. To minimize this function over a cell R, one only has to solve:

$$\min_{xeR} \sum_{j=1}^{/} d_f + xfx \sum_{j=1}^{J} b- - 2^{**} \sum_{>=1}^{J} b_{?l} + \sum_{;=1}^{J} b^{\wedge}.\% \qquad (3)$$

Let:

$$A = \sum_{7=1}^{j} a_j \quad B = \sum_{7=1}^{J} b_j \quad Z = \sum_{7=1}^{J} b_j z_j/B$$

$$Bx^t x - 2Bx^t Z + \sum_{j=1}^{J} b_j z_j^t z_j$$

$$= i4 + Bx^t x - 2Bx^t Z + BZ^t Z - BZ^t Z + \sum_{7=1}^{J} b_j z_j^t z_j$$

$$= A + B\{x-Z)\backslash x-7) - B2Z + \overset{J}{\overline{J}}\ 6\wedge . \qquad (4)$$

Solving this problem is then equivalent to solving

$$\min_{xeR} d\{x7)$$

the solution of which is given by

$$x^{i*}_R = \begin{cases} \alpha^i_R & \text{if } Z^i \le \alpha^i_R \\ \beta_{>} & \text{if } Z^i \ge \beta^i_R \\ Z^1 & \text{Otherwise} \end{cases} \qquad (5)$$

Were F not of this special quadratic form, the quadratic lower bounding function for each fj could be used in the above analysis. A lower bound for F in R can then be found by plugging this value of x*$_R$ into the above approximation. All that remains is to find the ^ and bj coefficients (these depend of course on the cell R) to use in forming the quadratic programming problem in Equation 3.

Let U$_j$ and L$_j$ be given by (the subscripts on R are omitted here):

$$L_j = \min_{y \in R} d(y,z_j)$$

$$Uj{=}maxd(y,zp \qquad (6)$$
$$_{y€R}$$

That is, these are the minimum and maximum distances from Zj to points in cell R. (One may use d(x(R),Zj)± r(R)) instead).

aj and bj are then given by:

$$a_j = \frac{U_j^2 f_j(L_j) - L_j^2 f_j(U_j)}{U_j^2 - L_j^2}$$

$$b_j = \frac{f_j(U_j) - f_j(L_j)}{U_j^2 - L_j^2} \qquad (7)$$

Note that bj must be non-negative since fj is a non-decreasing function. Also note that

$fj{<}Uj) = a_j + b_j U_j^2$

and

$f_j(L_j) = a_j + b_j L_j^2$

Hence this convex quadratic approximation must bound the concave fj from below for Lj < d < Uj. That is,

$q \qquad b_j d^2 \le f_j(d)$

for $LjZd \le Uj$

Now plug the values for ^ and bj into Equations 4 and 5 to get x^*_R. Then F(R) is obtained by computing

$$F(R) = FQ(x\backslash) \quad \wedge A + Bix'z{-}mx\wedge{-}V{-}BZZ{+}\wedge bjZj'\underset{j=1}{\overset{J}{Zj}} \qquad (8)$$

3.2. Power Cost Functions

Consider this variant of problem P1:

P1b) $\min_{x \in S} \sum_{j=1} f_j(d(x,z_j))$

Suppose that each of fj(d) cost functions are of the form

$f_j(d) = W_j/ft, \; C_j > 0$

This form is discussed is for example in [3] and [2], and allow for more accurate transportation cost fitting than the simple linear function (Cj=1).

There are three cases to be considered. The first is $l \ge Cj > 0$, the second is $2 \ge c_y > 1$, and the third is $c_y > 2$. In considering these cases, the subscript of j will be suppressed for notational simplicity. When the aj and bj terms have been computed, they can be used to compute F(R) in the same manner as for P1 a.

3.2.1. Case 1

For I£c>0, f(d) would be concave, and the lower bounding function from the previous section could be used. Applied to this function, a and b would be given by:

$$' \; ' \; ' \; '' \; \frac{iU^2L^C-L^2U^C}{U^2-L^2}$$

$$b \quad \bullet \quad \genfrac{}{}{0pt}{}{U^C-L^C}{\mathrm{VTF}} \qquad (9)$$

3.2.2. Case 2

Here, the same lower bounding function as for the first case may be used. Since Case 2 is convex, it remains to be shown since that this function actually bounds f from below. We must show that

$$f(d)-q(d) = wd^c - w\frac{U^2L^C-L^2U^C}{U^2-L^2} - w\frac{U^C-L^C}{U^2-L^2}d^2 \geq 0$$

for – $L<d<U$

Consider the problem

$$\underset{L \leq d \leq U}{\mathit{minftd)}} \; - \; q\{d)$$

Since both f and q are differentiable over the interval, the minimum must occur either at an endpoint or at a stationary point. Since the value of the difference is 0 at the endpoints (which is acceptable), we need only concern ourselves with stationary points:

$$f(d)-q\backslash d) = \mathrm{wear}^1 -^2 \wedge \wedge 5 \wedge = ^\circ$$

$$\rightarrow$$

$$wed^{0\,''2} = 2\mathrm{vv}\frac{f/\wedge - if}{PL^2}\sim$$

To test whether this d is a relative maximum or minimum, compute the second derivative at its value:

$$f''(d)-q''(d) = wc(c-1)d^{c-2}-2w\frac{U^C-L^C}{U^2-L^2}$$

Using $wcd?\sim^2 = 2w\frac{U^C-L^C}{IP\text{--}L^2}$

$$f''(d)-q''(d) = 2w(c-2)\wedge \; \underset{-}{\sim}\underset{\_}{\wedge} \; \leq 0$$

Therefore, this d is a relative maximum for the difference function, which implies that that the difference function is positive over the appropriate range.

3.2.3. Case 3

Let us construct the quadratic as follows:

Let r be any positive distance. Let q(d) be constructed so that q is tangent to f at r. Hence, find a and b so that:

$f = wr^c = a + br^2$ / = *wcr*-*^{1} =2*br*

a ss $w>\frac{2-c}{2}$ -/* · 6 = $w\frac{c}{2}r^{c-2}$

$q(d) = w\frac{2-c}{2}^{*>\wedge\gg} + w\frac{c}{2}r^{c-2}d^2$

Clearly, the quadratic approximation equals f at r, but it remains to be shown for other positive values of d that the quadratic approximation is not greater than f.

It is sufficient to show that for d > r than the derivative of f is greater that the derivative of the approximation, and for 0 < d < r, the reverse is true:

$f(d) = wcd^{c-1}$ *q'(d) = wcr*~*^{2}*d*

$\frac{fid)}{q\backslash d)} = \frac{wcd^{c-1}}{wcr^{c-2}d} = \frac{d}{r}_{c-2}$

For c> 2 (Case 3), this last term is greater that 1 for d>r and less than 1 for d<r.

To determine which value of r to use to the lower bounding function, it is plausible to select r so that

or

$wU^c — w\frac{2-c}{2}r^\circ — w\frac{c}{2}r\wedge\wedge U^2 = wL^c — w\frac{2-c}{2}r^\circ — vv\frac{c}{2}r^{c-2}L^2$

$\rightarrow$

$U^c - \frac{c}{2}r^{c-2}U^2 = L^c - \frac{c}{2}r^{c-2}L^2$

$\rightarrow$

$\frac{2}{c}\frac{U^C - L^C}{U^2 - L^2} = r^{c-2}$

Plugging this value of r into the equation for q(d),

$$q(d) = w\frac{2-c}{2}r^c + w\frac{U^C - L^C}{U^2 - L^2}d^2$$

The difference between this formula for q(d) and the one for case 2 is simply the value of the constant term.

3.3. Examples

The spatial branch-and-bound algorithm was programmed with two different lower bounding functions. The first is the basic location lower bounding function as in Equation 1, and the second is the quadratic lower bounding function from the previous section. Several problem sets of different dimensions were randomly generated. For all problems, the location of the sources were taken to be uniform on the square (or cube or hypercube) with width 100. The distribution of the weights and exponents were also taken to be uniform, with the bounds as given in Table 3-1. The runs were all done on a MicroVax II workstation at Carnegie Mellon. The running times listed in the tables are all in seconds.

The first four problems differ in the range of the exponent, with M211 having concave cost functions, M212 nearly linear having some concave, some convex, and M213 nearly quadratic having all convex cost functions. M214 has a exponent range covering those for the first three problems. The second four problems differ from the first only in that they have double the number of sources.

The results for the basic location lower bounding function on four of the problems are given in Table 3-2. While the runs were not as bad as those in Tables 2-2, 2-3, and 2-4, they were also not very successful. As in those other examples, the number of active cells was increasing linearly with the number of iterations, and convergence of the algorithm was being bogged down because of that. It should also be noted that the algorithm performed better for the more concave cost functions (M211 rather than M212 or M232) and it is precisely this sort of problem that would cause the normal Weber problem algorithms difficulties.

The improved lower bounding function of section 3.2 was used on the problems of Table 3-1. The results for the 2-dimensional problems are given in Table 3-3. Here, the results are far more successful than for the basic algorithm. All the runs converged in reasonable amounts of time (given in the tables in seconds) and iterations. Note that the number of active cells did not grow with the number of iterations, but stayed in some interval. Here, the more quadratic the cost functions (M213 and M223), the quicker the convergence. This is logical since the lower bounding function used here incorporated a quadratic approximation.

Runs for the higher dimensional problems are given in Table 3-4. In these cases also, the algorithm converged with the number of active cells again remaining relatively stable for each problem. However, one the drawbacks of the spatial branch-and-bound algorithm can be seen here. The computation time necessary to achieve a given level of accuracy grows exponentially with the dimension of the space. Here, each additional dimension resulted in a 4.5-5.5 time increase in execution time.

Problem Set	Dimension	# Sources	Weight Range	Exponent Range
M211	**2**	**50**	**[1.10]**	**[•2,9]**
M212	**2**	**50**	**[1.10]**	**[.5,1.5]**
M213	**2**	**50**	**[1.10]**	**[1.5,2.5]**
M214	**2**	**50**	**[1.10]**	**[.2,2.5]**
M221	**2**	**100**	**[1.10]**	**[.2,9]**
M222	**2**	**100**	**[1.10]**	**[.5,1.5]**
M223	**2**	**100**	**[1.10]**	**[1.5,2.5]**
M224	**2**	**100**	**[1.10]**	**[.2.2.5]**
M311	**3**	**50**	**[1.10]**	**[•2,9]**
M312	**3**	**50**	**[1.10]**	**[.2,1.5]**
M321	**3**	**100**	**[1.10]**	**[•2,9]**
M322	**3**	**100**	**[1.10]**	**[-2,1.5]**
M411	**4**	**50**	**[1.10]**	**[.2,1.5]**
M511	5	**50**	**[1.10]**	**[•2,1.5]**
M611	**6**	**50**	**[1.10]**	**[.2.1.5]**

Table 3-1: Problem Set Characteristics

PROBLEM	ITERATION	CELLS	LOWER BOUND	UPPER BOUND	PRECISION	TIME
M211	1	4	1.7311d+03	3.5953d+03	.52	
	238	410	3.3260d+03	3.3592d+03	.0099	117.6
	1038	2046	3.3518d+03	3.3592d+03	.0022	554.3
M212	1	4	1.2577d+04	3.1372d+04	.60	
	308	612	2.6191d+04	2.6454d+04	.0099	152.8
	1008	2025	2.6373d+04	2.6454d+04	.0030	530.9
M213	1	4	5.5326d+05	1.9846d+06	.72	
	361	587	1.3111d+06	1.3243d+06	.010	179.3
	1061	2099	1.3199d+06	1.3243d+06	.0033	564.6
M312	1	8	1.3305d+04	3.1590d+04	.58	
	500	2480	2.4711d+04	2.5545d+04	.033	556.1
	1000	4778	2.4989d+04	2.5544d+04	.022	1209.7

Table 3-2: Basic Lower Bounding Functions

PROBLEM	ITERATION	CELLS	LOWER BOUND	UPPER BOUND	PRECISION	TIME
M211	1	4	2.5964d+03	3.4221d+03	.24	
	15	11	3.3286d+03	3.3595d+03	.0092	16.5
	25	19	3.3561d+03	3.3593d+03	.00094	28.2
	42	15	3.3589d+03	3.3592d+03	.000094	48.8
	57	13	3.3592d+03	3.3592d+03	.0000099	66.9
M212	1	4	2.3259d+04	2.6688d+04	.13	
	11	6	2.6358d+04	2.6459d+04	.0038	12.0
	17	7	2.6441d+04	2.6454d+04	.00048	19.3
	23	6	2.6451d+04	2.6454d+04	.000099	26.6
	28	6	2.6454d+04	2.6454d+04	.0000047	32.7
M213	1	4	1.1726d+06	1.3248d+06	.11	
	7	4	1.3127d+06	1.3243d+06	.0088	7.7
	12	4	1.3231d+06	1.3243d+06	.00090	13.8
	18	4	1.3242d+06	1.3243d+06	.000088	21.8
	21	2	1.3243d+06	1.3243d+06	.0000059	25.8
M214	1	4	6.8608d+05	7.7445d+05	.11	
	7	4	7.6947d+05	7.7425d+05	.0062	7.0
	12	1	7.7402d+05	7.7419d+05	.00023	13.0
	16	4	7.7412d+05	7.7419d+05	.000087	18.1
	18	3	7.7418d+05	7.7418d+05	.0000079	20.7
M221	1	4	4.5456d+03	6.0336d+03	.25	
	16	13	5.8646d+03	5.9182d+03	.0090	35.3
	30	13	5.9155d+03	5.9182d+03	.00044	68.0
	40	13	5.9176d+03	5.9182d+03	.000088	92.0
	52	13	5.9181d+03	5.9181d+03	.0000098	120.8
M222	1	4	3.7929d+04	4.4351d+04	.14	
	11	6	4.3760d+04	4.3959d+04	.0045	22.8
	17	6	4.3928d+04	4.3953d+04	.00057	37.3
	21	8	4.3949d+04	4.3953d+04	.000096	47.0
	30	5	4.3953d+04	4.3953d+04	.0000053	68.6
M223	1	4	1.7857d+06	2.0639d+06	.13	
	8	3	2.0545d+06	2.0617d+06	.0035	15.8
	12	3	2.0590d+06	2.061Od+06	.00096	26.4
	16	3	2.0607d+06	2.0609d+06	.000075	36.9
	17	1	2.0609d+06	2.0609d+06	.0000032	39.6
M224	1	4	9.5239d+05	1.0843d+06	.12	
	8	3	1.0788d+06	1.0841d+06	.0049	15.3
	12	4	1.0831d+06	1.0838d+06	.00059	24.8
	15	2	1.0837d+06	1.0837d+06	.000060	32.4
	19	4	1.0837d+06	1.0837d+06	.0000060	42.4

Table 3-3: Improved Lower Bounding Functions, 2-Dimensions

PROBLEM	ITERATION	CELLS	LOWER BOUND	UPPER BOUND	PRECISION	TIME
M311	1	8	2.9883d+03	3.6119d+03	.17	
	29	13	3.5584d+03	3.5739d+03	.0043	65.1
	40	32	3.5699d+03	3.5732d+03	.00093	91.5
	75	27	3.5729d+03	3.5732d+03	.000074	178.0
	102	26	3.5731d+03	3.5732d+03	.0000081	244.9
M312	1	8	2.2649d+04	2.5666d+04	.12	
	18	13	2.5306d+04	2.5546d+04	.0094	38.2
	32	12	2.5521d+04	2.5544d+04	.00093	71.0
	46	15	2.5541d+04	2.5543d+04	.000092	105.6
	62	15	2.5543d+04	2.5543d+04	.0000081	145.2
M321	1	8	5.3202d+03	6.3380d+03	.16	
	25	23	6.2461d+03	6.2890d+03	.0068	110.6
	46	30	6.2810d+03	6.2872d+03	.00099	214.1
	79	35	6.2866d+03	6.2872d+03	.000094	377.2
	108	31	6.2871d+03	6.2872d+03	.0000093	520.0
M322	1	8	3.7452d+04	4.2476d+04	.12	
	21	10	4.1908d+04	4.2207d+04	.0071	88.9
	33	13	4.2174d+04	4.2205d+04	.00074	147.9
	46	16	4.2202d+04	4.2205d+04	.000082	212.0
	66	11	4.2204d+04	4.2205d+04	.0000088	310.7
M411	1	16	2.5603d+04	2.9856d+04	.14	
	33	39	2.9558d+04	2.9831d+04	.0091	126.3
	73	21	2.9800d+04	2.9829d+04	.00097	321.2
	98	21	2.9826d+04	2.9829d+04	.000096	448.0
	139	41	2.9828d+04	2.9829d+04	.0000084	655.9
M511	1	32	3.7145d+04	4.2223d+04	.12	
	75	78	4.1776d+04	4.2194d+04	.0099	634.3
	155	78	4.2152d+04	4.2194d+04	.00099	1437.5
	253	102	4.2190d+04	4.2194d+04	.00010	2453.7
	360	113	4.2193d+04	4.2194d+04	.000010	3573.8
M611	1	64	3.6256d+04	4.0759d+04	.11	
	163	187	4.0321d+04	4.0727d+04	.01	2706.1
	326	327	4.0685d+04	4.0726d+04	.00099	6074.7
	607	290	4.0722d+04	4.0726d+04	.00010	12046.9
	832	224	4.0725d+04	4.0726d+04	.0000098	16867.6

Table 3-4: Improved Lower Bounding Functions, Higher Dimensions

4. Other Formulations

4.1. Min-Max Problem

The min-max location problem is given by:

P2) $\min\limits_{x \in R^n} \max\limits_{0 \le j \le L} f_j(d(x,z_j))$

It is easily seen that the following is a lower bounding function for P2:

$F(x,r) = maxffd(x,zp-r)$

A slightly tighter cell lower bounding function is given by:

$F(R) = max f_j(d(y_j,\dot{z}_j)-r)$

where y^ is the solution to

$$\min_{y_j \in R} dtyzp \tag{10}$$

As for the min-sum problem, choosing as the initial cell for the algorithm the smallest cell containing all of the Zj points will suffice.

This lower bounding function in Equation 10 was used on several problems given in Table 3-1. The results are given in Table 4-1. The number of active cells increases here roughly linearly with the number of iterations, which accounts for the relatively slow convergence.

4.2. Max-Min Problem

Consider max-min problem:

PS) $max \quad minfjidfazj))$

This is equivalent to the following max-min problem:

P3') $\min\limits_{x \in S} \max\limits_{0 \le j \le L} -ffd(x,zp)$

A Lower Bounding Function for P3 is given by:

$\boldsymbol{F(x,r) = max -fj(d(x,Zj)+r)}$

As for problems P1 and P2, a slightly tighter cell lower bounding function is given by:

$$F(R) = max -f_j(d(y_{\cdot},z_j)-r)$$

where y^ is the solution to

$$max \quad d(y_{j9}zp \qquad (11)$$

Unlike problems P1 and P2 where the the unconstrained problem would generally have a bounded solution (in the convex hull of the points Z;)9 here S must be a specified compact set, for otherwise an unbounded solution would be likely. (That is, since it is desired to be as far away from the points z; as possible, infinity in any direction would be optimal if it were feasible.) We consider here some examples where S is is given as a box, using simple upper and lower bounds on the components of x.

Let S be a box defined by

$$S\ m\ \{xe\ \mathrm{R}^n|\mathrm{X}_\mathrm{L}''\leq * \leq Xtf_9\ \ «=1.....«\}$$

The spatial branch-and-bound algorithm given in Section 2 requires that the region be divided into initial non-overlapping cells. Since the box S is the same form as a cell, one may use the initial cell division as the single cell given by the box S.

As for the problems in Table 3-1, these problems were constructed by generating the sources according to a uniform distribution on the [0,100]X[0,100] square and using power functions with weights and exponents drawn from a uniform distribution. S was taken to be the [30,70]X[30,70] box. The parameters used to generate these problems are given in Table 4-2.

The results using the lower bounding function given in Equation 11 are given in Table 4-3. There is one thing of note concerning the runs. The algorithm converges much faster if the solution is on the boundary. (TC=n in the table give n, the number of tight bounds for the optimal solution.)

PROBLEM	ITERA'noN	CELLS	LOWER BOUND	UPPER BOUND	PRECISION	TIME
M211	1	3	2.4110d+02	3.6738d+02	.34	
	22	15	3.0247d+02	3.0517d+02	.0089	11.3
	81	54	3.0472d+02	3.0502d+02	.00098	39.8
	282	216	3.0498d+02	3.0501d+02	.000099	137.6
M212	1	4	2.5662d+03	5.2648d+03	.51	
	38	31	3.4737d+03	3.5066d+03	.0094	19.1
	120	76	3.5006d+03	3.5041d+03	.00098	58.7
	378	229	3.5035d+03	3.5039d+03	.000099	184.3
M213	1	4	1.2588d+05	3.7663d+05	.67	
	52	39	2.0864d+05	2.1074d+05	.010	25.9
	165	104	2.1053d+05	2.1074d+05	.00098	80.5
	505	336	2.1071d+05	2.1073d+05	.000099	246.7
M214	1	4	1.0195d+05	3.4891d+05	.71	
	51	44	1.9002d+05	1.9192d+05	.0099	25.0
	155	110	1.9164d+05	1.9183d+05	.00097	74.5
	506	318	1.9181d+05	1.9183d+05	.00010	243.0
M311	1	8	2.3722d+02	3.8445d+02	.38	
	57	72	3.1091d+02	3.1405d+02	.010	57.1
	272	342	3.1355d+02	3.1386d+02	.0010	271.2
	1142	1256	3.1382d+02	3.1386d+02	.00010	1163.9

Table 4-1: Min-Max Problems

Problem Set	Dimension	# Sources	Weight Range	Exponent Range
T211	2	**50**	[5.10]	[-5.1.5]
T212	2	**50**	[5,10]	[-5.1.5]
T213	2	50	[5.10]	[-5.1.5]
T214	2	50	[5.10]	[.5.1.5]
T221	2	100	[5.10]	[.5.1.5]
T222	2	100	[5.10]	[.5.1.5]
T223	2	100	[5.10]	[.5.1.5]
T224	2	100	[5.10]	[.5.1.5]

Table 4-2: Max-Min Problem Set Characteristics

PROBLEM	ITERATION	CELLS	LOWER BOUND	UPPER BOUND	PRECISION	TIME
T211	1	4	-4.9248d+01	-3.4557d+01	.30	
TC-1	9	5	-4.6561d+01	-4.6305d+01	.0055	5.2
	13	5	-4.6531d+01	-4.6493d+01	.00083	7.3
	18	6	-4.6523d+01	-4.6519d+01	.000097	9.7
T212	1	4	-5.4479d+01	-3.8406d+01	.30	
TC=0	10	7	-4.6722d+01	-4.6317d+01	.0087	5.7
	22	9	-4.6365d+01	-4.6339d+01	.00057	11.5
	41	12	-4.6350d+01	-4.6347d+01	.000062	20.6
T213	1	2	-4.1181d+01	-3.3203d+01	.19	
TC-1	6	3	-3.9198d+01	-3.8976d+01	.0057	3.8
	9	2	-3.9198d+01	-3.9167d+01	.00077	5.3
	12	4	-3.9189d+01	-3.9186d+01	.000084	6.9
T214	1	4	-4.2255d+01	-2.4001d+01	.43	
TC=1	7	5	-3.6391d+01	-3.6077d+01	.0086	4.3
	11	4	-3.6261d+01	-3.6226d+01	.00094	6.3
	15	4	-3.6243d+01	-3.6240d+01	.000092	8.3
T221	1	4	-4.9248d+01	-3.4557d+01	.30	
TC=0	9	6	-4.3969d+01	-4.3615d+01	.0081	9.8
	20	7	-4.3762d+01	-4.3720d+01	.00095	20.2
	29	7	-4.3737d+01	-4.3733d+01	.000095	28.8
T222	1	4	-4.5387d+01	-3.2318d+01	.29	
TC=0	13	11	-3.9464d+01	-3.9078d+01	.0098	13.6
	28	5	-3.9257d+01	-3.9233d+01	.00062	27.9
	37	6	-3.9243d+01	-3.9240d+01	.000098	36.5
T223	1	4	-3.7219d+01	-2.4164d+01	.35	
TC=0	16	4	-3.1595d+01	-3.1386d+01	.0066	16.6
	24	5	-3.1501d+01	-3.1470d+01	.00099	24.3
	32	5	-3.1487d+01	-3.1484d+01	.000083	32.0
T224	1	4	-4.0303d+01	-2.2015d+01	.45	
TC=0	19	18	-3.3955d+01	-3.3641d+01	.0093	19.4
	37	10	-3.3772d+01	-3.3749d+01	.00066	36.6
	46	10	-3.3761d+01	-3.3758d+01	.000085	45.2

Table 4-3: Max-Min in Box, 2-Dimensions

4.3. Mixed Formulations

It is possible to use the spatial branch-and-bound algorithm given above for various mixed location formulations. For example, consider the mixed min-sum and min-max formulation:

$$\underset{x\in R^{*}}{min}\ \overset{n}{\underset{j=1}{\pounds}}\ fjidOvp) + \underset{J1+1\leq j\leq J2}{max} f_j(d(x,z_j))$$

A d.l.b.f. for this formulation can be found by simply adding the d.l.b.f.'s for the two parts:

$$F(x,r) = F1(x,r)+F2(x,r)$$

where

$$F1(x,r) = \sum_{j=1}^{J} f_j(d(x, z_j)-r)$$

$$F2(x_9r) = \underset{QZjZJ}{maxffdixjp\text{-}r)}$$

A tighter cell lower bounding function can be found by using the tighter versions for these two parts as given In Equations 2 and 10:

$$F(x,r)=Fl(R)+F2(R)$$

where

$$F1(R) = \sum_{j=1}^{J} f_j(d(y_j x, z_j)-r)$$

$$F2(x_9r) = max\, f_j(d(y_j x,z_j)-r)$$

where *yj* is the solution to

$$\underset{y_j eR}{min}\ d(y_j,z_j)$$

5. Conclusions and Future Work

The spatial branch-and-bound algorithm appears promising, particularly for the general Weber (min-sum) problem with Euclidean metric and power cost functions. The key to acceptable convergence rates seems to be in an improved lower bounding function, For the problem P1a, the Lipschitz based lower bounding function resulted in rather slow convergence whereas a quadratic lower bounding function converged far more rapidly, and even gave reasonable results for 3 and 4-dimensional problems.

While good results were seen using the basic lower bounding function for the max-min problem, they were not so good for the min-max problem. Perhaps trying to improve the lower bound estimate may yield benefits for the min-max problem.

The special lower bounding function for the the min-sum problem relied on using the Euclidean metric and power cost functions. For other metrics, the basic lower bounding function (Equation 2) is still usable, but attempts should be made to find an improved bounding function as done here for formulation P1a and P1b.

One obvious extension of this algorithm is to location problems with constraints. For example a feasible region consisting of a union of polygons should be able to be handled in the way as in the BSSS algorithm of [7]. If the cost functions were power functions, it is expected that using the improved lower bounding function given in Section 3.2 would result in vastly improved performance over the BSSS algorithm.

I. Convergence proof of the Algorithm

$$P)\ \min_{x \in S} F(x)$$

Lemma 7: Let F(x,r) be a d.l.b.f. for F, and ($y_{pf}x_p$, r_p) p = 1....P be a finite set of points and radii that satisfy:

$$i) \sum_{p=1}^{P} C(x_p,r_p) \supseteq S \quad (x_p \in S)$$

$$ii) y_p e \qquad R_p n S$$

$$iii) \min_{p} F(y_p) - e \leq \min_{p} F(x_{p9} r_p)$$

Then the x* that minimizes F(y_p) is an e-solution for (P)

Proof:

$$F(x_{pf} r_p) \leq F(x) \quad VxeCix\text{^}p)$$

$$\min_{p} j\{x_p,r_p) \leq F(x) \quad VxeS$$

$$\longrightarrow$$

$$Fix^*) - e = \min_{p} F(x_p) - e \leq \min_{p} F(x_p,r\text{^} \ \text{^}\ F(x) \quad \text{VjceS}$$

or

$F(x^*) - e \leq F(x)$ **V** *xe* 5<- -> *x*\s* **an e-solution (Q.E.D.)**

Lemma 8: Let F(x,r) be a d.l.b.f. for F, and (y_p,x_{pf} r^ p = 1,...,P be a finite set of points and radii that satisfy:

$$i) \sum_{p=1}^{P} C(x_p,\text{p a S} \ (x_p e S)$$

$$ii) y_p e \quad R_p n S$$

iii) $r_p \leq$ **5(e) (5 as given in Definition 3).**

Then

$$\min_{p} F(y_p) - e \leq \min_{p} F(x_p,r_p)$$

(And consequently the hypothesis of Lemma 1 is satisfied.)

Proof:

$$F(y_p) - e \leq Fix\text{^}rJ \quad p=l,...\text{^(by Definition 3)}$$

$$\rightarrow$$

$$\min_{p} F(y_p) - e \ £ \ \min_{p} F(x_p,r_p) (Q£.D.)$$

Lemma 9: Let F(x,r) be a d.l.b.f. for F defined over a compact set S. Let {x_p, r_p} p=1 ,...,P be a collection of points in S and radii that satisfy:

$$\sum_{p=1}^{P} C(x_p, r_p) = S.$$

$$x_p eS.$$

Let x* be the solution of min $F(x_p)$, $\overline{x}7$ be the solution of min $F(x_p, r_p)$. Then if *7 £* 5(e), x* is an e-solution to (P).

Proof:

$$FQ\overline{cs}) \geq F(x) - e \quad \text{(Definition 2)}$$
$$FQD - e \geq \min_{p} F(x_p) - e = F(x^* - e)$$

$$\longrightarrow$$

$$\min_{p} F(x_p, r_p) \geq \min_{p} F(x_p) - e.\text{(Q.E.D. by Lemma 2)}$$

Theorem 10: Let F(x,r) be a l.b.f. to F(x) defined over a compact set S. Further suppose that S may be divided into cells R_{pf} p=1 ,...,P where

$$i) \sum_{p=1}^{P} R_p = S$$

$$ii) V\{R_p n R_q] = 0 \quad \text{forp*?.}$$

Then the ALGORITHM (Section 2) converges to an e-solution for (P) in a finite number of iterations (e > 0).

Proof: Let S = 5(e) as in Definition 2. Let $\overline{n}$ = max n(Rj, 5) as in Definition 5. By Lemma 3, any region R with $r_R \leq 5$ cannot be divided before the hypothesis of Lemma 1 is satisfied (and hence an e-solution is found). Therefore the maximum number of iterations possible is

$$\sum_{i=1}^{m} \left(\sum_{j=0}^{n(R_p,\delta)-1} p^j \right) \leq m \sum_{j=0}^{n-1} p^j = m\,(p^{(\bar{n}-1)}-1)$$

(p is the number of sub-regions *g\f* divides a region into.) [an iteration is taken to be Step 2 of the ALGORITHM.]

REFERENCES

[I] Branin, F. H., Jr.
Widely Convergent Method for Finding Multiple Solutions of Simultaneous Nonlinear Equations.
IBM Journal of Research and Development **16():504-522,1972.**

[2] Chen, Reuven.
Solution of Location Problems with Radial Cost Functions.
Computers and Mathematics with Applications **10(1):87-94,1984.**

[3] Cooper, Leon.
An Extension to the Generalized Weber Problem.
Journal of Regional Science **8():181-197,1968.**

[4] Drezner, Z.; Wesolowsky, G.O.
Single Facility l_p-distance Minimax Location.
SIAM J. Algebraic Discrete Methods **1(3):315-321,1980.**

[5] Drezner, Z.; Wesolowsky, G.O.
A Maximin Location Problem with Maximum Distance Constraints.
AIIE Trans. 12(3):249-252,1980.

[6] Goldstein, A. A., and Price, J. F.
On Descent from Local Minima.
Mathematics of Computation **25():569-574,1971.**

[7] Hansen, P, Peeters, D., Richard, D, and Thisse, J.
The Minusum and Minimax Location Problems Revisited.
Operations Research 33(6)'A**251-1265,1985.**

[8] Katz, I.N.
Local Convergence in Fermat's Problem.
Mathimatical Programming **6():89-104,1974.**

[9] Meewella, C. C, and Mayne, D. Q.
An Algorithm for Global Optimization of Lipschitz Continuous Functions.
Journal of Optimization Theory and Applications **57(2):307-322,May,1988.**

[10] Mitten, I.G.
Branch-and-Bound Methods: General Formulations and Properties.
Operations Research **8():24-34,1970.**

[II] Shubert, Bruno, O.
A Sequential Method Seeking the Global Maximum of a Function.
SIAM Journal of Numerical Analysis **9(3):379-388,1972.**