NSF GRANT #0400140
NSF PROGRAM NAME: Operations Research

Minimax Sensor Location to Monitor a Piecewise Linear Curve

Tom M. Cavalier
The Pennsylvania State University
University Park, PA 16802

Whitney A. Conner
The Pennsylvania State University
University Park, PA 16802

Enrique del Castillo
The Pennsylvania State University
University Park, PA 16802

Abstract: This paper addresses the problem of locating a finite number of sensors to detect an event occurring along a piecewise linear curve. The objective is to minimize the maximum probability of non-detection. This sensor location problem has several applications, including the placement of sentries along a border to detect enemy penetration and the location of cell phone towers along a highway. The problem is a difficult nonlinear nonconvex programming problem even in the case of two sensors. A fast heuristic based on Voronoi diagrams is developed in this paper. The algorithm can quickly generate high-quality solutions. Computational experience is provided.

1. Introduction: Suppose that an event occurs along a piecewise linear curve in $\mathbb{R}^2$. The problem is to position a finite number of identical sensors so that the probability of detecting such an event is maximized, assuming that the effectiveness of each sensor decreases with its distance from the event. The solution to such a problem has numerous applications; for example, the placement of sensors along a border to detect enemy penetration, or the location of sensors to monitor events along a roadway. Another application would be to provide uninterrupted cell phone service along a major highway.

The above applications require locating a network of sensors where it is crucial that an event does not go undetected because of its location. The problem can be considered a facility location problem with an objective of providing equitable service to the customers, where the facilities being located are the sensors, and the customers are the event (source) locations. It can be classified as a continuous multifacility problem; however, in this problem, the objective terms are not simply weighted distances as in most location problems, but a propagation function of distance converted into probabilities of detection.

A planar version of this problem has been addressed by Drezner and Wesolowsky [1997] and Cavalier, *et al*. [2007], who demonstrated that a suitable choice of objective function is one that minimizes the maximum probability of nondetection. That is, the poorest response is made to be as good as possible. Minimax location models have also been used in other applications such as locating fire stations, hospitals and ambulance bases [Elzinga and Hearn 1972, Plastria 1995].

It is assumed that events, within the scope of this research, produce a 'signal' that interacts with the environment through which they propagate and the signal intensity decreases as a function of the distance between the sensor and the event. As mention in Drezner and Wesolowsky [1997] and Cavalier, *et al*. [2007], these detection probability functions (dpf) can be modeled in a variety of ways where $\pi(d)$ represents the probability of detecting an event at a distance *d*. Some of the most common dpfs are (1) *power decay*, $\pi(d) = \dfrac{\alpha}{\mu + d^n}$, $0 < \alpha \le \mu$, $n > 0$; (2) *exponential decay*, $\pi(d) = Ae^{-\beta d^n}$, $0 < A \le 1$, $\beta, n > 0$; and (3) *gravity decay*, $\pi(d) = 1 - e^{-k/d^n}$,

$k, n > 0$. By changing the dpf parameters, families of detection probability functions can be generated. In this paper, computational experience is generated using gravity decay, although the algorithms developed in this paper are not tailored for a specific dpf, but address the problem in a more general sense.

The following assumptions are made for the piecewise linear sensor location problem (PWL-SELP).

1. The *m* sensors to be located are identical.
2. The region to be monitored is piecewise linear curve *PWL* in the plane $(\mathbb{R}^2)$.
3. The event occurs with equal probability anywhere along *PWL*.
4. There are no existing sensors.
5. The detection probability function is only dependent on the distance from the event to the *i*th sensor.
6. All sensors are in perfect working order with no internal or external influences affecting performance. This idealized sensor has perfect discrimination and no false detections. That is, it will signal a detection only when the event is present.
7. The effectiveness of a sensor is independent of its location.
8. There are no constraints on the location of the sensors.
9. The probability of detection is a non-increasing, real-valued, continuous function of distance from the event.
10. The sensors operate independently of each other.

It is understood that these simplifications may not be valid in some actual cases. However, achieving solutions to simpler models can provide the foundation, insight and understanding to develop methods for more complex models. Because of the nonconvex nature of the objective function, verifying that the solution obtained is a global optimum can be quite difficult even for small problems.

Drezner and Wesolowsky [1997] addressed the problem of locating *p*-identical signal detectors on a unit line and a unit square. For the unit line problem, they used mathematical programming and a special algorithm designed to achieve the necessary conditions for optimality. For the square planar problem, four procedures were considered: a univariate search, a mathematical programming formulation, simulated annealing, and a Demjanov-type method. The planar problem was formulated using a finite number of possible event locations to produce a uniform grid covering the square region. This provides an approximate solution to the continuous problem.

The Demjanov-type algorithm, according to Drezner and Wesolowsky, provided the best solutions for the test problems in a 'reasonable' run time. The current station locations are improved by moving them in the direction of steepest descent. The direction is found using a method proposed by Demjanov [1968]. The objective function is then optimized along this direction using a one-dimensional optimization procedure.

Cavalier, *et al*. [2007] address the minimax sensor location problem when the underlying domain is an arbitrary convex polygon in the plane. They developed an algorithm called Toward the Largest Peak (TLP) that relies on Voronoi diagrams to approximate the probability of non-detection and generate search directions. A Voronoi diagram can be described as follows [Preparata and Shamos, 1985]: Let *R* be a set containing a finite number of points (e.g. sensors) in the plane. For each point $\mathbf{p}_i \in R$, the set of locations in the plane closer to $\mathbf{p}_i$ than any other point in *S* is an element of the Voronoi polygon of $\mathbf{p}_i$, denoted $V(i)$. The Voronoi diagram associated with *R* is $\mathcal{V} = \{V(1),...,V(n)\}$, where *n* is the number of points contained in *R*. Several algorithms exist for computing Voronoi diagrams. Okabe *et al*. [1992] give a detailed description of incremental, divide-and-conquer, and plane-sweep methods. Through computational testing, Cavalier, *et al*. [2007] demonstrated that the TLP algorithm was quite fast and typically generated solutions that were superior to MATLAB's Fminimax, an algorithm specifically designed to solve minimax-type problems.

From an application standpoint, research has been done in the field of wireless communication, specifically with respect to the base station location problem (BSTLP). With the increased demand for mobile communication services and deregulation acts (Tutschku 1998), the competition between service providers and potential revenues has driven the research for finding an optimal deployment of base stations which is both cost effective and provides the maximum possible coverage. The BSTLP involves locating multiple base stations within a region while providing an acceptable quality of service to mobiles (Howitt and Ham 1999). The BSTLP is different from

the sensor location problem in that the traffic and capacity on each base station is an important consideration.

Although there are similarities between the *p*-center problem (e.g., Love *et al*. [1973], Elzinga *et al*. [1976], Suzuki and Drezner [1996]) and the sensor location problem, there are significant differences. The *p*-center minimizes the maximum distance between facilities and demand points. It is an allocation problem, and thus each demand point is assigned to and interacts with its closest facility. On the other hand, the objective function in the sensor location problem involves the product of probabilities, making each event point interact with all sensors. It is easy to see that an optimal set of facility locations for the *p*-center problem will, in general, be a very poor solution to the sensor location problem.

Prior attempts at solving signal detection problems have focused on problems where random events occurred in a continuous planar region. These techniques are not directly applicable when the underlying domain is a piecewise linear curve. It is also easy to show that attempting to derive an approximate solution by utilizing the convex hull of the breakpoints of the piecewise linear function results in very poor solution quality. In the spirit of Cavalier, *et al*. [2007], this paper attempts to exploit the geometry of the problem rather than consider the specific form of the objective function. This is done by deriving local information from Voronoi polygons and adjusting sensor positions by a technique called toward the largest peak (TLP). The remainder of this paper is organized as follows: The succeeding section provides a mathematical formulation of the problem. This is followed by a detailed description of differential evolution, the TLP algorithm, computational experience, and concluding remarks.

2. PWL-SELP Mathematical Model: Let *PWL* represent a piecewise linear curve in $\mathbb{R}^2$ and let $T = \{\mathbf{t}_i : i = 1,..,b\}$ represent the set of breakpoints associated with *PWL*. Suppose that an event occurs at location $\mathbf{z} = (x, y) \in PWL$ and that the placement of *m* sensors (detectors) must be decided to monitor *PWL*. These sensors may or may not be located directly on *PWL*. For convenience, an event occurring at location **z** will be referred to simply as event **z**. The probability of detecting the event with the *i*th sensor (located at $\mathbf{x}_i = (x_i, y_i)$) is $\pi(d_i)$ where $d_i = d(\mathbf{z}, \mathbf{x}_i) = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ represents the Euclidean distance between sensor *i* and event **z**. The poorest response for the PWL-SELP will occur where the probability of non-detection is largest. Under the minimax criterion the largest probability of non-detection for *PWL* should be made as small as possible. Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ denote the vector of all sensor locations. Then if the *m* sensors operate independently of each other, the probability that an event **z** is not detected for a fixed set **X** of sensors is the product that each sensor individually does not detect event **z** and is given by $\prod_{i=1}^{m} \left[1 - \pi\left(d\left(\mathbf{z}, \mathbf{x}_i\right)\right)\right]$. As in Cavalier, *et al*. [2007], events are assumed to occur with equal probability anywhere along *PWL*. Thus, the problem of minimizing the maximum probability of non-detection can be stated as follows:

$$\text{(PWL-SELP)} \qquad \min_{\mathbf{x}_1,\ldots,\mathbf{x}_m} \psi\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\right) = \min_{\mathbf{x}_1,\ldots,\mathbf{x}_m} \max_{\mathbf{z}} \left\{ \prod_{i=1}^{m} \left[1 - \pi_i\left(d\left(\mathbf{z}, \mathbf{x}_i\right)\right)\right] \right\}$$

As mentioned earlier, (1) is a difficult nonlinear, nonconvex programming problem when $m > 1$. A solution algorithm for (1) can be visualized as an iterative two-phase process. For a fixed set of sensors locations, the value of ψ is found (or approximated). In general, the calculation of the value of ψ is nontrivial and is often approached through discretization of the domain. The sensors are then moved and ψ is recalculated. As this process continues the goal is to decrease the value of ψ until no improvement can be achieved by sensor relocation. This is the basic strategy adopted in this paper via the PWL-TLP algorithm that is detailed in Section 4. For comparison, an evolutionary approach often used in global optimization is also applied to PWL-SELP; this technique, called differential evolution, is outlined in the following section.

3. Differential Evolution (DE): Introduced by Storn and Price [1997], Differential Evolution (DE) is an evolutionary algorithm used to solve continuous space optimization problems. Like most evolutionary techniques, DE begins with a population of initial solutions which are then modified through the processes of mutation and crossover, thereby evolving a generation of improved solutions. An overview of

DE, using notation similar to that of Storn and Price, follows.

Begin by representing the problem variables as a *D*-dimensional vector. Initialize the algorithm by randomly generating *NP* initial solutions, called population members, represented as $\mathbf{x}_{i,G} = (x_{i1,G}, x_{i2,G}, ..., x_{iD,G})$ for $i = 1, 2, ..., NP$, where *G* is the generation. The mutation process randomly chooses a population member $\mathbf{x}_{r1,G}$ to form the base of the mutation. Then the weighted difference of two different randomly chosen population members, $\mathbf{x}_{r2,G}$ and $\mathbf{x}_{r3,G}$, are added to the base. *NP* mutated vectors are formed in this way, with $F \in [0, 2]$ being a scaling factor.

Mutation: $\mathbf{v}_{i,G+1} = \mathbf{x}_{r1,G} + F(\mathbf{x}_{r2,G} - \mathbf{x}_{r3,G})$

where $i \neq r1 \neq r2 \neq r3$

Next, the mutated vectors undergo crossover with the population in generation *G* according to the following rule:

Crossover:

$$u_{ij,G+1} = \begin{cases} v_{ij,G+1} & \text{if } j = rnbr(i) \text{ or if } rand(j) \leq CR \\ x_{ij,G} & \text{if } j \neq rnbr(i) \text{ and } rand(j) > CR \end{cases},$$
$$j = 1, 2, ..., D$$

where $CR \in [0,1]$ is the crossover constant, $rand(j)$ is a uniform random number in $[0,1]$, and $rnbr(i)$ is a randomly chosen index in $\{1, 2, ..., D\}$ that ensures that $\mathbf{u}_{i,G+1}$ gets at least one value from $\mathbf{v}_{i,G+1}$. Note that higher values of *CR* encourage more mutated variables to pass on to the trial vector $\mathbf{u}_{i,G+1}$. If $\mathbf{u}_{i,G+1}$ yields a better objective value than $\mathbf{x}_{i,G}$, then $\mathbf{x}_{i,G}$ is replaced with $\mathbf{u}_{i,G+1}$ in the next generation. Otherwise, $\mathbf{x}_{i,G}$ carries onto the next generation. This process continues until a maximum number of generations have been formed or the population converges to a single solution. In order to balance the need to adequately explore the solution space and quickly converge to a solution, tuning of the control parameters (*NP*, *F*, and *CR*) is necessary.

Several variations of DE have been suggested (e.g., Fan and Lampinen [2003], Kaelo and Ali [2006]) including Modified Differential Evolution (MDE) developed by Bergey and Ragsdale [2005]. MDE differs from DE in the mutation process. To choose $\mathbf{x}_{r1,G}$, the base of the mutation, the population members are sorted from highest fitness to lowest. Then, starting from the top of the list, a series of Bernoulli trials are performed until a success is found. The successful population member is chosen as the base of the mutation. The remainder of the DE algorithm remains unchanged. The Bernoulli trials are controlled by the probability of a successful Bernoulli trial (*PR)*. The idea behind this modification is that by using fitter population members as the base for mutation, the algorithm will converge faster. This, of course, comes at the expense of tuning *PR*, another problem dependent control parameter. Note that when *PR* =1, the best population member is always chosen as the base of the mutation which gives another common variation of DE.

Because the basic form of DE suffered from high computation times when applied to the sensor location problem, MDE was implemented for comparison with PWL-TLP. Empirical testing resulted in the following control variable settings: $F = 0.5$, $CR = 0.8$, and $PR = 0.5$. As suggested by Storn and Price [1997], *NP* was set to 10*D*, which is 20 times the number of sensors, i.e. 20*m*. For the purpose of crossover, each sensor location (i.e. coordinate pair) was treated as a single entity. Initial solutions were generated by choosing random coordinates from the smallest rectangle that would encompass the piecewise linear curve. During the mutation process, if a mutated sensor location is located outside this rectangular region, it is forced back to the nearest border of the rectangle. The implementation of MDE was developed by modifying the Matlab code, run1.m and devec3.m, provided by Storn and Price [2006]. The objective function was evaluated based on the breakpoints of the piecewise linear curve and a 50-point grid uniformly distributed along the piecewise linear curve.

4. Piecewise Linear - Toward the Largest Peak (PWL-TLP): As noted in Cavalier *et al*. [2007], given a set of sensor locations within a convex polygonal region $S \subset \mathbb{R}^2$, the peaks (or points of maximum probability of non-detection) can be effectively approximated by using the vertices of the Voronoi polygons generated by the sensor locations and the boundary of *S*. This was shown to be more efficient that discretizing the entire set *S*, and was the basic strategy employed by Cavalier *et al*. [2007] in solving the planar sensor location problem.

However, it is easily demonstrated that the vertices of the Voronoi polygons are insufficient to attain a suitable approximation of the probability of non-detection if the domain is a piecewise linear curve. Since the maximum probability of non-detection tends to occur along the boundaries of the Voronoi polygons, it was determined that a good approximation of the maximum probability of non-detection could be determined by utilizing two sets of points: the breakpoints $\mathbf{t}_j$ of *PWL* and the points $\mathbf{s}_i$ where *PWL* intersects the Voronoi diagram (See Figure 1). Once the probability of non-detection is determined for each intersection point $\mathbf{s}_i$ and each breakpoint $\mathbf{t}_j$, the sensor location within each Voronoi polygon is repositioned by moving it toward the largest local peak(s) defined by these points, thus reducing the peak (probability) and improving the overall objective value. Moving an individual sensor may not improve the objective at all; in fact, it may actually make the objective worse. Therefore, all sensors are moved simultaneously using local information derived from the individual Voronoi polygons and *PWL*. After repeating this process until no further improvement is possible, the sensors are repositioned using global information derived from the Voronoi diagram and *PWL* to fine tune the solution. This second phase can result in a significant improvement in the final solution; however, computational testing has shown that using only global information is generally inferior to using local information followed by global information. A more precise mathematical statement of the algorithm follows.

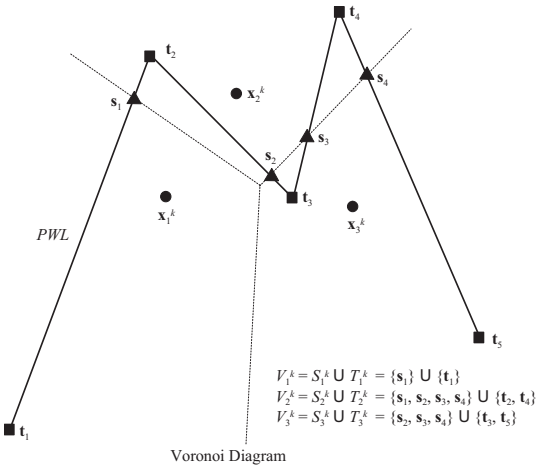

Figure 1. Sets V_i^k Defined by *PWL*, the Voronoi Diagram, and Three Sensors

PWL-TLP Algorithm

PHASE I – This phase uses local information derived from the Voronoi diagram to reposition the sensors.

Step 1. Set the iteration counter $k = 1$ and choose an initial set of sensor locations $\mathbf{x}_1^k, ..., \mathbf{x}_m^k \in S$

Step 2. Find the Voronoi diagram corresponding to the discrete set of points $\mathbf{x}_1^k, ..., \mathbf{x}_m^k$ and form a tessellation of the convex hull of *PWL*. Let S_i^k represent the set of points where *PWL* intersects the boundary of the Voronoi polygon corresponding to sensor location $\mathbf{x}_i^k$ (see Figure 1). Let T_i^k represent the subset of vertices of *PWL* contained in the Voronoi polygon corresponding to sensor location $\mathbf{x}_i^k$. Finally, Let $V_i^k = S_i^k \cup T_i^k$. V_i^k is the set of points in the Voronoi polygon containing $\mathbf{x}_i^k$ that will be used to approximate the maximum probability of non-detection.

Step 3. Determine the probability of non-detection $q_j = q(\mathbf{v}_j) = \prod_{i=1}^{m} \left[1 - \pi_i \left(d\left(\mathbf{v}_j, \mathbf{x}_i^k \right) \right) \right]$ for each point $\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^k$.

Step 4. For each sensor location $\mathbf{x}_i^k$, let $q_{i\max}^k = \max_{\mathbf{v}_j \in V_i^k} q_j$ and let $V_{i\max}^k = \left\{ \mathbf{v}_j \in V_i^k : q_j = q_{i\max}^k \right\}$. That is, $q_{i\max}^k$ is the maximum probability associated with the points contained in V_i^k and $V_{i\max}^k$ is the corresponding set of points. The current maximum probability of non-detection is $q_{\max}^k = \max_{i=1,..,m} q_{i\max}^k$.

Step 5. Compute $\mathbf{h}_i = \sum_{\mathbf{v}_j \in V_{i\max}^k} (\mathbf{v}_j - \mathbf{x}_i^k)$ and let $\mathbf{h}^t = (\mathbf{h}_1^t, ..., \mathbf{h}_m^t)$. Then $\mathbf{h} / \|\mathbf{h}\|$ is a unit direction vector that attempts to move each sensor location $\mathbf{x}_i^k$ toward the largest local peak(s) in its Voronoi

polygon.

Step 6. For each sensor location $\mathbf{x}_i^k$, compute $\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \Delta\left(\mathbf{h}_i / \|\mathbf{h}\|\right)$ for step length Δ. Find $V_i^{k+1} = S_i^{k+1} \cup T_i^{k+1}$, the set of points corresponding to each location $\mathbf{x}_i^{k+1}$. Determine the probability of non-detection q_j for each point $\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^{k+1}$ and the corresponding value of $q_{\max}^{k+1}$.

Step 7. If $q_{\max}^{k+1} < q_{\max}^{k}$, then replace k by $k+1$ and go to Step 4. Otherwise, replace Δ by 0.5Δ. If $\Delta < \varepsilon$, then stop; otherwise, go to Step 6.

PHASE II – This phase uses global information derived from the Voronoi diagram to reposition the sensor locations of Phase I.

Step 8. Given a set of sensor locations $\mathbf{x}_1^k, ..., \mathbf{x}_m^k$ determined in Phase I, set the iteration counter $k = 1$.

Step 9. Find the Voronoi diagram corresponding to the discrete set of points $\mathbf{x}_1^k, ..., \mathbf{x}_m^k$ and form a tessellation of the convex hull of *PWL*. Let S_i^k represent the set of points where *PWL* intersects the boundary of the Voronoi polygon corresponding to sensor location $\mathbf{x}_i^k$. Let T_i^k represent the subset of vertices of *PWL* contained in the Voronoi polygon corresponding to sensor location $\mathbf{x}_i^k$. Let $V_i^k = S_i^k \cup T_i^k$.

Step 10. Determine the probability of non-detection $q_j = q(\mathbf{v}_j) = \prod_{i=1}^{m}\left[1 - \pi_i\left(d\left(\mathbf{v}_j, \mathbf{x}_i^k\right)\right)\right]$ for each point $\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^k$.

Step 11. Let $q_{\max}^k = \max_{\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^k} q_j$ and let $V_{\max}^k = \left\{\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^k : q_j = q_{\max}^k\right\}$. That is, $q_{\max}^k$ is the maximum probability associated with the points derived in all Voronoi polygons and $V_{\max}^k$ is the corresponding set of points. Let $V_{i\max}^k$ be the subset of $V_{\max}^k$ consisting of the points that are closest to $\mathbf{x}_i^k$.

Step 12. Compute $\mathbf{h}_i = \sum_{\mathbf{v}_j \in V_{i\max}^k} (\mathbf{v}_j - \mathbf{x}_i^k)$ and let $\mathbf{h}^t = (\mathbf{h}_1^t, ..., \mathbf{h}_m^t)$. In this case, $\mathbf{h}/\|\mathbf{h}\|$ is a unit direction vector that attempts to move sensor location $\mathbf{x}_i^k$ toward the closest global peak(s).

Step 13. For each sensor location $\mathbf{x}_i^k$, compute $\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \Delta\left(\mathbf{h}_i / \|\mathbf{h}\|\right)$ for step length Δ. Find $V_i^{k+1} = S_i^{k+1} \cup T_i^{k+1}$, the set of points corresponding to each location $\mathbf{x}_i^{k+1}$. Determine the probability of non-detection q_j for each point $\mathbf{v}_j \in \bigcup_{i=1}^{m} V_i^{k+1}$ along with $q_{\max}^{k+1}$, $V_{\max}^{k+1}$, and $V_{i\max}^{k+1}$.

Step 14. If $q_{\max}^{k+1} < q_{\max}^{k}$, then replace k by $k+1$ and go to Step 11. Otherwise, replace Δ by 0.5Δ. If $\Delta < \varepsilon$, then stop; otherwise, go to Step 13.

Choosing a Set of Initial Solutions

Since PWL-SELP is a nonconvex programming problem, it has many local optima, and the choice of initial solution plays an important role in final solution quality. One option is to use many randomly generated initial solutions. However, through empirical testing, it was determined that high quality solutions could be obtained from initial solutions generated by uniformly distributing sensors along the original piecewise linear function (*PWL*) or a second piecewise linear function (*MPWL*) derived from the midpoints of the original segments. Thus, this was the tactic used to initiate PWL-TLP. Four types of starting solution sets were generated. Let L represent the length of *PWL*. In a Type 1 initial solution, the m sensors are uniformly distributed along *PWL* in a symmetric fashion with sensor 1 located a distance d from $\mathbf{t}_1$ and sensor m located a distance d from $\mathbf{t}_{n+1}$. Thus, the distance between adjacent sensors is $(L - 2d)/(m-1)$. Seven different such starting solutions are generated by varying the value of d uniformly on the interval $[0, 0.25L]$.

In a Type 2 starting solution, the sensors are separated by a uniform distance as they are located along *PWL*. For each solution, the separation distance between sensors is L/m. Again, seven such solutions are generated. The first solution set has sensor 1 located at $\mathbf{t}_1$ and sensor m located at a distance L/m from $\mathbf{t}_{n+1}$. The last solution set has sensor 1 located at a distance L/m from $\mathbf{t}_1$ and sensor m located at $\mathbf{t}_{n+1}$. The remaining starting solution sets are uniformly distributed between these two limiting cases.

Whereas Types 1 and 2 generate starting solutions lying directly on *PWL*, Types 3 and 4 contain sensors locations that are not directly on *PWL*. This is done by utilizing the same techniques used in defining Types 1 and 2, but applying the techniques to a second piecewise linear function, called the midpoint piecewise linear function (*MPWL*). This function is easily generated by utilizing $\mathbf{t}_1$ and $\mathbf{t}_{n+1}$ along with the midpoint of each original piecewise linear segment (e.g. see Figure 2).

Type 3 solutions are generated in exactly the same manner as Type 1 solutions except *MPWL* is used. Similarly, Type 4 solutions are generated by applying the techniques used in Type 2 solutions to *MPWL*.

A total of (7 + 7 + 7 + 7) = 28 starting solutions were generated. Figure 2 provides an example of each type of starting solution when there are five sensors $(m = 5)$ along with *PWL* and *MPWL*.

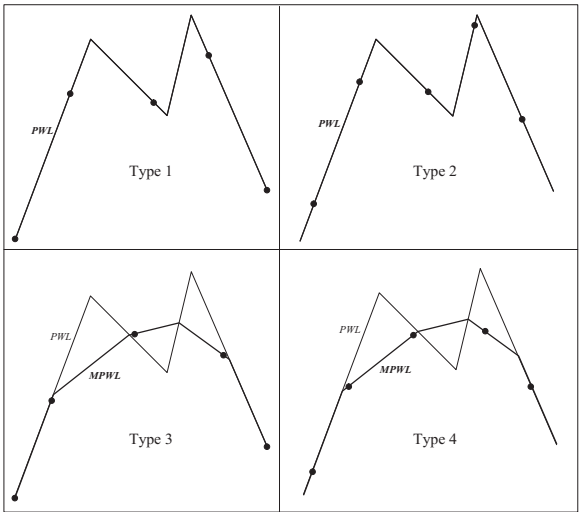

Figure 2. Examples of Starting Solutions

Choosing a Step Length Δ

The choice of the step length Δ in Step 6 of Phase I and Step 13 of Phase II also has a significant impact on the efficiency of the algorithm as well as the quality of the final solution. If the initial Δ is small, then many additional iterations may be necessary and the local optimum found will likely be near the initial solution. Whereas if the initial Δ is large, it may be possible to find a local optimum that is some distance away from the initial solution, but there may be an excessive number of unsuccessful steps where the objective does not improve. Consider Phase 1. Let $\mathbf{x}_1^k, ..., \mathbf{x}_m^k$ represent the current solution and for each $\mathbf{x}_i^k$, let $V_{i\max}^k = \left\{\mathbf{v}_j \in V_i^k : q_j = q_{i\max}^k\right\}$ as in Step 4. Let $d_{\min}^k = \min\limits_{i=1,...m}\left\{\min\limits_{\mathbf{v}_j \in V_{i\max}^k}\left\{d(\mathbf{v}_j, \mathbf{x}_i^k\right\}\right\}$. That is $d_{\min}^k$ is the minimum distance between the sensor locations and their respective largest local peaks. Then, to allow the algorithm the opportunity to find multiple local optima based on a single initial solution, several initial step sizes were used, $\alpha d_{\min}^k$ where $\alpha = 1, 3, 5, 7, 9$. Note, however, that in Step 6, the step length may be reduced by multiples of 0.5 until a successful step is taken. Let Δ^k be the actual successful step length used at iteration k. Then since the step lengths of successive iterations generally decrease during the course of the algorithm, the step length rule used for iteration $k+1$ is $\min\left\{6\Delta^k, \alpha d_{\min}^{k+1}\right\}$. These parameter settings were chosen using empirical testing. So, in essence, Phase I is solved five times with different step length parameters. The best solution is then passed to Phase II, where the step length rule is $\min\left\{6\Delta^k, \beta d_{\min}^{k+1}\right\}$ with $\beta = 1, 3, 5$.

5. Computational Results: The PWL-TLP algorithm was coded in FORTRAN and compiled with Compaq Visual FORTRAN Professional Edition 6.6.B. PWL-TLP was compared with the modified DE algorithm in Matlab that is outlined in section 3. For the modified differential evolution (MDE) algorithm, which was compiled using Matlab Compiler version 7.0, a grid of 50 uniformly-spaced points plus the breakpoints were used to discretize each piecewise linear curve and evaluate the maximum probability of non-detection. This grid size was chosen to attain a reasonable resolution while also taking into account cpu time. The initial population size was chosen as 20*m* where *m*

is the number of sensors; this choice was based on guidelines in Storn and Price [1997]. Twenty randomly generated populations were used to start MDE, and the best final solution was used for comparison with TLP-PWL.

Both programs were run under identical conditions on a PC with an Intel Pentium 4 2.4 Ghz CPU with 512 MB of RAM. For comparison, the probability of non-detection for the final solution of each algorithm was determined by utilizing a uniform 1000-point grid.

Both algorithms were used to find sets of sensors to monitor three different piecewise linear curves when the dpf was gravity decay defined by $\pi(d) = 1 - e^{-1/d^2}$. Piecewise Linear 1 (*PWL*1) has breakpoints $\{\mathbf{t}_1,...,\mathbf{t}_5\}$ = {(0,0), (3,8), (6,5), (7,9), (10,2)}. *PWL*2 has breakpoints $\{\mathbf{t}_1,...,\mathbf{t}_8\}$ = {(0,0), (1,5), (4,10), (7,7), (10,9), (11,6), (14,6), (17,2)}, and *PWL*3 has breakpoints $\{\mathbf{t}_1,...,\mathbf{t}_{11}\}$ = {(0,0), (1,4), (3,7), (5,6), (7,3), (9,4), (10,6), (12,8), (14,6), (16,7), (17,10)}. Tables 1, 2, and 3 summarize the results for the three different cases. Considering all 57 problems, the average cpu time for PWL-TLP was 4.62 seconds whereas MDE required an average of 714.27 seconds for 20 runs (an average of 35.71 seconds per run). Even more importantly (as the cpu times may be compiler-dependent), PWL-TLP attained a better final in 40 of the 57 problems with an average probability difference of 0.0084 (37.8%). The final solution of DE was better 17 times with an average probability difference of 0.0015 (0.30%). Note that PWL-TLP yielded the better solution in all cases with more than 10 sensors and as the number of sensors increases the quality of the PWL-TLP solutions become significantly better those provided by MDE. Finally, the number of times the final solution of PWL-TLP was generated by Type 1, 2, 3, and 4 starting solutions was 11, 12, 20, and 14, respectively.

6. Conclusion: This paper presented a heuristic algorithm for determining the location of a finite number of identical sensors to detect an event along a piecewise linear curve in the plane, when the objective is to minimize the maximum probability of non-detection. The problem is a difficult nonlinear nonconvex programming problem with a multitude of applications. The heuristic algorithm utilizes Voronoi polygons and intersection points along the piecewise linear curve to estimate the probability of non-detection and to determine a search direction. Computational results demonstrated that the algorithm is relatively fast and generates high-quality solutions.

7. Acknowledgment: This material is based upon work supported by the National Science Foundation under Grant No. 0400140. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Table 1: Results for gravity decay $\pi(d) = 1 - e^{-1/d^2}$ on $PWL1$

Number of Sensors	PWL-TLP Algorithm		Modified Differential Evolution				Difference $P_{TLP} - P_{MDE}$	(% Difference) $\frac{100*(P_{TLP} - P_{MDE})}{\min(P_{TLP}, P_{MDE})}$
	Probability of Non-detection P_{TLP}	CPU Time (secs)	Probability of Non-detection P_{MDE}	Total CPU Time (20 Runs) (secs)	Average Probability of Non-detection Per Run	Average CPU Time Per Run (secs)		
2	0.923979	0.17	0.923791	6.89	0.923800	0.34	0.000188	0.02
3	0.825445	0.31	0.824666	13.78	0.824748	0.68	0.000779	0.09
4	0.686947	0.58	0.688951	26.97	0.688992	1.34	-0.002004	-0.29
5	0.609177	0.89	0.607635	49.90	0.608746	2.49	0.001542	0.25
6	0.452588	1.63	0.450589	74.57	0.453885	3.72	0.001999	0.44
7	0.365932	1.64	0.364393	121.32	0.366678	6.06	0.001539	0.42
8	0.270097	2.65	0.274351	168.94	0.277947	8.44	-0.004254	-1.57
9	0.182762	3.45	0.184481	231.64	0.185757	11.58	-0.001719	-0.94
10	0.130374	4.19	0.137821	343.05	0.140096	17.15	-0.007447	-5.71
11	0.103159	4.43	0.106926	442.21	0.111480	22.10	-0.003767	-3.65
12	0.053221	4.73	0.063218	488.37	0.063956	24.41	-0.009997	-18.78
13	0.037402	5.91	0.041882	718.75	0.046033	35.93	-0.004480	-11.98
14	0.019408	6.36	0.025312	778.23	0.027888	38.90	-0.005904	-30.42
15	0.014571	7.15	0.021933	850.52	0.023684	42.52	-0.007362	-50.53
16	0.009566	7.32	0.013199	805.29	0.015660	40.26	-0.003633	-37.98
17	0.004915	9.61	0.007855	1091.72	0.009707	54.58	-0.002940	-59.82
18	0.002360	10.17	0.005821	866.13	0.006943	43.30	-0.003461	-146.65
19	0.001794	10.40	0.002914	919.49	0.005414	45.97	-0.001120	-62.43
20	0.000442	12.48	0.002035	884.45	0.003553	44.22	-0.001593	-360.41

Table 2: Results for gravity decay $\pi(d) = 1-e^{-1/d^2}$ on $PWL2$

Number of Sensors	PWL-TLP Algorithm		Modified Differential Evolution				Difference $P_{TLP} - P_{MDE}$	(% Difference) $\frac{100*(P_{TLP} - P_{MDE})}{\min(P_{TLP}, P_{MDE})}$
	Probability of Non-detection P_{TLP}	CPU Time (secs)	Probability of Non-detection P_{MDE}	Total CPU Time (20 Runs) (secs)	Average Probability of Non-detection Per Run	Average CPU Time Per Run (secs)		
2	0.958736	0.28	0.958549	6.38	0.958556	0.31	0.000187	0.02
3	0.896845	0.46	0.896029	13.01	0.896065	0.64	0.000816	0.09
4	0.812251	0.71	0.810871	24.20	0.811012	1.20	0.001380	0.17
5	0.754757	1.00	0.752736	43.59	0.752779	2.17	0.002021	0.27
6	0.625782	1.60	0.626193	75.81	0.627113	3.78	-0.000411	-0.07
7	0.552479	1.81	0.558455	123.13	0.558602	6.15	-0.005976	-1.08
8	0.494787	2.45	0.492183	235.58	0.495358	11.77	0.002604	0.53
9	0.345942	2.82	0.345611	286.93	0.345862	14.34	0.000331	0.10
10	0.325439	3.02	0.321957	454.82	0.327716	22.73	0.003482	1.08
11	0.228723	3.86	0.255171	489.04	0.257567	24.45	-0.026448	-11.56
12	0.167942	4.44	0.183760	601.02	0.187277	30.04	-0.015818	-9.42
13	0.141588	4.78	0.160231	930.38	0.169489	46.51	-0.018643	-13.17
14	0.098016	5.46	0.110165	1188.63	0.113764	59.42	-0.012149	-12.39
15	0.071787	5.99	0.085766	1680.22	0.091443	84.00	-0.013979	-19.47
16	0.062824	6.47	0.079274	1809.28	0.082416	90.46	-0.016450	-26.18
17	0.031273	7.94	0.044305	1847.05	0.052136	92.34	-0.013032	-41.67
18	0.026950	8.34	0.038961	1671.71	0.044773	83.58	-0.012011	-44.57
19	0.017278	9.33	0.028478	2138.63	0.034182	106.92	-0.011200	-64.82
20	0.008651	10.22	0.019464	2219.61	0.027278	110.97	-0.010813	-124.99

Table 3: Results for gravity decay $\pi(d) = 1 - e^{-1/d^2}$ on $PWL3$

Number of Sensors	PWL-TLP Algorithm		Modified Differential Evolution				Difference $P_{TLP} - P_{MDE}$	(% Difference) $\frac{100*(P_{TLP} - P_{MDE})}{\min(P_{TLP}, P_{MDE})}$
	Probability of Non-detection P_{TLP}	CPU Time (secs)	Probability of Non-detection P_{MDE}	Total CPU Time (20 Runs) (secs)	Average Probability of Non-detection Per Run	Average CPU Time Per Run (secs)		
2	0.942040	0.33	0.941996	6.76	0.942005	0.33	0.000044	0.005
3	0.880704	0.58	0.881048	13.80	0.881076	0.68	-0.000344	-0.04
4	0.811110	0.86	0.809195	29.33	0.809580	1.46	0.001915	0.24
5	0.686219	1.30	0.690278	44.76	0.690637	2.23	-0.004059	-0.59
6	0.631843	1.79	0.630601	87.92	0.630845	4.39	0.001242	0.20
7	0.536986	2.27	0.531142	129.05	0.533892	6.45	0.005844	1.10
8	0.420759	2.83	0.420389	190.64	0.420873	9.53	0.000370	0.09
9	0.346431	3.11	0.352247	286.95	0.358809	14.34	-0.005816	-1.68
10	0.299309	3.47	0.300323	502.11	0.304929	25.10	-0.001014	-0.34
11	0.233970	4.48	0.242710	573.40	0.247077	28.66	-0.008740	-3.74
12	0.148366	4.73	0.156838	809.13	0.160759	40.45	-0.008472	-5.71
13	0.116869	5.17	0.135757	868.71	0.140185	43.43	-0.018888	-16.16
14	0.099422	6.20	0.109824	1767.85	0.113187	88.39	-0.010402	-10.46
15	0.067541	6.40	0.077230	1558.93	0.086828	77.94	-0.009689	-14.35
16	0.039541	8.50	0.052981	1616.18	0.059911	80.80	-0.013440	-33.99
17	0.031710	8.20	0.044998	1414.37	0.049662	70.71	-0.013288	-41.90
18	0.020541	8.74	0.028283	2058.75	0.038232	102.93	-0.007742	-37.69
19	0.013209	9.44	0.021848	2144.96	0.026994	107.24	-0.008639	-65.40
20	0.007867	10.15	0.017381	1888.44	0.021983	94.41	-0.009514	-120.94

8. References:

Bergey, P., C. Ragsdale [2005] Modified Differential Evolution: a Greedy Random Strategy for Genetic Recombination, *Omega: The International Journal of Management Science* 33:255-265.

Cavalier, T.M., W.A. Conner, E. del Castillo, S.I. Brown [2007] A Heuristic Algorithm for Minimax Sensor Location in the Plane, *European Journal of Operational Research* 182:42-55.

Demjanov, V.F. [1968] Algorithms for Some Minimax Problems, *Journal of Computers and System Science* 2:342-380.

Drezner, Z., G.O. Wesolowsky [1997] On the Best Location of Signal Detectors, *IIE Transactions* 29(11):1007-1015.

Elzinga, D.J., D.W. Hearn [1972] Geometrical Solutions for Some Minimax Location Problems, *Transportation Science* 6:379-394.

Elzinga, D.J., D.W. Hearn, W.D. Randolph [1976] Minimax Multifacility Location with Euclidean Distances, *Transportation Science* 10:321-336.

Fan, H., J. Lampinen [2003] A Trigonometric Mutation Operation to Differential Evolution, *Journal of Global Optimization* 27:105-129.

Howitt, I., S.-Y Ham [1999] Base Station Location Optimization, *IEEE Vehicular Technology Conference* 4:2067-2071.

Kaelo, P., M. Ali [2006] A Numerical Study of Some Modified Differential Evolution Algorithms, *European Journal of Operations Research* 169:1176-1184.

Love, R.F., G.O. Wesolowsky, S.A. Kraemer [1973] A Multifacility Minimax Location Method for Euclidean Distances, *International Journal of Production Research* 11:32-40.

Okabe, A.; Boots, B.; and Sugihara, K. [1992] *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, New York: John Wiley and Sons Ltd.

Plastria, F. [1995] Continuous Location Problems, in *Facility Location: A Survey of Applications and Methods*, Edited by Z. Drezner, Springer-Verlag, New York, pp. 225-262.

Preparata, F.P. and Shamos, M. I. [1985] *Computational Geometry: An Introduction*, New York: Springer-Verlag.

Storn, R., K. Price [1997] Differential Evolution- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization* 11: 341-359.

Storn, R., K. Price [2006] Differential Evolution (DE) for Continuous Function Optimization. http://www.icsi.berkeley.edu/~storn/code.html#matl (accessed 30 November 2006).

Suzuki, A., Z. Drezner [1996] The *p*-Center Location Problem in an Area, *Location Science* 4(1/2):69-82.

Tutschku, K. [1998] Demand-Based Radio Network Planning of Cellular Mobile Communication Systems, *Proceedings - IEEE INFOCOM* 3:1054-1061.