Maximal Breach and Support: Geometric Characterisation and Algorithms

Anirvan DuttaGupta, Arijit Ghosh, Arijit Bishnu, Indranil Sengupta

Abstract. *Coverage* is an important issue in Wireless Sensors Networks (*WSN*s). It gives a measure of the quality of surveillance a WSN provides over a field it is designed to monitor. Different measures of coverage capture different aspects of surveillance. In this paper, we study two measures of coverage - *maximal breach/support*, and *all-pairs average* maximal breach/support. The geometric characterisation of these measures is one of the principal goals of our study. We have treated the coverage problem from two angles - from the *intruder*'s point of view, and from the *defender*'s point of view. These two view-points give rise to two generic combinatorial optimisation problems.

I Searching for "safe" paths in the field (important for the intruder).

II Optimising the degree of coverage over all parts of the field (important for the defender).

Our study of the geometric and combinatorial properties of single-pair and average maximal breach have led to exact polynomial time algorithms that solve problem **I** above when applied to these two specific measures. We have also framed and solved problem **II** for *single-pair* maximal breach. We have proved a "relaxed" version of problem **II**, applied to maximal breach, to be **NP-Hard** and designed an approximation algorithm for it. We have devised an efficient heuristic for solving some insertion based problems on maximal breach. We have stated and proved four basic theorems describing the behaviour of single-pair maximal breach under sensor insertion and deletion. These theorems have been used to develop the heuristic mentioned above. They have also led to important *lower-bound* results for single-pair and all-pairs average maximal breach.

1 Introduction

In this section, we begin with a brief introduction to Wireless Sensor Networks, with special reference to sensor network *coverage*, which is the focus of our study. In the later part of this section, we provide an overview of the research problems that form the substance of this paper and summarise our approaches and results. We end with a brief description of the organisation of the thesis. We refer the interested reader to [21] for a detailed study of the material in §1.1.

1.1 Wireless Sensor Networks: A Brief Introduction

Recent advances in wireless technologies have led to low cost and low power devices, efficient protocols for reliable communication among them and embedded computing ability within devices. This has been coupled with theoretical work leading to centralised and distributed algorithms for various information processing tasks. All these have made Wireless Sensor Networks (*WSN*s) a common and effective solution in a wide range of applications.

Indeed, proponents of ubiquitous computing and other industry visionaries anticipate a "not-too-distant future" when *WSN*s will integrate with our daily lives to such an extent that we shall cease to notice their presence [22]. The sensors of the future will be cheap enough to be used in huge numbers, almost wastefully. Networks of such sensors will perform all imaginable kinds of monitoring, surveillance and data accumulation tasks like monitoring urban vehicular traffic, protecting wildlife habitats from human-induced or natural catastrophies like forest fires and tracking job flows in automated factories [21]. The integration of networks of sensors with PCs, network-enabled hand-held devices and the *Internet* will bring about the next big revolution in computing [27].

So much for the future. Research effort in the past few years, in the area of Wireless Sensor Networks, has been intense and diverse. The area has become the meeting ground of researchers in signal processing and embedded computation [26, 27], network architecture and protocols [28], distributed algorithms [4] and computational geometry [1, 5, 25], to name just a few.

1.2 Key Problems in Sensor Network Design

The diverse research areas that have merged into the study of *WSN*s indicate that the key problems that *WSN* designers face are equally varied. Consider *internetworking* for example. Since networked sensors extract data from physical environments as varied as chemical plant reactors and urban traffic intersections, the volume and nature of data they communicate to the control/interpretation centres are fundamentally different from those flowing in the conventional *TCP/IP* internets. Researchers have begun to feel the need of re-engineering long-established protocols like the *TCP/IP* to suit the needs of the modern *WSN*s [21].

In surveillance applications, the *location* of a detected event is always as important a piece of information as the event itself. For instance, the intimation that a forest-fire has started is of no use unless it is accompanied by an accurate estimate of *where* the fire has been detected. Sensors are more often than not *randomly* deployed, meaning that their exact coordinates are not predetermined. Networked sensors thus have the task of locating themselves as well as the events they report. This has a bearing on *connectivity* too. Sensor networks *communicate* information to central controlling facilities - they do this through a series of messages routed through intervening sensors that are *connected*, i.e., their sensing regions overlap. Individual sensor nodes thus have to form a *local* view of the network to fix their positions in relation to their neighbours. See [29, 30] for details. This is one area where distributed algorithms play an important role - because they are less demanding in terms of computing resources and offer better scalability.

Power conservation is another important consideration. The battery-life of individual sensors is usually small, and a network is expected to outlast the sensors that it is made up of. Judicious activation/deactivation of individual sensors, without compromising on the quality of service offered, is an interesting problem in this area. Saving power *during* communication is another important issue. Energy spent per bit of data sent between two sensors at a distance d is given by $E = Bd^y$, where B stands for the overhead per bit and y ($y > 1$) is a path loss exponent [1]. However, indiscriminately increasing the number of hops for transmitting a message is not a feasible solution, because energy is also consumed due to the computation required for storing and fowarding messages by the individual nodes. Designing effective topologies and routing schemes that minimise the average energy spent in communication is a significant problem. See [7, 23, 24] for a more detailed account of this

class of problems.

Coverage is one of the basic problems in sensor networks - it is a measure of the quality of surveillance offered by a given network of sensors over the field it is designed to protect. Any measure of coverage quantifies how well the network monitors each and every point on the field. Geometric characterisation of, and optimisation problems pertaining to, two specific measures of coverage form the subject matter of our work. We shall formally define these measures in the later Sections.

1.3 Measures of Coverage

Quite a few measures of coverage are found in the literature. Different measures capture different aspects of surveillance quality. We shall mention four common and important measures here. The first one, *k-Coverage*, is intuitively the simplest - it indicates how secure *each* and *every* point in the field is. The second measure, *k-Barrier-Coverage*, is geared towards protecting international borders. The third measure, *Exposure* takes into account the fact that it is more difficult to detect fast-moving objects than slow-moving ones. Finally, *maximal breach* and *support*, which are the measures we study in this paper, deal with upper and lower bounds on the distances of the sensors from the paths an intruder can follow.

1.3.1 *k*-Coverage

This measure is defined in [5]. Given a field A and a set of sensors S, a point $p \in A$ is said to be *covered* by a sensor $s \in S$ if p belongs to the sensing region of s. A point $p \in A$ is said to be *k-Covered* if and only if it is covered by at least k sensors in S. See Fig. 1. The circles with dots at the centres denote sensing regions of the sensors. The numbers in the figure show the coverage of the points in the corresponding sub-region. With regard to *k*-Coverage, Huang and Tseng [5] have

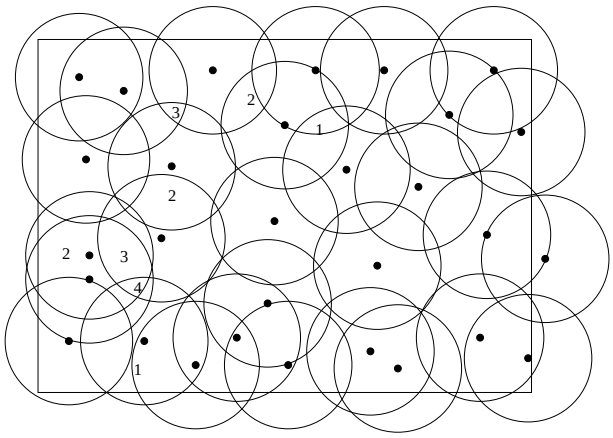

Figure 1: *k*-**Coverage.**

solved the following decision problems: Given A, S and a positive integer k,

1. If the sensing region of each $s \in S$ is a *unit*-circle, is A k-Covered?
2. If the sensing regions are circles of varying radii, is A k-Covered?
3. If the sensing regions are irregular, but bounded by *precisely* defined closed curves, is A k-Covered?

But they have left the following *optimisation* problem open:

- A sub-region $B \subset A$ is said to be *insufficiently* covered if B is j-Covered, but the desired level is k-Coverage for some $k > j$. Determine the *optimal* set of sensors that can patch all such regions B so that A becomes fully k-Covered.

1.3.2 k-Barrier-Coverage

This measure is defined in [8]. A *belt*-region is defined as a subset of points on the plane bounded by two "parallel" curves. An annulus is an example; so is the strip between a pair of railway tracks. Unlike k-Coverage, k-Barrier coverage does not attempt to cover all points within the field; it is restricted to covering crossing paths through belt-regions. A crossing path through a belt-region is said to be k-Barrier-Covered by a set of sensors S if it intersects at least k sensors in S. A belt-region is k-Barrier-Covered if and only if *all* crossing paths are k-Barrier-Covered. The belt region in Fig. 2 is 3-Barrier-Covered.

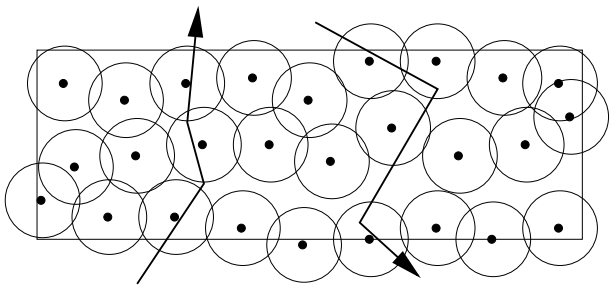

Figure 2: 3-**Barrier-Coverage.**

It is quite evident that this measure is especially suited to securing international borders. Kumar et al. [8] have derived the necessary and sufficient conditions for achieving k-Barrier-Coverage for a variety of belt regions.

1.3.3 Exposure

This measure, defined by Veltri et al. [3], is particularly interesting because it captures the difficulty of detecting or tracking a moving object in the field. Intuitively, the longer the time an object spends within the sensing region of a sensor, the more it is "exposed" to the likelihood of being detected by it. So, the velocity of an object should be taken into account when measuring the insurance of coverage provided by S (the set of sensors) against moving objects in the field A. Let

$I_A(s, p)$ denote the "intensity" of the sensor $s \in S$ at the point $p \in A$. The function $I_A(s, p)$ is negatively correlated with the euclidean distance $d(s, p)$. The intensity of the set S at p is defined as $I_A(S, p) = \sum_{s \in S} I_A(s, p)$.

Now, let $P \equiv p(t)$ be a curve, parameterised by time $t \in [0, T]$, representing the trajectory of an object moving through A. Let $v(t) = \frac{dp}{dt}$ denote its instantaneous velocity at time t. Then, the exposure of the object to S is given by

$$E_S(P) = \int_0^T I_A(S, p(t))v(t)dt.$$

Paths in A for which this measure is minimised or maximised are called *minimal* and *maximal exposure paths* respectively. When S is singleton, Veltri et al. [3] have derived an analytic solution to the problem of finding the minimal exposure path. When $|S| > 1$, they claim that a closed-form solution does not exist. However, they propose grid-based approximate solution procedures in such cases. They also claim to have proved the problem of finding the *maximal* exposure path **NP-Hard**. The result might be correct, but the proof is not, because they have made a reduction in the wrong direction.

1.3.4 Breach and Support

Breach and support are measures of central importance in our work. We shall define these measures formally in the succeeding sections. Intuitively, the *breach* of a path P represents a lower bound on the distance of P from any sensor in S, while the *support* of a path represents an upper bound. For an *intruder*, a path with a high breach value is safe to take, and a path with a small support value is extremely unsafe.

1.4 Overview and Contribution of the Paper

The multitude of coverage measures discussed in §2 capture different aspects of coverage based on the requirement in hand. There is one thing common among all these measures - the designer of a *WSN*, whom we call the *defender*, will try deploy a set of sensors S such that the best possible (optimal) value of the chosen measure is achieved. On the other hand, the intruder will always try to find segments or paths in the field that are the most insecure, i.e., the value of the coverage measure chosen is the *worst* possible. These conflicting goals indicate that the coverage problem can be viewed from two angles: from the point of view of the intruder, who chooses poorly covered paths; and from that of the defender, who tries to ensure that the coverage level is fairly high for all paths. In other words, the intruder's focus is on a *search* problem: finding the safest path. The defender concentrates on an *optimisation* problem: finding a set of sensors S that provides maximum coverage over the field A.

In our work, we have looked at the coverage problem from these two angles. We have concentrated upon two measures of coverage: maximal breach and support and average maximal breach and support . For these two measures,

- We have developed improved algorithms for computing the minimal and maximal coverage paths.

- We have framed a class of optimisation problems. We have solved some of them entirely. For others we have proved NP-Hardness results and devised some heuristics.
- From a theoretical stand-point, we have looked for geometric and combinatorial characterisations of these measures. In this we have achieved a degree of success - most of our exact algorithms have been directly derived from these characterisations.

The principal contributions of this paper are summarised below:

1. We have proposed a completely new algorithm for computing maximal breach and support paths. This algorithm offers a number of advantages over the previous centralised algorithm due to Megerian et al. [1].
2. We have stated and proved four fundamental theorems describing the behaviour of the maximal breach measure under addition and removal of sensors from the field. As corollaries to these theorems, we have established the correctness of two breach minimisation heuristics.
3. We have framed optimisation and decision problems based on the maximal breach measure and solved them.
4. We have pointed out a shortcoming of the maximal breach and support measures in their original (*single-pair*) form, and proposed an extension to these measures - *average* maximal breach and support. We show that they are sound and useful.
5. We have developed an *optimal* algorithm for computing average maximal breach and support.
6. We have reframed the optimisation/decision problems in terms of the average measures and solved one of the variants (minimising the value of average breach). We have proved a "relaxed" version of the other variant (minimising the *number* of sensors) to be hard. We have also proposed a heuristic for minimising average maximal breach. It starts from a random set S and iteratively inserts a k sensors at a time; at each step it ensures maximum improvement in the value of average breach. We have verified it formally and through simulations.

2 Background and Related Work

As stated in §1.4, the object of our study is the geometric characterisation of some coverage measures in *WSN*s and some optimisation problems that arise naturally from them. Among the coverage measures we have studied, Maximal Breach and Maximal Support are the most important. These were first proposed in [6]. In this section, we formally introduce these measures and summarise the work done in this area.

2.1 Single-Pair Maximal Breach and Support

The measures we define below are known by the name *Maximal Breach* and *Maximal Support* in the literature. However, to distinguish them from two other closely related measures defined in later sections (Sections 7), we shall refer to them as *single-pair* maximal breach and support.

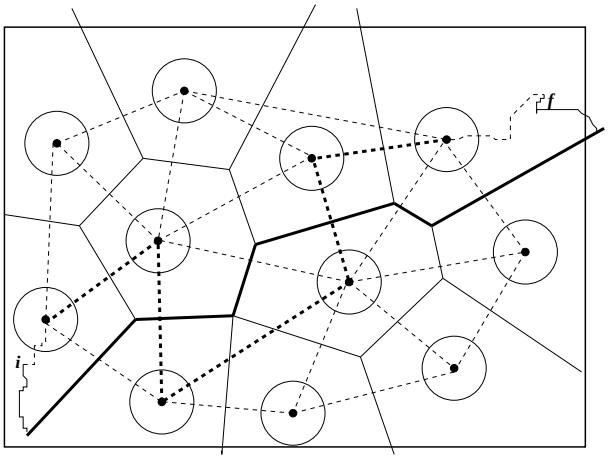

Figure 3: **Maximal Breach and Support Paths:** The sensors are depicted as circles with prominent centres. The continuous line segments are the edges of the Voronoi Diagram, and the dashed ones are the edges of the Delaunay triangulation. The segments drawn in bold type signify maximal breach and support paths.

2.1.1 Setup

Suppose a number of sensors are deployed over a terrain (field). Without loss of generality we assume a field A will stand for an open region on the plane bounded by a unit-square. But most of our results are equally valid if we take A to be a planar region bounded by any *convex polygon*. Let $S = \{s_1, s_2, \ldots, s_N\}$ be a set of N sensors deployed over the field. We shall look upon the sensor nodes s_i as points $(x_i, y_i) \in A$. We shall also assume that the *intruder* has complete knowledge of the coordinates of *all* the sensors in S. Suppose, in this worst-case scenario, the *defender* wants to guage the extent of security S provides against an intruder armed with complete knowledge. Before going into details, we start with some definitions.

2.1.2 Definitions

We shall follow [1] in defining the maximal breach and support measures. A is a unit-square field, and $S = \{s_i \equiv (x_i, y_i) \mid 0 < x_i, y_i < 1, \forall 1 \leq i \leq N\}$. Suppose an intruder tries to traverse the field from an initial point i to a final point f (Figure 3). We denote *points* within A, with lower-case letters and *paths* with upper-case letters.

Consider a path $P(i, f)$ through the field from i to f.

Definition 1 ***[Breach]*** *The quantity* breach *is defined as the* minimum *Euclidean distance from $P(i, f)$ to any sensor in the field.*

Intuitively, the breach of a path is a lower bound on the distance of any point on the path to any sensor in S. Thus, for every point p on $P(i, f)$, we measure the distance of p from the *nearest* sensor and finally take the *minimum* of all these distances. But in A, there are infinitely many paths connecting i and f. One of these has a special property:

Definition 2 ***[Maximal Breach Path]*** *Among the infinitely many paths connecting i and f, one that has the* maximum *breach value is called a* maximal breach path, $P_b(i, f, S)$. Maximal Breach, $breach_{max}(i, f, S)$, *is the breach of the maximal breach path.*

Note that both the maximal breach *path*, P_b, and the maximal breach *value*, $breach_{max}$ are functions of the points i, f and the set of sensor nodes S. Hence, the symbols are qualified with (i, f, S). For a more detailed account of the notations we follow throughout the paper, we refer the reader to §3.2.

In geometric terms, *support* is the dual of *breach*.

Definition 3 ***[Support]*** *The measure* support *for a path $P(i, f)$ connecting i to f is defined as the* maximum *euclidean distance from $P(i, f)$ to the nearest sensor.*

Intuitively, the support of a path is an *upper*-bound on the distance of any point on the path to any sensor in S. For every point p on $P(i, f)$, we measure the distance of p from the nearest sensor, and finally take the *maximum* of all these distances. As in the the case of breach, there is one path which is special in terms of *support*.

Definition 4 ***[Maximal Support Path]*** $P_s(i, f, S)$, *the* maximal support path *is defined as the one with the* lowest *value of support among all possible paths connecting i to f. Maximal Support, $support_{min}(i, f, S)$, is the support of the maximal support path.*

The prefix "*_min*" and the name *maximal* support may sound anomalous. But we shall see later on that *reducing* the value of support enhances an aspect of coverage, which resolves this apparent ambiguity.

2.1.3 Significance

Intuitively, for the intruder, the maximal breach path is the best path to take within the field, because he *knows* that the closest sensor he will encounter is at the farthest possible distance. For the same reason, given S, this is the most insecure path possible in the field. Thus from the point of view of the *defender*, this path stands for the *worst-case coverage* scenario. This explains why maximal breach is referred to as the worst-case coverage measure in [1].

On the other hand, clearly the chances of detecting an intruder along some path improves when the support value for that path is low. Along the maximal *support* path, the intruder knows that most sensors are quite nearby. He is sure to avoid this path. Thus, from the *defender*'s angle, this corresponds to the *best-case coverage* scenario.

In a battlefield, for example, the field A is an enemy camp and the set S stands for sentinel locations. An enemy infiltrator (*intruder*) would like to know the safest path through the camp. At the same time, the defender would like to know the most poorly guarded paths through the field and secure them. On the other hand, the maximal *support* path is the most dangerous path to follow for the intruder. For an alternate example, consider forest-fire detection. The maximal breach path here signifies the segments in the forest where fires are most likely to spread undetected.

2.2 Prior Work on Maximal Breach and Support

There are uncountably many paths connecting any pair of points in A. How do we find the two special paths $P_b(i, f, S)$ and $P_s(i, f, S)$? Two fundamental results are established in [1] that reduce the search space (set of candidate paths) to a *finite* size.

Theorem 1 *At least one maximal breach path must lie along the edges of the* bounded Voronoi diagram *[10] determined by the sensor nodes S and the boundaries of the unit-square field A.*

The solid bold path in Figure 3 is a maximal breach path. Henceforth, we shall denote the Voronoi Diagram of the point set S by $VD(S)$. Each voronoi edge is the perpendicular bisector of the line segment joining its two adjacent sensors. So, if an intruder travelling through the field ever strays from a voronoi edge, he would go *nearer* to either of the adjacent sensors. This explains why an intruder should always stick to paths comprising voronoi edges only, and hence the theorem.

The analogous result for support paths is the following [1].

Theorem 2 *At least one maximal support path must lie along the edges of the Delaunay Triangulation [10] of S.*

The dashed bold path in Figure 3 is a maximal support path. Henceforth, we shall use $DT(S)$ to denote the Delaunay Triangulation of the point set S.

It is not difficult to see why this theorem is true. Firstly, observe that a delaunay edge connects two sensors. Thus at the endpoints of the edge, the coverage is the maximum possible. So, any best-case coverage path should always include the endpoints of delaunay edges. Second, as any delaunay triangle has the property that its circumcircle cannot contain a site (sensor) in its interior. From this, it follows that, within any delaunay triangle, the distance from the closest sensor is maximised at one of the three midpoints of the sides. So a best-case coverage path should always include delaunay edges as well. Refer to [4] for a rigorous proof of this theorem.

We have confined our paths to follow only voronoi (or delaunay edges) - but what do we do about the starting and ending points, i and f, that do not coincide with voronoi vertices (or some $s_i, s_f \in S$)? The approach followed by [1, 4] is to connect i and f to their nearest voronoi vertices (resp. sensor nodes), and follow voronoi (resp. delaunay) edges in between. We do away with this technicality, justifying ourselves with the observation that an intruder seeking a maximal breach path will prefer to start and end his journey on voronoi vertices. Similarly, an agent seeking the maximal support path will start and end his tour on sensor nodes. More on this later on.

From the set of sensor nodes S, we can derive a *unit disc graph* [4] $UDG(S)$, a *Gabriel Graph* [20] $GG(S)$ and a *Relative Neighbourhood Graph* [19] $RNG(S)$ as follows. Since the sensors are all identical, assume their sensing regions are discs of (normalised) radius 1[1]. There is a node in $UDG(S)$, $GG(S)$ and $NG(S)$ for every $s \in S$. Denote the euclidean distance between $s_u, s_v \in S$ by $d(s_u, s_v)$. Let $disc(s_u, s_v)$ denote the circle drawn with $\overline{s_u s_v}$ as diameter. Finally, let $lune(s_u, s_v)$ denote the *lune* formed by s_u and s_v[2]. Put an edge (s_u, s_v) in

1. $UDG(S)$ iff $d(s_u, s_v) \leq 1$;

[1] The dimensions of the field A are scaled appropriately

[2] A lune is the common region of two unit-radius circles centred on s_u and s_v.

2. $GG(S)$ iff $disc(s_u, s_v)$ contains no $s \in S \setminus \{s_u, s_v\}$ in its interior;

3. $RNG(S)$ iff $lune(s_u, s_v)$ contains no $s \in S \setminus \{s_u, s_v\}$ in its interior.

Theorems 1 and 2 make the computation of maximal breach and support paths possible by limiting the search spaces to the finite ($O(N)$) set of voronoi and delaunay edges respectively. S is said to be *connected* when $UDG(S)$ is connected. Under the assumption that S is connected, [4] has shown that the search space for the maximal *support* path can be progressively restricted as follows. There is a maximal support path that uses only

1. The edges of $UDG(S)$.

2. The edges of $DT(S)$.

3. The edges of $GG(S)$.

4. The edges of $RNG(S)$.

As we go down the above list, the graphs get sparser - $UDG(S)$ might have $O(N^2)$ edges, while $DT(S)$ has at most $3n - 6$ edges. $RNG(S)$ is the sparsest of the lot. This means that $RNG(S)$ can be constructed efficiently in a distributed manner with the local (meagre) computing resources embedded within the sensors. Li et al. [4] have used this to develop *distributed* algorithms for computing maximal support paths. But there is no known distributed algorithm for computing the maximal *breach* path. The algorithm devised by [1] was the first centralised polynomial time algorithm for computing maximal breach. In §3.4 we shall present what we believe is the first centralised polynomial time algorithm for maximal breach that gives exact results (does not need the integral weights assumption of [1]) and also computes the maximal breach *path*. In the rest of this thesis, we shall confine ourselves to centralised algorithms.

2.2.1 Centralised Algorithms and Associated Graphs

In the algorithm proposed by [1] as well as our own algorithms, a graph-theoretic abstraction is used to compute the maximal breach and support paths from $VD(S)$ and $DT(S)$. A weighted, undirected graph, called the associated graph, G_{VD} (or G_{DT}) is computed from $VD(S)$ (or $DT(S)$).

For each voronoi vertex in $VD(S)$, there is a node in G_{VD}. Additionally, the peripheral edges of $VD(S)$ intersect the boundaries of A. There is a node in G_{VD} for each such intersection point. Finally, the four corners of A are added to the node set of G_{VD}. There is an edge $E = (u, v)$ in G_{VD} iff the corresponding points in $VD(S)$ are connected by a voronoi edge or are adjacent points on the boundary of A. In the former case, the weight $w(E)$ is set to $breach(E)$ (a quantity proportional to the length $|\overline{ss'}|$, where s, s' are the sensors that share the voronoi edge; see §3.3.2, Eqn. 6); in the latter case, the weight $w(E)$ is set to its distance from the nearest site. See Figure 4. It is easy to see that G_{VD} can be computed in $O(N)$ time, because we only need to traverse cyclically the edges of $VD(S)$ for each face $VOR(s_i)$.

Similarly, for each site $s_i \in S$, there is a node in G_{DT}. There is an edge $E = (s_i, s_j)$ in G_{DT} iff s_i and s_j are connected by a delaunay edge. The edge-weight $w(E)$ is set to $support(E)$ (equal to $|\overline{s_i s_j}|/2$; see §3.3.2, Eqn. 9). G_{DT} can be computed in $O(N)$ time too.

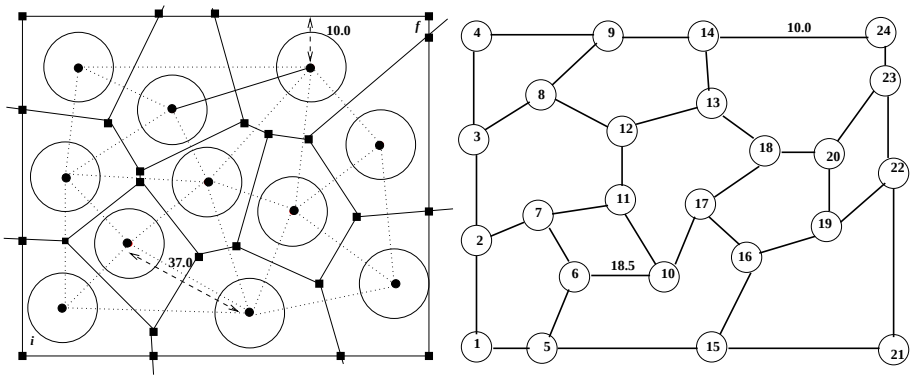

Figure 4: **Associated graphs:** $VD(S)$ and G_{VD}.

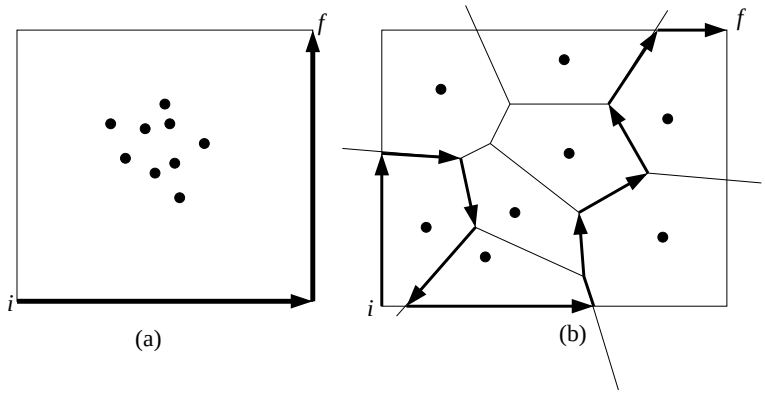

Figure 5: **Exceptions to Theorem 1:** (a) Sensors clustered near the centre. The intruder won't walk along the Voronoi edges at all. (b) A more general scenario.

There is one small, but significant aspect in which our construction of G_{VD} from $VD(S)$ differs from that of [1]. Consider the scenario of Figure 5(a), where the sensors are clustered near the centre of the field. An intruder with knowledge of these coordinates would never walk along the Voronoi edges in this case - he would rather walk right, and then up, along the boundary of the field, in order to maximise his distance from any of the sensors. This illustrates a special case in which the maximal breach path does *not* lie along Voronoi edges. Megerian et al. [1] fail to consider, or at least mention, this special case. Our implementation takes care of this - along with the Voronoi edges, we insert edges into the associated graph corresponding to the segments on the bounding box that are created by intersections with Voronoi edges. Figure 5(b) depicts a general scenario where the sections of the maximal breach path comprise a mix of Voronoi edges and segments on the bounding box.

Now we give the algorithm of Megerian et al.[1].

1. The associated graph G_{VD} is computed from $VD(S)$ by the method outlined earlier.

2. From G_{VD}, the maximal breach is computed using a combination of binary search and *BFS*. The minimum and maximum edge weights are computed and a search-criterion, *breach_weight* is set to the mid-point of this range. At every iteration,

   (a) All edges in the associated graph with weights *greater* than *breach_weight* are dropped.

   (b) Using *BFS*, if a path can be found between the nodes corresponding to i and f, the search is continued with *breach_weight* shifted to the mid-point of the interval [*min_weight*, *breach_weight*]. Otherwise, the search is continued with *breach_weight* shifted to the mid-point of [*breach_weight*, *max_weight*]. Observe, however, that binary search is performed on the *range* of weights, rather than the actual weights.

The reader is referred to [1] for an illustration of this algorithm. The maximal support path can also be computed by running a close variant of the same algorithm on G_{DT}.

What the above algorithm computes is not exactly the maximal breach *path*, but a certain *critical* edge, $e^b_{cr}(i, f, S)$. e^b_{cr} has the following properties:

1. $breach_{max}(i, f, S) = w(e^b_{cr})$.
2. e^b_{cr} is the *lightest* edge in $P_b(i, f, S)$.
3. Take any other path $P(i, f)$ connecting i, f along a sequence of voronoi edges. Let e be the lightest edge in P. Then $w(e) \leq w(e^b_{cr})$.

Henceforth, we shall call this special edge e^b_{cr} (or e^s_{cr} for the maximal support path) the *critical edge* for breach (or support).

Megerian et al. [1] have also suggested a heuristic for lowering the value of maximal breach. They compute the maximal breach for an initial random configuration S, and then deploy additional sensors along the critical edge. They have presented simulation results showing that this simple strategy gives quite a good improvement in the value of maximal breach even after a comparatively large number (say 100) of sensors have already been deployed.

2.2.2 Drawbacks

The algorithm of Megerian et al. outlined above suffers from the following drawbacks:

1. **Exact results only for integral weights.** Since search is done on the *range* of weights, the algorithm finds *exact* value of the maximal breach *only* if the edge weights are all *integral* - but this is an unreasonable assumption, since the weights are derived from euclidean distances.

2. **Time Complexity.** Pruning of edges and *BFS* run in time linear in the number of nodes and edges of the associated graph. From [9], we know that the latter are $O(N)$. Thus, each search iteration runs in $O(N)$ time. However, since binary search is performed on the range [*min_weight*, *max_weight*], we might need, in the worst case, $\log C$ iterations, where C = *max_weight* − *min_weight*. This makes the overall complexity of the algorithm, excluding the construction of $VD(S)$, $O(N \log C)$. Thus, the time-complexity is dependent on the magnitude of the range of weights - the larger the range of weights, the more the runtime, even for graphs of the same topology. We remove this difficulty by proposing a modification on their algorithm that takes $O(N \log N)$ time.

3. The algorithm, as it stands, does not compute the maximal breach *path*, it just computes the maximal breach *value*.

From the theoretical point of view, the methods and algorithms of Megerian et al., though intuitively correct, have not been *proved* formally. For example, the breach reduction heuristic that has been proposed, though good in practice, has been used without proof. The same applies for the algorithm described above.

2.3 Improvement

In the next section, we show that a simple modification to the algorithm of Megerian et al. [1] removes the first two drawbacks. We present the modified form of the algorithm and then formally prove its correctness and derive its complexity. We then present an entirely new way of solving the same problem that leads us to the algorithm **McaBreach**. This algorithm is fast, provides exact results for all values of edge-weights, and computes the maximal breach *path* at no extra cost in terms of time.

We also state and prove four fundamental theorems governing the behaviour of the maximal breach measure under insertion and deletion of sensor nodes to/from S. We show that the heuristic of Megerian et al., along with many other important, and sometimes surprising results, follow from these four theorems. These, in effect, put the whole of the work discussed in this paper, on a sound theoretical basis.

3 Single-Pair Maximal Breach/Support: Computation

3.1 Straight-forward Improvement on Previous Algorithm

We make a simple modification of Megerian et al's [1] algorithm and we also prove its correctness Instead of running binary search on the range of edge weights, we do it on a sorted list of edge weights.

Input: The associated graph G_{VD}, a pair of nodes s, t.

Output: The maximal breach $breach_{max}(s,t)$ and the critical edge e.

Method: See Algorithm 1.

This algorithm is almost identical to that of [1]. The reader is referred to that paper for an illustration.

For the correctness and time-complexity of Algorithm 1, we state and prove the following theorem.

Theorem 3 *Algorithm 1 computes the maximal breach in time* $O(N \log N)$.

Proof. For correctness, it is sufficient to show the following.

1. If *BFS* succeeds for some value of *candidate*, then the actual value of maximal breach is at least as large as *candidate*. Thus, it make sense to increase our estimate of *candidate* to $(max\_weight - candidate)/2$ for the next iteration.
2. If, on the other hand, *BFS fails* for some value of *candidate*, maximal breach is *strictly less* than *candidate*. Thus, we must carry out the next pruning with the *smaller* value $(candidate - min\_weight)/2$.

Algorithm 1 BfsBreach.

{**Sort.**}
Sort the edges of G_{VD} in ascending order of weights and store the results in an array *Edges*;
{**Initialise.**}
$lb \leftarrow 0$;
$ub \leftarrow Edges.size()$;
{**Main Loop.**}
repeat
  $mid \leftarrow (lb + ub)/2$;
  $candidate \leftarrow Edges[mid].weight()$;
  Prune G_{VD}**.** Drop all edges from G that have weight larger than *candidate*. Let $G^{'}$ be the resulting graph;
  Run *BFS* on $G^{'}$ starting from node s.
  if t is reachable from s **then**
    Report *success*;
  else
    Report $failure$;
  end if
  if *success* **then**
    $W \leftarrow candidate$;
    $e \leftarrow Edges[mid]$;
    $lb \leftarrow mid + 1$;
  else
    $ub \leftarrow mid - 1$;
  end if
until *lb* and *ub* cross over
$breach_{max}(s,t) \leftarrow W$;
$e_{cr} \leftarrow e$;
return $breach_{max}(s,t)$ and e_{cr};

For the first part, since *BFS* succeeded, there *is* a path P_1 in the pruned graph between i and f. Let e_1 be the edge in P_1 with the minimum weight. $w(e_1) \geq$ *candidate*, because all edges lighter than *candidate* have been dropped. It follows from Definition 2 that the actual value of maximal breach cannot be less than *candidate*.

For the second part, observe that failure of *BFS* implies that the pruned graph has become disconnected. Thus *all* paths P between i and f *must* have at least one edge e with $w(e) <$ *candidate*. This implies maximal breach is strictly less than *candidate*.

To complete the proof, we observe that there are $O(N)$ edges, and thus binary search takes $O(\log N)$ iterations. In each iteration, we do a pruning ($O(N)$) and a BFS ($O(N)$). This makes the overall complexity $O(N \log N)$. □

3.2 Notation.

We will fix the notations that will be used n the rest of the paper.

- In the unrestricted scenario, A denotes the unit-square field, $A \subset \mathbb{R}^2$, over which S is deployed. However, for deriving the "combinatorial footholds" that enable us to deal with hardness/complexity of the problems we pose, we need to restrict ourselves to finite domains. In such cases, A will denote the set of points acting as (1) feasible sensor locations, or (2) feasible starting/ending points of tours made by the intruder within A.
- S denotes the set of sensors (i.e. the set of points where they are located). N denotes the number of sensors ($|S| = N$).
- $VD(S)$ denotes the bounded voronoi diagram determined by S and the boundaries of A. $VE(S)$, $VV(S)$ denote the set of edges and vertices in $VD(S)$ respectively. $G_{VD} = (V_{VD}, E_{VD})$ denotes the associated graph of $VD(S)$.
- $DT(S)$ denotes the delaunay triangulation of A. $DE(S)$ denotes the set of edges in $DT(S)$. $G_{DT} = (E_{DT}, V_{DT})$ denotes the associated graph of $DT(S)$.
- The maximal breach and support paths (values) are clearly functions of both A and S. We shall often use phrases like "average maximal breach in the system $\langle A, S \rangle$" to express this.
- $P_b(i, f, S)$ denotes a maximal breach *path* in $\langle A, S \rangle$ and $b^S(i, f)$ the breach *value*. Similarly, $P_s(i, f, S)$ and $s^S(i, f)$ denote the maximal *support* path and value respectively in $\langle A, S \rangle$.
- $e^b_{cr}(i, f, S)$ and $e^s_{cr}(i, f, S)$ denote the critical edges in $P_b(i, f, S)$ and $P_s(i, f, S)$ respectively.
- For a set of points P, $CH(P)$ denotes the points on the convex hull of P.

3.3 New Formulation of Maximal Breach and Support

In this section we present a series of equations for maximal breach and support that are central to the development of the algorithm. We take a bottom-up approach; i.e., we define the quantities breach and support for *points* in the field A, *paths* in A and *worst-* and *best-case* paths in A. Also, we progressively restrict our domain. We begin with general equations for unrestricted points and

paths. Next, we take advantage of Theorems 1 and 2 to restrict ourselves to the bounded voronoi diagram and delaunay triangulation of S. Finally, we cast our equations in terms of the associated graphs described in §2.2.1. The equations are most useful in the associated graph form. They establish the two striking features of maximal breach and support paths, viz., their relationship with flow networks and the duality between the two measures. The former feature is used directly to derive our new algorithm.

3.3.1 General Equations for Breach and Support

Let $d(x,y)$ denote the euclidean distance between points x and y. Given S and a point $p \in A$, the closest sensor observability at p [4] is defined as

$$I_C(p,S) = \min_{s \in S} d(s,p). \tag{1}$$

We define the quantities breach and support in terms of $I_C(p,S)$.

For a path $P(i,f)$ in A connecting the points $i, f \in A$,

$$breach(P(i,f)) = \min_{p \in P(i,f)} I_C(p,S). \tag{2}$$

Let $\Pi(i,f)$ denote the set of all (infinitely many) paths, within A, connecting i and f. The maximal breach *path* between i and f is the one that maximises the right-hand-side of Eqn. 2. The maximal *breach* between i and f is accordingly defined as

$$\mathbf{breach_{max}(i,f,S)} = \max_{P \in \Pi(i,f)} breach(P) = \max_{\mathbf{P \in \Pi(i,f)}} \min_{\mathbf{p \in P}} \mathbf{I_C(p,S)}. \tag{3}$$

The corresponding equations for *support* are as follows: For a path $P(i,f)$,

$$support(P(i,f)) = \max_{p \in P(i,f)} I_C(p,S). \tag{4}$$

The maximal support *path* between i and f is the one that minimises the right-hand-side of Eqn. 4. The maximal *support* between i and f is given by

$$\mathbf{support_{min}(i,f,S)} = \min_{P \in \Pi(i,f)} support(P) = \min_{\mathbf{P \in \Pi(i,f)}} \max_{\mathbf{p \in P}} \mathbf{I_C(p,S)}. \tag{5}$$

3.3.2 Breach and Support in $VD(S)$ and $DT(S)$

Let $VE(S)$ denote the set of edges in $VD(S)$. The maximal breach path $P_b(i,f,S)$ comprises a set of edges from $VE(S)$. Consider a voronoi edge $E = s_E t_E$ determined by adjacent voronoi cells $VOR(s_i)$ and $VOR(s_j)$. Then s_i, s_j are the sensors that are closest to any point on E. If $s_i s_j$ intersects E, the minimum distance from any point on E to either of s_i and s_j is $|\overline{s_i s_j}|/2$. Otherwise, it is the distance of s_i from the end-point of E nearest to s_i. Thus, for the edge E,

$$breach(E) = \min\{d(s_i,s_j)/2, d(s_E,s_i), d(t_E,s_i)\} \tag{6}$$

For a path $P(i, f) = \{E_1, E_2, \ldots, E_k\}$, $E_j \in VE(S)$, connecting i and f along a sequence of voronoi edges,

$$breach(P(i, f)) = \min_{E_j \in P(i,f)} breach(E_j) \tag{7}$$

Note that, we tacitly assume that i and f are voronoi vertices. Finally, the equation for $breach_{max}$ takes the form

$$breach_{max}(i, f, S) = \min_{P \in \Pi(i,f)} \max_{E \in P} breach(E), \tag{8}$$

where $breach(E)$ is in turn given by the RHS of Eqn. 6. The difference here is that $\Pi(i, f)$ is *finite* - it is the set of all paths, comprising voronoi edges only, that connect the voronoi vertices i and f.

Let $DE(S)$ denote the set of edges in $DT(S)$. The *support*-analogues of Equations 6 and 7 are the following. Here, we work under the tacit assumption that i and f are nodes in the delaunay triangulation (i.e., some $s_i, s_f \in S$). Let E be the delaunay edge connecting s_p and s_q. Delaunay triangulations have the property that the circumcircle of no triangle contains a site in its interior; so for any point on a delaunay edge, the closest sensor is one of the endpoints of the edge - and the distance is maximised at the midpoint.

$$support(E) = d(s_p, s_q)/2 \tag{9}$$

For a path $P(i, f) = \{E_1, E_2, \ldots, E_k\}$, $E_j \in DE(S)$, connecting i and f, along a sequence of delaunay edges,

$$support(P(i, f)) = \max_{E_j \in P(i,f)} support(E_j) \tag{10}$$

Finally, maximal support between two points i and f is given by,

$$support_{min}(i, f, S) = \min_{P \in \Pi(i,f)} \max_{E \in P} support(E), \tag{11}$$

where $support(E)$ is given by the RHS of Eqn. 9. Here also, $\Pi(i, f)$ is finite. It is the set of all paths, comprising delaunay edges only, that connect i and f.

3.3.3 Breach and Support in G_{VD} and G_{DT}

In G_{VD}, an edge $e \in E_{VD}$ is assigned a weight $w(e) = breach(E)$ (Eqn. 6), where E is the corresponding edge in $VD(S)$. Similarly, in G_{DT}, an edge $e \in E_{DT}$ is assigned a weight $w(e) = support(E)$ (Eqn. 9), where E is the corresponding edge in $DT(S)$. Let the graph-nodes s and t correspond to the points i and f. Restricted to associated graphs, the problem of computing the maximal breach and maximal support paths between a pair of nodes s, t can be expressed succinctly in terms of the following equations, which demonstrate the duality between the two measures:

$$breach_{max}(s, t, G_{VD}) = \max_{P \in \Pi(s,t)} \min_{E_i \in P} w(E). \tag{12}$$

$$support_{min}(s, t, G_{DT}) = \min_{P \in \Pi(s,t)} \max_{E_i \in P} w(E). \tag{13}$$

3.3.4 Maximal Breach and Network Flow

We recapitulate some standard terminology from *Network Flow* [12].

Definition 5 *In a flow network $G = (V, E)$, $c : E \to \mathbb{R}$, given a path $P = \langle e_1, e_2, \ldots, e_k \rangle$, $e_i \in E$ between the source s and sink t, the* bottleneck capacity *of P is defined as the* minimum *capacity among* all *edges of P; i.e.,*

$$bottleneck(P) = \min_{e_i \in P} c(e_i) \tag{14}$$

Definition 6 *The* maximum bottleneck path, *$P_{MB}(s, t)$, in a flow network is a path with the largest bottleneck capacity among* all *paths that connect the source s and sink t; i.e. $P_{MB}(s, t)$ maximises the right-hand side of Equation 14.*

Definition 7 *The* maximum bottleneck capacity *of a flow network is the bottleneck capacity of its maximum bottleneck path connecting s and t. Thus, for the flow network $G = (V, E)$, $c : E \to \mathbb{R}$, the maximum bottleneck capacity is given by*

$$bottleneck_{max}(s, t) = \max_{P \in \Pi(s,t)} \min_{e_i \in P} c(e_i) \tag{15}$$

where $\Pi(s, t)$ denotes the set of all possible paths from s to t.

Comparing Equations 12 and 15, it is hard to miss the fact that $breach_{max}$ and $bottleneck_{max}$ are identical quantities.

Observation 1 *The* maximal breach path *for a given sensor configuration S is nothing but the* maximum bottleneck path *of G_{VD} derived from $VD(S)$, when G_{VD} is viewed as a flow network with a chosen pair of vertices i, f acting as source and target, and the edge weights $w(e)$ acting as the link capacities.*

Now, we propose a greedy algorithm **McaBreach** (Algorithm 2) for computing the maximal breach path from the associated graph G. It basically follows the relaxation step of Dijkstra's shortest path algorithm.

3.4 Algorithm and Analysis for Maximal Breach Path

Input: The associated graph G_{VD}, a pair of nodes s, t.
Output: The maximal breach $breach_{max}(s, t)$, the critical edge e and the *maximal breach path* $P_b(s, t)$.
Method: See Algorithm 2.

Algorithm 2 runs just like Dijkstra's algorithm. The node set V_{VD} of G_{VD} is partitioned into R and $V_{VD} - R$, where at any point of time, R contains all the nodes that have been relaxed - the maximum bottleneck path from s to any node u in R is known exactly. At every iteration, a new node j is picked from $V_{VD} - R$ with the highest bottleneck capacity, and, if it so happens that via j, the bottleneck of some adjacent node u improves, $bottleneck[u]$ is updated. For justification, we turn to the following recursive relations. Figure 6 shows a flow network along with the bottleneck capacities of all the nodes computed from the designated source node.

Algorithm 2 McaBreach.

Variables: The following *vectors* are used:
bottleneck[j]: *real*; {Stores the maximum bottleneck over all paths from s to node j.}
isRelaxed[j]: *boolean*; {Flags whether j has been relaxed.}
path[j]: *Node*; {Stores the predecessor of j in the maximum bottleneck path from s to j.}
critEdge[j]: *Edge*; {Stores the critical edge among all paths from the s to j.}
{**Initialise.**}
for all $i \in V_{VD}$ **do**
    bottleneck[i] $\leftarrow$ **unassigned**;
    path[i] $\leftarrow$ **unassigned**;
    isRelaxed[i] $\leftarrow$ **false**;
    critEdge[i] $\leftarrow$ **unassigned**;
end for
{An implementation note: choose *unassigned* to be a negative quantity. This will ensure that s will be the first node to be picked, since its bottleneck is initialised to 0 and those of the other nodes are all initialised to *unassigned*.}
path[s] $\leftarrow s$, *bottleneck*[s] $\leftarrow 0$;
critEdge[s] $\leftarrow (s, s)$;
for all nodes j *adjacent* to s **do**
    critEdge[j] $\leftarrow (s, j)$;
end for
{**Main Loop.**}
repeat
    Pick an *un-relaxed* node, j, with the maximum *bottleneck*[] value;
    isRelaxed[j] $\leftarrow$ *true*;
    for all *un-relaxed* nodes u adjacent to j **do**
        if $j = s$ **then**
            $bn \leftarrow w(j, s)$;
        else
            $bn \leftarrow \min\{$*bottleneck*$[j], w(j, u)\}$;
        end if
        if $bn >$ *bottleneck*[u] **then**
            {Update *bottleneck*[u]}
            bottleneck[u] $\leftarrow bn$;
            path[u] $\leftarrow j$;
            if $w(j, u) > bn$ **then**
                critEdge[u] $\leftarrow$ *critEdge*[j];
            else
                critEdge[u] $\leftarrow (j, u)$;
            end if
        end if
    end for
until *all* nodes have been relaxed
Reconstruct $P_b(s, t)$ from the predecessor array *path*[];
$breach_{max}(s, t) \leftarrow$ *bottleneck*[t], $e \leftarrow$ *critEdge*[t];
return $breach_{max}(s, t)$, e and $P_b(s, t)$;

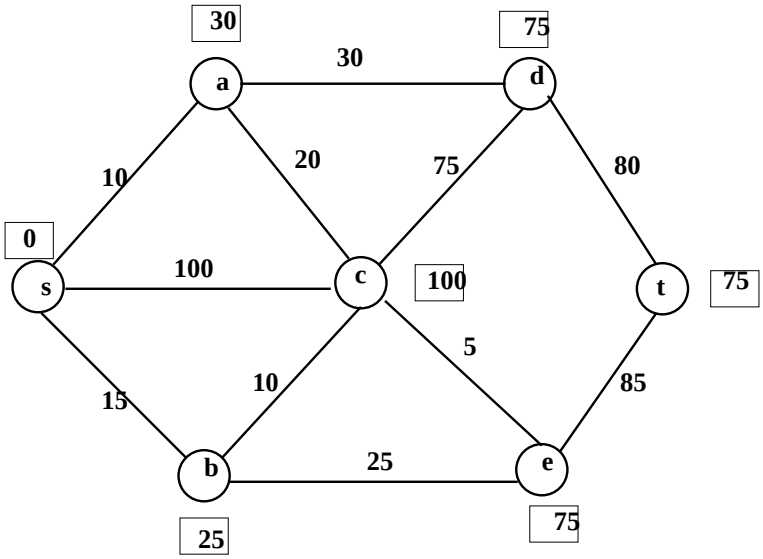

Figure 6: **Maximum bottleneck capacity:** s is the source, so $bottleneck[s] = 0$. (s, c) has the highest capacity among all links incident on s; so $bottleneck[s] = 100$. $bottleneck[b] = 25$ because the maximum bottleneck path from s to b is s, c, d, t, e, b.

1. $bottleneck[s] = 0$
2. Let u be a node adjacent to s such that for any other adjacent node x, $w(s, u) \geq w(s, x)$. Then $bottleneck[u] = w(s, u)$
3. Let u and v be adjacent nodes.The bottleneck of v can be improved via u as follows: $bottleneck[v] = \max\{bottleneck[v], \min\{bottleneck[u], w(u, v)\}\}$.

Here, $bottleneck[u]$ denotes the best known estimate of the maximum bottleneck capacity of node u over all paths from s.

The proof of correctness of Algorithm 2 mirrors that of Dijkstra's shortest path algorithm.

Lemma 1 *At the end of any iteration, R, the set of relaxed nodes has the property that the maximum bottleneck capacity from s to any node in R is correctly known and does not change henceforth.*

Proof. The loop invariant is in force at the end of the first iteration, because at this point $R = s$ and $bottleneck[s] = 0$, which is correct by definition.

The invariant also holds at the end of the second iteration. Note that, after iteration 1, all nodes u with direct edges from s were marked with $bottleneck[]$ estimates equal to $w(s, u)$. Suppose node v is picked, which implies $bottleneck[v] \geq bottleneck[u]$ $\forall u$ adjacent to s. Suppose $bottleneck[v]$ is not the correct (final) maximum bottleneck for node v. Then, there must exist some other path P from s to v, such that $bottleneck(P) > bottleneck[v]$. Since one of the other nodes adjacent to s, say u, must lie on P, this implies $bottleneck[u] \geq bottleneck(P) > bottleneck[v]$ - a contradiction.

Suppose the loop invariant holds at the end of the kth iteration, and during the $(k + 1)$th iteration, node v is picked. Suppose, again, that $bottleneck[v]$ is *not* the maximum bottleneck for v and there exists an alternate path P from s to v that satisfies $bottleneck(P) > bottleneck[v]$. Let x be the first node from $V - R$ that lies on

P. Since $bottleneck[x] \geq bottleneck(P) > bottleneck[v]$, node x should have been picked instead of v. Again, we have a contradiction. □

Theorem 4 *Algorithm 2 computes the maximal breach path in $O(N \log N)$ time.*

Proof. It is sufficient to prove that Algorithm 2 correctly computes the maximum bottleneck capacity from source s to sink t. First, the algorithm *does* terminate, because exactly one node is relaxed at every iteration, and we run the algorithm only as long as there is an unrelaxed node. Thus, by the loop invariant proved in Lemma 1, the maximal breach path from s to t is known when the algorithm terminates.

The complexity of this algorithm is the same as that of Dijkstra's shortest path algorithm - $O(|V| \log |V| + |E|)$ with a Fibonacci heap implementation [12] - which translates to $O(N \log N)$ here. □

The principal merits of Algorithm 2 are:

1. It does not assume integral weights.
2. It is fast. With a good implementation e.g. one using Fibonacci heaps [12] - it outperforms Algorithm 1.
3. What makes this algorithm significantly more useful than Algorithm 1 is that it computes the actual maximal breach *path* on the fly, at no extra cost in terms of time and space. Algorithm 1 returns only the maximal breach and the critical edge; it does *not* compute the breach path.
4. Finally, as we shall see in §3.5, Algorithm 2 affords an almost trivial extension to an algorithm computing the maximal support path as well.

Table 1 shows a comparison of the running times (in μsecs) of Algorithms **BfsBreach** and **McaBreach** for random configurations of 25, 50, 75 and 100 sensors respectively. Because of the integral weights problem of the original algorithm of [1], we do not compare **McaBreach** directly with it. Instead, we compare **McaBreach** with our modified version of Megerian et al's algorithm, **BfsBreach**. We have used *LEDA* [11] for all our experiments. The programmes have been run on a standard Pentium 4 PC running the Fedora Core 4 distribution of Linux. The data corroborates our claim that the latter algorithm out-performs the former, with the added feature of computing the maximal breach *path* at no extra cost.

3.5 An Algorithm and Analysis for Computing Maximal Support.

The dual nature of $breach_{max}(i, f, S)$ and $support_{min}(i, f, S)$, as expressed in Equations 12 and 13, suggests that these measures can be computed by virtually the same algorithm. The following algorithm shows how.

Input: G_{DT}, the associated graph of $DT(S)$, and the source and target nodes s and t.

Output: $support_{min}(s, t)$, the maximal support path $P_s(s, t)$ and the critical edge e.

Method: See Algorithm 3.

From G_{DT}, we build another graph G'_{DT} that is identical to G_{DT} in all respects except edge weights. Let w_{max} denote the weight of the heaviest edge in G_{DT}. Then each edge $e' \in G'_{DT}$ is

# Sensors	BfsBreach CPU Time	McaBreach CPU Time	% Gain
25	349308	192398	44.9
25	366147	40268	89.0
25	223421	185185	17.1
50	1243582	802282	35.5
50	738003	482315	34.6
50	1252359	478352	61.8
75	2297251	1098142	52.2
75	2016794	1065805	47.4
75	2118170	1214090	42.7
100	3118262	2005446	35.7
100	3107053	1294002	58.4
100	3251317	3408833	−4.8
100	3058631	681152	77.7
100	3388012	2399938	29.2

Table 1: **Experimental Data.** A Comparison of the Running Times of Algorithms 1 and 2. All figures are in μseconds.

Algorithm 3 McaSupport

1: $w_{max} \leftarrow w(e_{max})$; {$e_{max}$ is the heaviest edge in G_{DT}.}
2: **for all** $e \in E_{DT}$ **do**
3: $\quad w(e) = w_{max} - w(e)$;
4: **end for**
5: $\langle support_{min}, P_s, e\rangle \leftarrow$ **McaBreach**(G_{DT});
6: $support_{min} \leftarrow w_{max} - support_{min}$;
7: **return** $support_{min}(s,t)$, $P_s(s,t)$ and e;

assigned the weight $w'(e') = w_{max} - w(e)$ where e is the corresponding edge in G_{DT}. Now we compute $breach_{max}(s,t)$ on G'_{DT}. We can call either of Algorithms 1 and 2 for this, but the latter has the advantage of returning the maximal support *path* as well. Since the transition from G_{DT} to G'_{DT} can be done in linear time, **McaSupport** runs in $O(N \log N)$ time too. That the algorithm correctly computes $support_{min}(i, f, S)$ is obvious. The following relationship justifies it.

$$\min_{P \in \Pi(s,t)} \max_{e \in P} w(e) = w_{max} - \max_{P \in \Pi(s,t)} \min_{e \in P} (w_{max} - w(e)). \tag{16}$$

4 Single-Pair Maximal Breach/Support: Optimisation and Decision Problems

The coverage problem is, in general terms: *Given a number of sensors, how to deploy them so as to achieve the maximum degree of coverage at every point on the field.* Huang and Tseng [5] give a precise statement of this problem. However, we emphasize this should be viewed as a *class* of problems rather than a single one. The number of available sensors and their sensing ranges/angles serve as constraints, and the objective is to minimise or maximise a predefined measure of coverage. Our measures of interest are $breach_{max}$ and $support_{min}$. We assume that the sensing regions of the sensors can be approximated by a circular disc of radius r.

For both the measures we have chosen, the optimisation problem is a *minimisation* problem. The defender would try to secure the weakest segments of the field by reducing the minimum distance from a sensor that the intruder *must* encounter along any path. Similarly, minimising support further strengthens the *most* secure segments of the field. We state the problems formally below.

In what follows, $A \subset \mathbb{R}^2$ denotes a unit-square field over which a set of sensors S is to be deployed.

4.1 Optimisation Versions

The breach/support optimisation problem, in its most general form, is

Problem 1 ***[P0]*** *Given A and two points i and f in A, find S such that $breach_{max}(i, f, S)$ (or $support_{min}(i, f, S)$) is minimised.*

However, instead of attacking the problem in its general form **[P0]**, we shall look at *two* flavours of optimisation. The defender may have two design goals:

1. Optimal *Coverage*: Given a set of N sensors, the defender wants to deploy the sensors so as to achieve the best value of *breach* (or *support*).
2. Optimal *Number* of Sensors: Given a desired coverage threshold (upper bound for breach and support), the defender wants to meet it with the *minimum* number of sensors.

Accordingly, we frame two variants of the optimisation problem.

Problem 2 ***[P1]*** *Given A, two points i and f in A, and a set of N sensors, find an arrangement of the sensors such that $breach_{max}(i, f, S)$ (or $support_{min}(i, f, S)$) is minimised.*

Problem 3 ***[P2]*** *Given A, two points i and f in A, and a positive real number T, find the smallest set of sensors S such that $breach_{max}(i, f, S) \leq T$ (or $support_{min}(i, f, S) \leq T$).*

4.2 Decision Versions

Are the problems **[P1]** and **[P2]** tractable or intractable? To answer these questions, we frame the *decision* versions of the problems.

Problem 4 ***[P1-DEC]*** *Given A, two points i and f in A, a set of N sensors and a positive real number T, is it possible to arrange the sensors such that $breach_{max}(i, f, S) \leq T$ (or $support_{min}(i, f, S) \leq T$)?*

Problem 5 ***[P2-DEC]*** *Given A, two points i and f in A, a positive real number T, and a positive integer N, does a set of sensors S exist such that $|S| \leq N$ and $breach_{max}(i, f, S) \leq T$ (or $support_{min}(i, f, S) \leq T$)?*

Evidently, **[P1-DEC]** is a special case of **[P2-DEC]**; in the former we are asked to meet a threshold with *exactly N* sensors. In the remainder of this section, we address the four problems stated above. In fact, we prove a very strong result. All these problems are trivial when the coverage measure under study is *single-pair* maximal breach. Before that, we present a local tightening procedure that, when applied to an initial configuration S, makes small local perturbations to arrive at the lowest value of breach achievable without altering the topology of S. This result is mainly of theoretical interest, though.

5 Single-Pair Maximal Breach Under Insertion and Deletion of Sensors: Theorems

As more and more sensors are deployed over the field A, the value of maximal breach should reduce for any given pair of initial and final positions i and f. The opposite should happen when sensors are *removed* from the field. The following four theorems describe the behaviour of maximal breach under insertion and deletion of sensors.

5.1 The Monotonicity of Maximal Breach

Theorem 5 *Let S and S' be two arrangements of sensors over the unit-square field A such that $S \subset S'$ (i.e. S' is formed by adding one or more sensors to the configuration S). Then, for any two points $i, f \in A$,*

$$breach_{max}(i, f, S') \leq breach_{max}(i, f, S)$$

Proof. We will prove that for any S^+ obtained by adding exactly *one* sensor to S,

$$breach_{max}(i, f, S^+) \leq breach_{max}(i, f, S)$$

The general result will then follow easily by induction on $|S' - S|$.

Let $P_b^S(i, f)$ and $P_b^{S^+}(i, f)$ denote the maximal breach paths for S and S^+ respectively, and b^S and b^{S^+} the corresponding maximal breach *values*. Suppose, for contradiction, $b^{S^+} > b^S$. Note that $P_b^{S^+}(i, f)$ is a path

connecting i and f in A, though not one comprising solely the the edges of $VD(S)$[3]. Since P_b^S, *not* $P_b^{S^+}$, was the maximal breach path for S, we must have,

$$\min\{I_C(p,S) \mid p \in P_b^{S^+}\} \leq b^S. \tag{17}$$

But, addition of the sensor s made $P_b^{S^+}$ the new maximal breach path. Since we assumed that the value of breach *increased*, $\exists p \in P_b^{S^+}$ such that $I_C(p,S^+) = d(p,s) = b^{S^+} > b^S$. Comparing this with Eqn. 17 shows that b^{S^+} cannot be the minimum $I_C(p,S^+)$ value for points p in $P_b^{S^+}$. That is, $breach_{max}(i,f,S^+) \neq b^{S^+}$, a contradiction. □

The following theorem is the converse of Theorem 5.

Theorem 6 *Let S and S' be two arrangements of sensors over the unit-square field A such that $S \supset S'$ (i.e. S' is formed by deleting one or more sensors from the configuration S). Then, for any two voronoi vertices i and f common to both $VD(S)$ and $VD(S')$,*

$$breach_{max}(i,f,S') \geq breach_{max}(i,f,S)$$

If a common pair of vertices cannot be found for $VD(S)$ and $VD(S')$, choose any two corners of A for i and f.

Proof. As above, we will prove that for any S^- obtained by deleting exactly *one* sensor from S,

$$breach_{max}(i,f,S^-) \geq breach_{max}(i,f,S)$$

The general result will follow by induction on $|S - S'|$.

Suppose the deletion of a point $s \in S$ resulted in a reduction of the maximal breach. Then, we have $S^- \subset S$ and a point s such that insertion of s into S^- results in an *increase* in maximal breach. This contradicts Theorem 5. □

5.2 Critical Regions for Sensor Insertion and Deletion

Now that we know that adding a new sensor to S will result in a value of maximal breach that is less than or equal to the initial value, the natural thing to do is to look for a region $A^{ins}(i,f,S) \subset A$ that guarantees that $breach_{max}(i,f,S)$ *reduces* whenever the new point $s \in A^{ins}(i,f,S)$. In what follows, $C(p,q,r)$ will denote the circle through the points p, q and r.

Theorem 7 *Let a and b be the endpoints of the critical edge $e_{cr}(S)$ of $P_b^S(i,f)$, the maximal breach path in $VD(S)$, and let s_0 and s_1 be the corresponding sites. Then*

$$A^{ins}(i,f,S) = C(s_0,a,s_1) \cup C(s_0,b,s_1)$$

is a region such that insertion of any point s within it will guarantee that

$$b^{S^+} = breach_{max}(i,f,S^+) < breach_{max}(i,f,S) = b^S,$$

where $S^+ = S \cup \{s\}$.

[3]Since $P_b^{S^+}(i,f)$ follows the edges of $VD(S^+)$, it might contain some of the *new* edges in $VD(S^+)$ that were not present in $VD(S)$.

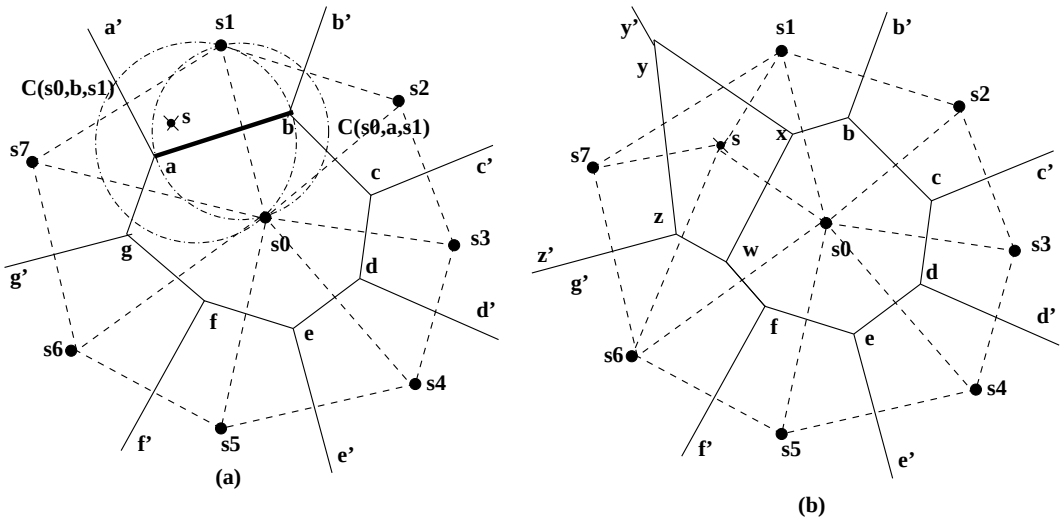

Figure 7: **Critical region for sensor insertion.** (a) The union of the circles $C(s_0, a, s_1)$ and $C(s_0, b, s_1)$ defines the region in which the insertion of any point s *must* shift the maximal breach path. (b) The scene after inserting s.

Proof. The voronoi edge $\overline{ab}$ is the locus of all points q such that the *maximum empty circle (MEC)*, C_q, centered at q will contain the points, and *only* the points, s_0 and s_1 on their perimeters [9]. See Figure 7(a). As boundary cases, the *MEC*s C_a and C_b contain exactly *three* points ($\{s_0, s_1, s_7\}$ and $\{s_0, s_1, s_2\}$ respectively), on their perimeters. Thus, within the region $C(s_0, a, s_1) \cup C(s_0, b, s_1)$, there cannot exist another site s_k such that $d(s_0, s_k) < d(s_0, s_1)$ or $d(s_1, s_k) < d(s_0, s_1)$. The introduction of a new sensor s in this region violates precisely this condition. As a consequence, two new voronoi edges, $\overline{xy}$ and $\overline{xw}$ (bisecting $\overline{ss_0}$ and $\overline{ss_1}$ respectively), are introduced that have weights lower than that of $\overline{ab}$ (see Figure 7(b)).

Now, suppose $\overline{xb}$ (the shortened version of the original edge $\overline{ab}$) is still the critical edge in $P_b^{S^+}$. But this path must contain either $\overline{xy}$ or $\overline{xw}$ because one must either *enter* $\overline{xb}$ or *exit* $\overline{bx}$ via one of these edges. But we have just proved that both $\overline{xy}$ and $\overline{xw}$ have lower weights than $\overline{xb}$. This contradicts the premise that $\overline{xb}$ is the critical edge of $P_b^{S^+}$.

It is sufficient to ensure that $b^{S^+} < b^S$, because, either $P_b^{S^+}$ was also a path in $VD(S)$ (in which case it lost out to P_b^S as the maximal breach path between i and f in $VD(S)$) or it uses some of the new edges in $VD(S^+)$ - in either case the breach of $P_b^{S^+}$ is less than b^S. □

Note that Theorem 7 expresses a *sufficient* condition for the alteration of maximal breach path, but not a *necessary* one - insertions *outside* $A^{ins}(i, f, S)$ might reduce breach as well. Introduction of a new sensor inside this region guarantees a reduction in breach except in the very rare case where there exists *two* paths in $VD(S)$ with the same breach value.

Conversely, we might want to determine the set of sensors $A^{del}(i, f, S) \subset S$ such that *deletion* of any $s \in A^{del}(i, f, S)$ guarantees that $breach_{max}(i, f, S)$ *increases*.

Theorem 8 *Let s_0 and s_1 be the sites across the critical edge $e_{cr}(S)$ of $P_b^S(i, f)$, the maximal breach path in $VD(S)$, and let*

$$A^{del}(i, f, S) = \{s_0, s_1\}.$$

Then the deletion of a point $s \in S$ will guarantee

$$b^{S^-} = breach_{max}(i, f, S^-) > breach_{max}(i, f, S) = b^S$$

if and only if $s \in A^{del}(i, f, S)$, where $S^- = S \setminus \{s\}$.

Proof. Suppose, for contradiction, that the deletion of some site $s \in S - A^{del}(i, f, S)$ resulted in $b^{S^-} > b^S$. Thus, the erstwhile critical edge $e_{cr}(S)$ remains intact. We can assume without loss of generality that the site s is remote from P_b^S and the entire path P_b^S remains intact in $VD(S^-)$ - disturbing some non-critical edge in P_b^S will merely remove some existing edges and add some new ones with higher weight. So $e_{cr}(S)$ still determines the breach value of the resultant path.

Can $P_b^{S^-}$ belong to $VD(S)$? No; because, by the preceding argument, P_b^S belongs to $VD(S^-)$ and $P_b^{S^-}$ cannot be the maximal breach path (it lost out to P_b^S as the maximal breach path in $VD(S)$).

Let $\Delta(S, S^-) = VE(S) \setminus VE(S^-)$ and $\Delta(S^-, S) = VE(S^-) \setminus VE(S)$. So, $P_b^{S^-}$ *must* contain some edges from $\Delta(S^-, S)$. That is, $P_b^{S^-} = \langle e_1, e_2, \ldots, e_k \rangle$ must satisfy the following properties:

1. There exist integers $m_1, \ldots, m_l$ and $n_1, \ldots, n_l$, $1 \le m_1 \le n_1 \le \ldots \le m_l \le n_l$, such that,

   - $e_1, \ldots, e_{m_1 - 1} \in VE(S)$,
   - $e_{m_1}, \ldots, e_{n_1} \in \Delta(S^-, S)$,
   - $e_{n_1+1}, \ldots, e_{m_2-1} \in VE(S)$,
   - $e_{m_2}, \ldots, e_{n_2} \in \Delta(S^-, S)$,

   and so on, until,

   - $e_{n_l+1}, \ldots, e_k \in VE(S)$.

   Put the edges $e_{m_i}, \ldots, e_{n_i}$, $1 \le i \le l$, in the the set $New(P_b^{S^-})$ and the rest in $Old(P_b^{S^-})$.

2. Let $P' = P_b^{S^-} \cap Old(P_b^{S^-})$. The crucial observation here is that P' is a subpath of some path $Q(i, f)$ in $VD(S)$. Why? Observe that all edges in P' were also edges in $VD(S)$ and more importantly, $e_{m_j - 1}$ and e_{n_j+1} were *connected* in $VD(S)$[4]. This implies, $\min\{e_i \mid e_i \in P'\} \le b^S$.

Refer to Figure 8. Now, all the new edges, $\Delta(S^-, S)$, are confined to $VOR(s)$ (in $VD(S)$). Let $S_s = \{s_1, s_2, \ldots, s_p\}$ be the points such that in $VD(S)$, $VOR(s_i)$ shared an edge with $VOR(s)$. Observe that each edge in $\Delta(S^-, S)$ connects some pair of points from $CH(S_s)$. We claim that for any edge e in $Old(P_b^{S^-})$ that was incident upon $VOR(s)$ before the deletion, e cannot have a weight larger than any edge in $\Delta(S^-, S)$. For example, in Fig. 8(a), consider the edge $\overline{d'd}$. It was incident upon $VOR(s)$ in $VD(S)$. The weight of this edge cannot exceed those of $\overline{uv}$, $\overline{vw}$, $\overline{wx}$ or $\overline{xy}$ (Fig. 8(b)). But $P_b^{S^-}$ must use at least one such edge e to enter or exit edges in $\Delta(S^-, S)$. This means the edges in $\Delta(S^-, S)$ are not candidates for being the minimum edge in $P_b^{S^-}$, and thus cannot be $e_{cr}(S^-)$. This, together with property 2 above implies $b^{S^-} = \min\{e_i \mid e_i \in P'\} \le b^S$, which is a contradition.

We have proved one end of the theorem: to increase breach, deletion of either of the points s_0, s_1 is *necessary*. To see that it is sufficient - observe that deletion of s_0 or s_1 destroys the critical edge. Thus the maximal breach must change. But by Theorem 6, the breach can only increase. □

[4]Take, for instance, the edges $c'u$ and xf' in $VD(S^-)$ (Fig. 8(b)). These are the erstwhile edges $c'c$ and ff' in $VD(S)$ (Fig. 8(a)), and were connected (via cd, de, ef, for example).

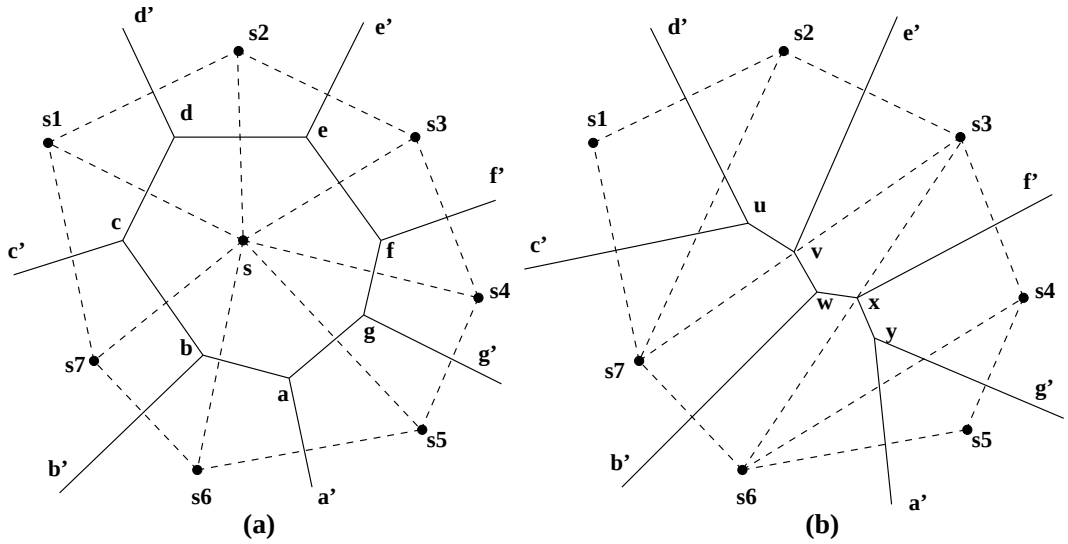

Figure 8: **Critical set for deletion.** (a) Scene before and (b) after deleting s from $VD(S)$. All the new edges created are confined to $VOR(s)$.

5.3 Corollary: Correctness of Insertion Strategy of Megerian et al.

Theorem 7 formally proves the correctness of sensor addition heuristic proposed by [1]. At each step, they determined the critical edge $e_{cr}(S)$ of P_b^S and deployed additional sensors along e_{cr}. But e_{cr} is wholly contained in $A^{ins}(i, f, S)$, so their heuristic is correct. We have a much *stronger* result now:

1. The critical region for sensor insertion to work is much larger than just e_{cr}.
2. The addition of just *one* sensor will do the job at each step.

6 Implication of the Sensor Insertion/Deletion Theorems on Single-Pair Optimisation Problems

As described in §4, one of our objectives is to optimise a coverage measure. In this section, we establish that this is a trivial problem, as long as the measure of interest is *single-pair* breach.

6.1 Intuitive Reasoning

Refer to Figure 9. Suppose i and f are at the bottom-left and top-right corners of the field. The line segment $\overline{ab}$ cuts all possible paths, within the field, connecting i and f. So, if we cluster the N sensors uniformly along $\overline{ab}$, we place a ceiling on the maximal breach: $breach_{max}(i, f, S) = |\overline{ab}|/2N$. Now, if we slide $\overline{ab}$ to the new position $\overline{a'b'}$ (nearer i), we get a lower value $breach_{max}(i, f, S) = |\overline{a'b'}|/2N$. In fact, we can carry on and make $breach_{max}(i, f, S)$ arbitrarily small. Similarly, we can reduce $support_{min}(i, f)$ arbitrarily by clustering all our sensors along the line segment $\overline{if}$.

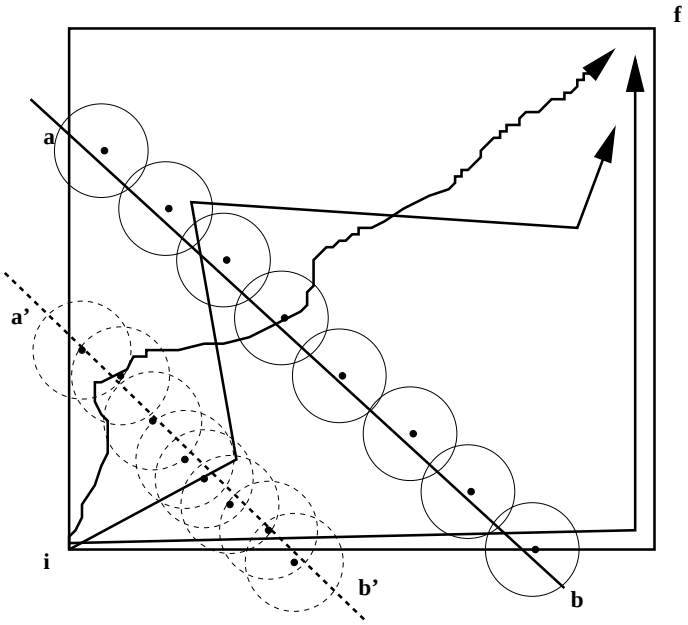

Figure 9: **No lower-bound on breach.** The value of $breach_{max}(i, f, S)$ can be reduced arbitrarily.

The point is, for any designated pair of source and target positions, it is trivial to minimise $breach_{max}(i, f, S)$. We prove this claim formally through theorem 9. But there is a trade-off here. In clustering all our sensors on $\overline{ab}$ or $\overline{a'b'}$, we leave a considerably large region $a'b'fa'$ unattended. A realistic objective would be to simultaneously reduce $breach_{max}(i, f, S)$ for *all* possible pairs of points i and f. The measures defined in the next chapter, $avgBreach_{max}(S)$ and $avgSupport_{min}(S)$, capture this requirement.

6.2 A Theorem

Theorem 9 *Given a unit-square-field A, two points i and f in A and any positive number B, there exists a set of sensors* S_{min}, $|S_{min}| = 8$, *such that* $breach_{max}(i, f, S_{min}) \leq B$.

Proof. We prove the theorem by constructing the set S_{min}. We start with a set of sensors $S = \{s_1, s_2, \ldots, s_N\}$ chosen as follows. Pick any positive number δ and draw circles, $C_\delta(i)$ and $C_\delta(f)$, of radius δ centred on i and f respectively (δ should be small enough so that $C_\delta(i)$ and $C_\delta(j)$ do not overlap). Place three sensors at random on the circumferences of each of $C_\delta(i)$ and $C_\delta(f)$. Call the subset of S made up solely of these 6 dummy sensors S_d. Place the remaining $N - 6$ sensors at random within $A \setminus [C_\delta(i) \cup C_\delta(f)]$. Note that i are f are forced to be vertices in $VD(S)$.

Now call **McaBreach**(i, f, S). If $b^S(i, f) \leq B$, well and good. Else, repeatedly insert additional sensors to S, one at a time, in the critical region $A^{del}(i, f, S)$ until $b^S(i, f) \leq B$. Note that, by virtue of Theorem 7, we are bound to end up with such a set.

Finally, let $e_{cr}(i, f, S)$ be the critical edge of $P_b(i, f, S)$ and s_p, s_q the corresponding sites. Set $S_{min} = \{s_p, s_q\} \cup S_d$. Delete all sensors in $S \setminus S_{min}$. Clearly, by Theorem 8, this does not increase maximal breach.

Thus $breach_{max}(i, f, S_{min}) \leq B$ and S_{min} is our required set. □

The proof of Theorem 9 might appear, at first glance, needlessly convoluted. Afterall, we could have made $breach_{max}(i, f, S) = 0$ by placing sensors at i and f. However, consider a situation where it is impossible to place sensors on i or f (for example, i, f could be located on water-bodies). The geometric significance of the above theorem lies in the fact that it shows the existence of *alternate* locations in A where sensors can be placed to reduce $breach_{max}(i, f)$ without bound.

6.3 Implications

There are two things that we learn from the results of this section:

1. We have solved both of the problems **[P1]** and **[P2]** - it is possible to achieve arbitrarily small values of single-pair maximal breach with a constant number of sensors.
2. In order to ensure good quality surveillance uniformly over the entire field, we should look for some other measures of coverage.

7 *All-Pairs Average* Maximal Breach and Support: Measure and Computation

The extension from maximal breach/support to their *average* counterparts is simple. Instead of confining ourselves to a fixed pair of starting and ending coordinates (of the intruder), we consider all possible pairs of points (i, f) within A. We determine the critical edge of the maximal breach path between each pair of points, and then take the average. But, as before, we need to make our domain finite. It is natural to restrict the set of feasible starting and ending positions to the set of vertices of $VD(S)$ for breach, and to the set S (i.e. the vertices of $DT(S)$) for support. This makes sense, since a voronoi vertex is the center of a maximum empty circle [9, 10]. It is precisely the point that has the maximum value of $I_C(p, S)$ (Equation 1) in its immediate vicinity. So, an intruder would always prefer to land up on a voronoi vertex. Similarly, since each vertex in $DT(S)$ coincides with some $s_i \in S$, support is the maximum possible on the delaunay vertices. We can now define our two new measures in terms of the associated graphs G_{VD} and G_{DT}.

Definition 8 ***[Average Maximal Breach].*** *Let $E_b(S) = \{e \in E_{VD} \mid \exists i, j \in V_{VD}$ such that $e^b_{cr}(i, j, S) = e\}$. In other words $E_b(S)$ is the subset of E_{VD} made up of only the critical edges of maximal breach paths in $\langle A, S \rangle$. Then,*

$$avgBreach_{max}(S) = \frac{\sum_{e \in E_b(S)} w(e)}{|E_b(S)|}. \tag{18}$$

Definition 9 ***[Average Maximal Support].*** *Let $E_s(S) = \{e \in E_{DT} \mid \exists i, j \in V_{DT}$ such that $e^s_{cr}(i, j, S) = e\}$. Then,*

$$avgSupport_{min}(S) = \frac{\sum_{e \in E_s(S)} w(e)}{|E_s(S)|}. \tag{19}$$

Clearly, $|E_b|$ and $|E_s|$ are both $O(N)$. Lemma 3 proves that $E_b(S)$ is nothing but the *Maximum Cost Spanning Tree* [16] of G_{VD}. Thus $|E_b| = |V_{VD}| - 1$.

The significance of average breach/support measures are the following:

1. An "optimal" value of the measure should imply that the given sensor configuration S provides the desired level of coverage *uniformly* all over the field. $avgBreach_{max}$ meets this requirement because the domain over which it is measured - the set of vertices of G_{VD} (which include the voronoi vertices as well as the intersection of the voronoi edges with the boundaries of A), is uniformly dispersed over A. Geometrically, an optimal value of $avgBreach_{max}$ would cause the sensors to spread out uniformly over A. Contrast this with single-pair breach. The optimal value for that measure attracts the sensors towards a cluster around the critical edge.

2. The measure should be *sound* - it should conform to the intuitive requirement that adding sensors to S gives better coverage. $avgBreach_{max}$ meets this requirement - adding sensors to S reduces its magnitude.

3. The measure should be useful in practical scenarios. This is true for $avgBreach_{max}$ because we take into account all *reasonable* trajectories of the intruder within A.

7.1 All-Pairs Maximal Breach/Support: Greedy Algorithm

In §3.4, we developed a greedy algorithm that computes the single-pair maximal breach path. There are two properties of maximal breach paths that indicate that a greedy strategy will do the job for all-pairs average breach as well. In this section, we state and prove those properties and develop an *optimal* algorithm for computing the measure.

7.1.1 Optimal Substructure Property of Maximal Breach Paths

Lemma 2 *Let $P_b(s,t) = \langle v_1, v_2, \ldots, v_k \rangle$, where $s = v_1$ and $v_k = t$, be a maximal breach path between s and t. Then $\forall i, j$ such that $1 \leq i < j \leq k$, $P(v_i, v_j) = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is also a maximal breach path between v_i and v_j. In other words, any sub-path P of P_b is a maximal breach path between the end-points of P.*

Proof. Let $e_{cr} = (v_l, v_{l+1})$ be the critical edge of P_b, where $1 \leq l \leq k-1$. Also, let $b = w(e_{cr})$. Choose any v_i, v_j from $P_b(s,t)$. Let $P = \langle v_i, v_{i+1}, \ldots, v_j \rangle$.

Now, consider an arbitrary path $P' = \langle v_i, v_{r_1}, v_{r_2}, \ldots, v_{r_n}, v_j \rangle$ between v_i and v_j. Then $Q(s,t) = \langle v_1, \ldots, v_i, v_{r_1}, \ldots, v_{r_n}, v_j, \ldots, v_k \rangle$ is another path connecting s to t. So, $\min\{w(e) \mid e \in P\} \leq b$. But the edges in $\langle v_1, \ldots, v_i \rangle$ and $\langle v_j, \ldots, v_k \rangle$, being edges in P_b, are all heavier than e_{cr}. Thus, $\min\{w(e) \mid e \in P'\} \leq b$. On the other hand, since P is a sub-path of P_b, $\min\{w(e) \mid e \in P\} \geq b$. Thus, we have proved that the weight of the lightest edge in P is larger than that of any other path P', which means P is the maximal breach path between v_i and v_j. □

The next logical step would be to look for a method of combining subproblems, and we would have a dynamic programming solution to the problem. Unfortunately, this cannot be done. Given the maximal breach paths $P_b(u,v)$ and $P_b(v,w)$ between u, v and v, w respectively, the concatenation of $P_b(u,v)$ and $P_b(v,w)$ is, in general, *not* the maximal breach path between u and w. For, there

might exist an alternate path $P'_B(u, w)$, *edge-disjoint* with the above two paths, that might well be the maximal breach path between u, w.

Lemma 2 does not go far in giving us the actual algorithm. It merely gives us an inkling about the *nature* of the solution. The following lemma seals the issue.

7.1.2 Maximal Breach Path and *Maximum* Cost Spanning Tree

The *Maximum Cost Spanning Tree* (*MaxST*) of a connected graph can be computed by an $O(|E| \log |E|)$ greedy algorithm that parallels Kruskal's Minimum Cost Spanning Tree algorithm [16]. The associated graph G_{VD} is connected, and $|E_{VD}| = O(N)$. Thus, $MaxST(G_{VD})$ can be computed in $O(N \log N)$ time by first sorting E_{VD} in *descending* order of weights and then running Kruskal's algorithm on G_{VD}. The only difference is that at each step we pick the *heaviest* feasible edge, instead of the lightest one.

Lemma 3 *Suppose T is a Maximum Cost Spanning Tree of G_{VD} computed by the greedy algorithm outlined above. Pick any pair of nodes $s, t \in V_{VD}$. Then the path $P_T(s, t)$ in T between s and t is also a maximal breach path between s and t in G_{VD}.*

Proof. Call the edges in T the *branch* edges, denoted by b_i, and the edges in $G_{VD} \setminus T$ the *arc* edges, denoted by a_j. Let $E_T = \{b_1, b_2, \ldots, b_{|V_{VD}|-1}\}$ denote the set of all branches and $P_T(s, t) = \langle b_{l_1}, b_{l_2}, \ldots, b_{l_k} \rangle$. Also, let $P'(s, t) = \langle e_{r_1}, e_{r_2}, \ldots, e_{r_i}, \ldots, e_{r_p} \rangle$ be another path, in G_{VD}, between s and t.

Suppose $e_{r_i} = a_j = (u, v)$ is the first *arc* in the sequence $P'(s, t)$. Then, $b_{l_s} = e_{r_s}$, $1 \leq s \leq i - 1$ (because $P_T(s, t)$ is a *unique* path in T). Now, a_j could have been omitted from T for two reasons:

1. All the branches were considered before a_j. Then, $w(a_j) \leq \min\{b \mid b \in E_T\}$. In this case, the minimum edge in $P_T(s, t)$ is heavier than that of P'.
2. The introduction of a_j would have created a cycle $s \rightsquigarrow u \rightarrow v \rightsquigarrow s$ in the spanning forest of G_{VD}. This implies, at the point of time a_j was considered, there already existed a path $P_T(s, v)$ between s and v, using *only* branches encountered *before* a_j. Since we wish to maximise the minimum edge of P', we can discard the prefix $\langle e_{r_1}, \ldots, a_j \rangle$ in favour of $P_T(s, v)$.

In this manner, all arcs a_j can be eliminated. They either do not figure as the critical edge, or can be discarded in favour of alternate paths comprising only branches. □

7.2 An Optimal Algorithm and Analysis

Lemma 3 leads directly to a simple algorithm for computing all-pairs average breach. The algorithm follows.
Input: $G_{VD} = (V_{VD}, E_{VD})$, the associated graph of $VD(S)$.
Output: $avgBreach_{max}(S)$.
Method: See Algorithm 4.

Algorithm 4 makes just one addition to Kruskal's algorithm: the assignment in line 18, where the critical edge between a set of node-pairs (i, j) is actually determined. The following lemma justifies the assignment.

Algorithm 4 MaxSTAverageBreach

Variables:
F: Set of Sets. {F stores the spanning forest of G_{VD} at all times.}
E: $Array$ of $Edge$s.
e: $Edge$.
u, v: $Node$.
T_u, T_v, T: Set of $Edge$s.
B_{max}: $|V_{VD}| \times |V_{VD}|$ two-dimensional $Array$ of $Edge$s. {$B_{max}[i, j]$ stores the critical edge of the maximal breach path between i and j.}
$F \leftarrow \{\{1\}, \{2\}, \ldots, \{|V_{VD}|\}\}$. {The nodes of G_{VD} are numbered 1, 2, ...}
$E \leftarrow E_{VD}$.
Sort E in *descending* order of weights.
while $|F| > 1$ **do**
    $e \leftarrow E.pop\_first()$.
    $u \leftarrow e.source()$, $v \leftarrow e.target()$.
    $T_u \leftarrow Find(u)$, $T_v \leftarrow Find(v)$.
    if $T_u \neq T_v$ **then**
        $F.remove(T_u)$, $F.remove(T_v)$.
        $\forall i \in T_u$, $\forall j \in T_v$, $B_{max}[i, j] \leftarrow e$.
        $T \leftarrow Union(T_u, T_v)$.
        $F.insert(T)$.
    end if
end while
$avgBreach \leftarrow \frac{COST(F)}{|V_{VD}|-1}$.

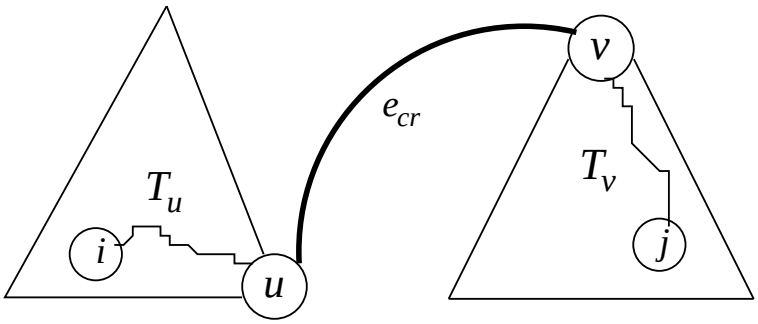

Figure 10: **Computing critical edges bottom-up.**

Lemma 4 *Let $F = \{T_1, T_2, \ldots, T_n\}$ be the Maximum Cost Spanning Forest of G_{VD} just before an edge $e = (u, v)$ is added to F. Let T_u and T_v be trees in F to which the endpoints u and v of e belong. Then, $\forall i \in T_u$, $\forall j \in T_v$, the critical edge between i and j is e.*

Proof. Consider nodes $i \in T_u$ and $j \in T_v$. See Fig. 10. Adding $e = (u, v)$ to F connects T_u and T_v and introduces a path $P_T(i, j)$ between i and j. Moreover, $P_T(i, j)$ is the path connecting i and j in the Maximum Cost Spanning Tree constructed by the algorithm. Thus, by Lemma 3, $P_T(i, j)$ is a maximal breach path between i and j, and e is one of its edges. But, since the algorithm picks heavier edges first, $w(e) \leq \min\{b \mid b \in T_u \cup T_v\}$. Thus e must be the critical edge of $P_T(i, j)$. □

Lemma 4 helps us prove the following loop-invariant for Algorithm 4.

Lemma 5 *Let $F = \{T_1, T_2, \ldots, T_n\}$ be the Maximum Cost Spanning Forest of G_{VD} at the end of the ith iteration ($1 \leq i \leq |V_{VD}| - 1$) of the loop of line 12. Then, for all nodes x and y that are* connected *in F, the critical edge of the maximal breach path between x and y is known, and does not change thereafter.*

Proof. Let b_i denote the branch added to F during the ith iteration, $1 \leq i \leq |V_{VD}| - 1$. The proof is by induction on i.

For the base case ($i = 1$), note that $b_1 = (u_1, v_1)$ is the heaviest edge in G_{VD}. So b_1 constitutes the maximal breach path, as well as the critical edge, between u_1 and v_1. Also u_1 and v_1 are the only nodes connected in F at this point. Thus the loop invariant holds at the end of iteration 1.

Suppose the invariant holds at the end of some iteration $i - 1$. During the ith iteration, the new branch $b_i = (u_i, v_i)$ joined the trees T_{u_i} and T_{v_i}, and connected exactly $|T_{u_i}| \times |T_{v_i}|$ new node-pairs in F. By Lemma 4, b_i is the critical edge for all these pairs. Hence, the invariant holds after the ith iteration as well. □

Finally, we have the following theorem.

Theorem 10 *Algorithm 4 computes the the average breach over all pairs of nodes in G_{VD} in $O(N^2)$ time.*

Proof. Firstly, the algorithm halts because at each iteration exactly one edge is added to F, until there is a single connected component in F. This outcome is guaranteed because G_{VD}, by definition, is connected.

When the algorithm terminates, all nodes in G_{VD} are connected by F. Thus, by Lemma 5, the critical edges between all pairs of nodes is correctly known.

For computing the time-complexity, recall that $|E_{VD}| = O(N)$. Lines 8 and 9 take $O(N)$ time. The sorting in line 10 takes $O(N \log N)$. Within the while loop of line 12, the operations of lines 12 and 13 take constant-time. The *Union-Find* operations (lines 14 and 18) can be done in $O(\log^* N)$. And the set operations of lines 16 and 19 can be done in linear time. Since the while loop runs $O(N)$ times, in the absence of line 17, the loop-complexity would have been $O(N^2)$.

The costliest operation in the loop is done in line 17. By aggregate analysis, we need to populate $O(N^2)$ entries. Thus the entire algorithm runs in $O(N^2)$ time. □

We end this section with some remarks.

1. Algorithm 4 is optimal, because it computes $O(N^2)$ values in $O(N^2)$ time.
2. Algorithm 4 can be modified to report the set of point-pairs i, j for which the maximal breach is the *maximum* possible, without any run-time trade-off. The modification is as follows. After computing the Maximum Cost Spanning Forest F, delete the heaviest edge $b_1 = (u_1, v_1)$ from F. Compute the residual connected components T_{u_1} and T_{v_1} (in linear time). Report the pairs $(i, j) \in T_{u_1} \times T_{v_1}$.
3. If we treat the computation of all-pairs maximal breach as a *preprocessing* step, subsequent queries for $breach_{max}(i, f, S)$ can be answered in *O(N)* time. This is true because we can compute $P_b(s, t)$ by a traversal of the maximum cost spanning tree.

7.3 Yet Another Algorithm for Single-Pair Maximal Breach

Construct the MaxCost Spanning Tree in $O(N \log N)$ time and return the unique path between designated nodes s and t ($O(N)$). Thus we now have three algorithms for computing single pair maximal breach, progressively better ones: BfsBreach, McaBreach, and this one.

8 All-Pairs Maximal Breach: Non-Existence of a Lower-Bound

We again have, as a counterpart of Theorem 9, a negative result.

Theorem 11 *Let $F \subset A$ be a set of N points in A. The points in F act as feasible starting and ending points for an intruder dropped inside A. For any choice of F, and any positive real number B, there exists a set S of $O(N^2)$ sensors such that $avgBreach_{max}(S) \leq B$.*

Proof. Pick any pair of points i and f from F. By Theorem 9, there exists a set of sensors $S_{i,f}$, $|S_{i,f}| \leq 8$, such that $breach_{max}(i, f, S_{i,f}) \leq B$. Now, this is true for *all* $i, f \in F$. Observe that, as far as satisfying the breach upper-bound B is concerned, each pair of points i, f can be treated *independently*. This is because additional sensors can only decrease the value of $breach_{max}(i, f, S_{i,f})$, by Theorem 5.

So, let

$$S = \bigcup_{i,f \in F} S_{i,f}.$$

It follows from the preceding argument that $breach_{max}(i, f, S) \leq B$, $\forall i, f \in F$. Hence, $avgBreach_{max}(S) \leq B$. Moreover, since we have $O(N^2)$ pairs of points, and 8 sensors for each pair, $|S| = O(N^2)$. □

Like in the case of single-pair maximal breach, the theorem above says that all-pairs average has no lower bound for any given set of points acting as feasible starting and ending positions of an intruder's tours through A. However, there is one fundamental difference. In case of single-pair breach, any breach threshold can be met with a *constant* number of sensors. For all-pairs average breach, however, we could potentially need $O(|F|^2)$ sensors. As we shall show below, the problem of minimising the number of sensors while meeting a given average breach threshold is a non-trivial (in all probability, NP-Hard) problem.

9 All-Pairs Maximal Breach/Support: Optimisation/Decision Problems

Theorem 11 says states that with at most $8|F|^2$ sensors (where F is the set of feasible starting and ending points for the intruder), any average breach threshold is achievable. But there is no guarantee that this is the *optimal* number of sensors. Next, we concentrate on the problem of meeting the threshold with the *optimal* number of sensors. Accordingly, we frame the average breach version of problems **[P2]** and **[P2-DEC]** below.

Problem 6 ***[P2-AVG]*** *Given A and a positive real number T, find the smallest set of sensors S such that $avgBreach_{max}(i, f, S) \leq T$ (or $avgSupport_{min}(i, f, S) \leq T$).*

The decision version is:

Problem 7 ***[P2-AVG-DEC]*** *Given A, a positive real number T and a positive integer N, does a set of sensors S exist such that $|S| \leq N$ and $avgBreach_{max}(i, f, S) \leq T$ (or $avgSupport_{min}(i, f, S) \leq T$)?*

In this section, we look into the tractability of problem **[P2-AVG-DEC]**. However, note that the solution space of the problems is uncountable, and as such, not combinatorial in nature. A has uncountably many feasible positions at which sensors from S might be placed. We shall follow the technique used by [14] to restrict the feasible solution space to a *finite* size.

9.1 Decision Problem Restricted to Finite Domain

We restrict the field A to a set of N discrete points on the plane. Without loss of generality we can restrict the points in A to ones with integral coordinates (i, j). The set A represents the *feasible* positions for placing sensors, as well as feasible starting and ending positions of tours made by the intruder.

In this restricted setup, the decision problem **[P2-AVG-DEC]** takes the following form.

Problem 8 ***[P2-AVG-DEC-FINITE]*** *Given A, a positive real number T_b (or T_s) and a positive integer n, does there exist a set of points S, $|S| \leq n$, such that $avgBreach_{max}(S) \leq T_b$ (or $avgSupport_{min}(S) \leq T_s$)? We can encode an instance of this problem by the tuple $\langle A, n, T_b($ or $T_s)\rangle$. Clearly, the size of the problem is determined by $|A| = N$.*

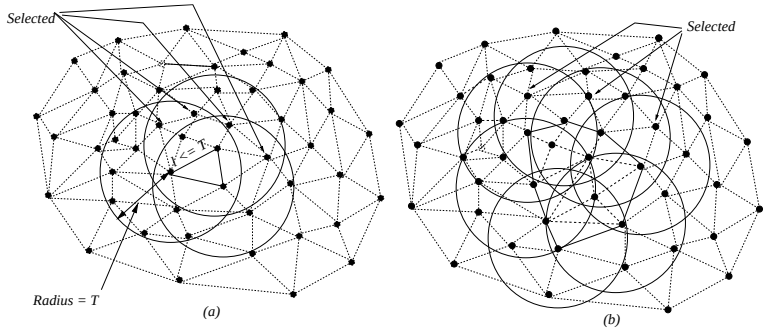

Figure 11: **Illustration of Algorithm 5.**

To the best of our knowledge, the question of hardness of **[P2-AVG-DEC-FINITE]** is open. But we have found some results about two "relaxed" versions of the problem, stated below as Problems 9 and 10.

Suppose, instead of trying to meet the threshold T_s for average support with an optimal number of sensors, we place a threshold on the *maximum* critical edge e^s_{cr}. This amounts to relaxing the problem because placing a ceiling on the maximum critical edge is sufficient to bound average support; but this is not *necessary*. Support paths follow delaunay edges. Geometrically, this amounts to placing a ceiling T_s on the length of the maximum delaunay edge.

Similarly for breach, we place an upper bound T_b on the maximum possible breach value for *any* path between *any* pair of points i and f in A. Then we try to achieve this with an optimal number of sensors.

9.2 Max-Delaunay-Edge-Finite: A "Relaxed" Problem

Problem 9 ***[Max-Delaunay-Edge-Finite]*** *Given A, a positive real number T_s and a positive integer n, select a set S of n points from A such that the length of the longest edge in $DT(S)$ is at most T_s.*

9.2.1 A Polynomial Algorithm

Input: $\langle A, N, n, T\rangle$.
Output: A set S, $|S| = n$, such that $DT(S)$ contains no edge longer than T; $\emptyset$ if no such set exists.
Method: See Algorithm 5.

9.2.2 How it Works

Algorithm 5 is quite straight-forward. First, we triangulate all of A. Then we start with an arbitrary delaunay triangle t whose longest edge is not longer than T. See Fig. 11. Starting from the vertices of t, we progressively augment our set S as follows. Draw circles of radius T at each vertex of $CH(S)$. Find out the points in $A \setminus S$ that belong to the interior of at least *two* circles. Inclusion of these points (e.g. points marked as "Selected" in the figure) does not introduce an edge in $DT(S)$ that is longer than T. So S is augmented with all such points, until we have $|S| = n$. If, on the other hand, we run out of points with which to augment S, we start afresh with a different triangle t' in

Algorithm 5 MaxDelaunayEdge

Compute $DT(A)$;
$S \leftarrow \emptyset$;
for all Triangles $t \in DT(A)$ such that the longest edge in t is at most T units long **do**
    $S \leftarrow t$;
    Polygon $P \leftarrow t$;
    $L \leftarrow$ length of the longest edge in $DT(S)$;
    while $|S| < n$ **and** $L \leq T$ **do**
        Let $\{p_1, \ldots, p_l\}$, $1 \leq k < n$ denote the set of points in P.
        Draw a circle $C_T(p_i)$ of radius T centred on each $p_i \in P$;
        $\Delta_T(A, S) \leftarrow \{q \in A \setminus S \mid \exists C_T(p_i), C_T(p_j)$ such that $q \in C_T(p_i) \cap C_T(p_j)\}$;
        $l \leftarrow |\Delta_T(A, S)|$;
        if $l = 0$ **then**
            break while;
        end if
        if $l \leq n - |S|$ **then**
            $S \leftarrow S \cup \Delta_T(A, S)$;
            $P \leftarrow CH(P)$;
            $L \leftarrow$ length of the longest edge in $DT(S)$;
        else
            Pick any $n - |S|$ points from $\Delta_T(A, S)$ and add them to S;
        end if
    end while
end for
return S.

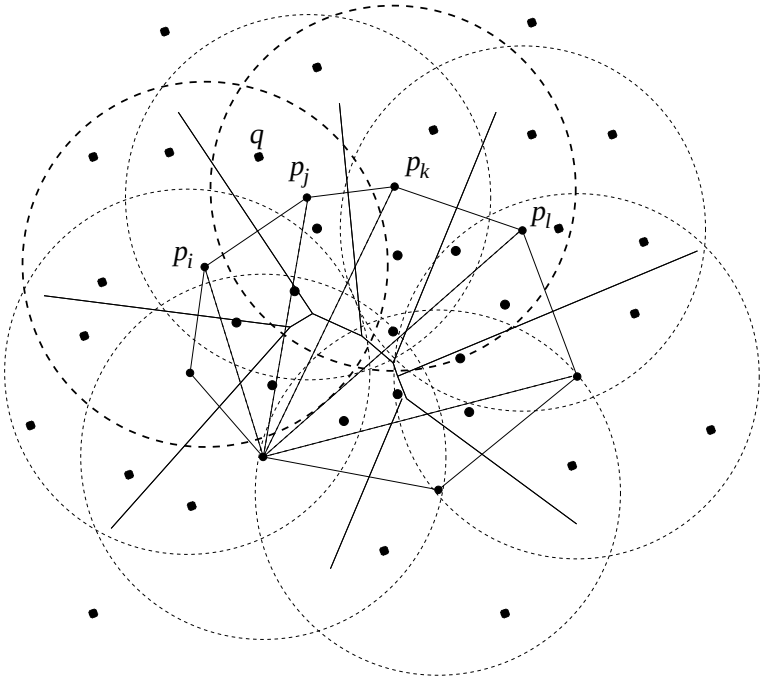

Figure 12: **A key step of Algorithm 5.**

$DT(A)$. We report failure if and only if none of the triangles in $DT(A)$ can be augmented to form a set S, of size n, that satisfies the required property.

9.2.3 Correctness and Analysis

Line 10 of Algorithm 5 performs the key step of computing the set of points in $A \setminus S$ that belong to the interior of at least *two* T-radius circles drawn on the vertices of $CH(S)$. The following lemma shows that this step can be performed quite efficiently.

Lemma 6 *Let S be the set of candidate points at some stage of the Algorithm 5. Let $\mathbb{C} = \{C_T(s) \mid s \in CH(S)\}$ be the set of circles of radius T drawn on the vertices of $CH(S)$. Then the set $\Delta_T(A, S) = \{q \in A \setminus S \mid \exists C_T, C'_T \in \mathbb{C}$ such that $q \in C_T \cap C'_T\}$ can be computed in $O(N \log N)$ time.*

Proof. Refer to Fig. 12. Let $p_i p_j p_k \ldots$ be the points in $CH(S)$. We are interested only in the points *outside* $CH(S)$. Compute $VD(CH(S))$. Now, let $q \in A - S$ be the point under consideration. We shall include q in $\Delta_T(A, S)$ iff q belongs to the interior of at least two circles (dashed circles in the figure). We can locate the voronoi polygon, $VOR(p_j)$, to which q belongs in $O(\log |CH(S)|)$ time [9]. Let p_j be the corresponding site, and p_i, p_k the sites preceding and succeeding p_j in $CH(S)$. Once p_j is located, p_i and p_k can be retrieved in constant time. Observe that if q falls in the interior of two or more circles, then $C_T(p_j)$ and either (or both) of $C_T(p_i)$ and $C_T(p_k)$ *must* belong to that set, since these are the three circles that are nearest to q. This means that once we locate the voronoi polygon for q, the decision regarding inclusion/exclusion of q can be made in constant time. Thus, given $VD(CH(S))$, each point $q \in A \setminus S$ can be processed in $O(\log |CH(S)|)$ time, which takes the worst-case value $O(\log N)$. We need to compute $VD(CH(S))$ only once, which takes $O(N \log N)$ in

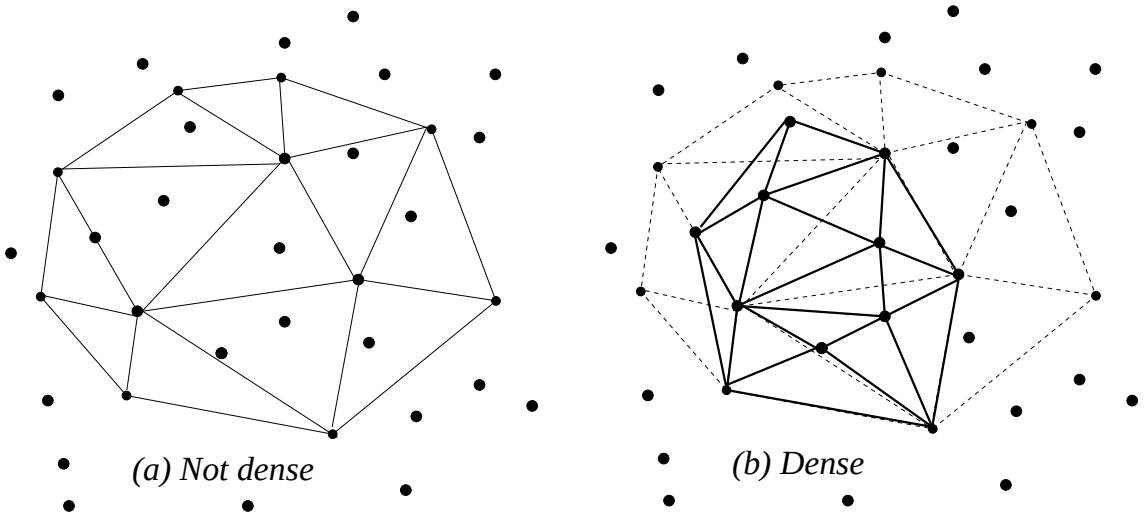

Figure 13: **Correctness of Algorithm 5.** (a) A set S missed by the algorithm. (b) A set S' that the algorithm cannot miss.

the worst case, when $|CH(S)|$ is $O(N)$. Thus all the $|A - S| = O(N)$ points q can be processed in $O(N \log N)$ time. This proves the lemma. □

Theorem 12 *Algorithm 5 correctly computes the required set S in $O(N^3 \log N)$ time.*

Proof. From the manner in which Algorithm 5 augments the set S iteratively, it is easy to see that if it *does* return a set S such that $|S| = n$, then S satisfies the required property.

What we need to prove is that when Algorithm 5 returns **no**, it does not miss a solution S. We call a set $S \subseteq A$ *dense* when there is no point $p \in A \setminus S$ that is contained within $CH(S)$. In other words, a solution S is dense iff all points within $CH(S)$ participate in $DT(S)$. Fig. 13(a) shows a solution S that is *not* dense, while 13(b) (bold edges) shows a dense solution.

Now, suppose, for contradiction, S is some solution that our algorithm missed, $|S| = n$. Let $S^* = S \cup \{s \in A \setminus S$ such that s is inside $CH(S)\}$. Construct $DT(S^*)$. Now, any dense subset $S' \subseteq S^*$ of n points is necessarily a solution (e.g. the one shown in Fig. 13(b)), because no edge in $DT(S')$ can be longer than T. But since S' is a *dense* solution, our algorithm would have found S', because it examines every triangle in $DT(A)$ and systematically augments each triangle to arrive at a solution. This contradicts the hypothesis that our algorithm had answered **no**.

The **for** loop of line 3 runs $O(N)$ times, and in the worst case, the **while** loop of line 7 runs $n - 3 = O(N)$ times. Within the while loop, the costliest operation is the computation of $\Delta_T(A, P)$. By Lemma 6, this takes $O(N \log N)$ time in the worst case. Thus, the overall complexity of Algorithm 5 is $O(N^3 \log N)$. □

10 Hardnesss and Approximation of Maximum-Breach-finite

Problem 10 ***[Maximum-Breach-Finite]*** *Given A, a positive real number T_b and a positive integer n, does a set of points $S \subset A$ exist such that $|S| \leq n$ and for any $i, j \in A$ and* any *path $P(i, j)$ between*

them, breach$(P(i, j)) \leq T_b$*?*

In geometric terms, this problem requires us to find a set of points S such that all points in any arbitrary path in A are within a distance T_b from at least one point in S.

Theorem 13 ***Maximum-Breach-Finite*** *is* **NP-*Hard***.

Proof. We prove this theorem by reducing **Minimum-Geometric-Disc-Cover** [33] to **Maximum-Breach-Finite**.

An instance I_{mgdc} of **Minimum-Geometric-Disc-Cover (MGDC)** is given by $\langle A, T_b, n\rangle$, where the goal is to determine whether the points in A can be covered by at most n discs of radius T_b and the discs are centered on the points of A. The corresponding instance I_{mbf} of **Maximum-Breach-Finite (MBF)** is also $\langle A, T_b, n\rangle$, where the interpretation is as given in the theorem statement.

Suppose answer(I_{mgdc}) = **yes**. Then we have a set $\mathbb{C}$ of at most n discs of radius T_b such that A is covered by $\mathbb{C}$. Then, let $S = \{s_i \mid s_i$ is the center of the ith disc in $\mathbb{C}$, $1 \leq i \leq |\mathbb{C}|\}$. By hypothesis, for all points $p \in A$, there is an $s \in S$ such that $d(s, p) \leq T_b$. This is sufficient to ensure that $breach(P(i, j)) \leq T_b$ for any points $i, j \in A$ and any path $P(i, f)$ connecting them. Thus answer(I_{mbf}) = **yes**.

Similarly, it can be proved that if answer(I_{mbf}) = **yes**, i.e. there exists a set S, $|S| \leq n$ that satisfies the maximum breach criterion, then $|S|$ disks of radius T_b centered on the points in S will cover A. Suppose we take any point $p \in A$, then we can define a path F s.t $F(x) = p, \forall x \in [0, 1]$. Since F is a path and breach of any path is less than T_b, hence there exist a point in $q \in S$ s.t $distance(q, p) \leq T_b$. Hence all the points in A lies within T_b distance of some point in S. Hence discs centered on the points in S with radius T_b cover A. □

Since we have already shown that maximum breach is finite, now we will give an $(O(1) + \epsilon)$ approximation algorithm. The approximation algorithm uses the following results mentioned below as subroutine as a subroutine.

Result 1 *Given a disc of radius r it can be covered completely by* 16 *discs of radius* $\frac{r}{2}$.

We have earlier mentioned one version of the Geometric Packing theorem which is know to be NP-Hard. Now we state another version of the Packing theorem which has also been shown to be NP-Hard.

Problem 11 *Given m points in d-dimensional Euclidean space where* $d \geq 1$*, can be cover them with n discs of radius r.*

For the case $d = 1$, one can compute an optimal solution in linear time with the following algorithm: We always place the next interval (i.e 1-dimensional disc) with its left end at the leftmost point that is not yet covered.

Result 2 (Hochbaum et al. [33]) *Let* $d > 1$ *be some finite dimension. Then there is a polynomial time approximation scheme* H^d *such that for every given natural number* $l \geq 1$*, the algorithm* H^d_l *delivers a cover of n given points in a d-dimensional Euclidean space by d-dimensional balls of a given diameter D in* $O(l^d(l\sqrt{d})^d(2n)^{d(l\sqrt{d})^d+1}$ *with performance ratio* $\leq (1 + \frac{1}{l})^d$.

Theorem 14 *We give a* $O(1) + \epsilon$ *algorithm for* ***Maximum-Breach-Finite*** *problem for a point set A of size m and breach value* T_b.

Proof. Given m points on 2-dimensional Euclidean plane and suppose that the optimal solution of the Maximum-Breach-Finite problem be of size S. Then we can cover point set A with S discs of radius T_b. So the optimal solution for Geometric Covering problem where centre of the discs are not fixed on the set A is $\leq S$. We have seen 2 that Geometric Covering problem has a $(1+\epsilon/16)$ approximation algorithm, so we can find a covering in polynomial time a $(1+\epsilon/16)S$ cover of the point set A with discs of diameter $2T_b$. We cover each circles obtained in above step with with 16 discs of diameter T_b. Now we have a covering of A of size $(16+\epsilon)S$ with discs of diameter T_b. Take one point from each discs of diameter T_b, and we will get a point set of size at most $(16+\epsilon)S$ with the required property. □

11 All-Pairs Maximal Breach: A Sensor Insertion-Based Minimisation Strategy

11.1 An Auxilliary Problem: k-Insert

In the context of **[P2-AVG]**, a solution to the following problem is useful.

Problem 12 ***[k-Insert]*** *A set of N sensors are deployed over a unit-square field A. Given an integer $k > 0$ (which we can assume to be smaller than N), how do we add k sensors to S so as to achieve the best possible improvement in $avgBreach_{max}$?*

How does a solution to **[k-Insert]** help in solving **[P2-AVG]**? Suppose we have a limited number, say 100, sensors at our disposal. We can pick 25 of them, deploy them over A at random, and then insert 5 sensors at a time, intelligently, to use up the remaining 75. A solution **[k-Insert]** thus leads to a good local search strategy for **[P2-AVG]**.

11.2 A Procedure for Minimising Average Breach

Theorem 7 provides the answer to **[k-Insert]** too. To achieve the maximum possible improvement in average breach, we must disturb as many critical edges as we can. Addition of a single sensor in the region $A^{ins}(i, f, S)$ guarantees a reduction in $breach_{max}(i, f, S)$. The natural thing to do would be to select the k *heaviest* critical edges, and add a sensor apiece to the critical regions defined by these edges. The following algorithm describes the procedure.

Input: A set of sensors S, $|S| = N$, and a positive integer k.
Output: A set of sensors (points) S_Δ, $|S_\Delta| = k$, such that $avgBreach_{max}(S \cup S_\Delta) - avgBreach_{max}(S)$ is maximised.
Method: See Algorithm 6.

Algorithm 6 MinimiseAvgBreach.

1: Using Algorithm 4, compute the $|V_{VD}| - 1$ critical edges of $\langle A, S \rangle$;
2: Select the k heaviest critical edges $e^l_{cr} = (u_l, v_l)$, $1 \leq l \leq k$.
3: $S_\Delta \leftarrow \{u_l \mid 1 \leq l \leq k\}$
4: **return** S_Δ.

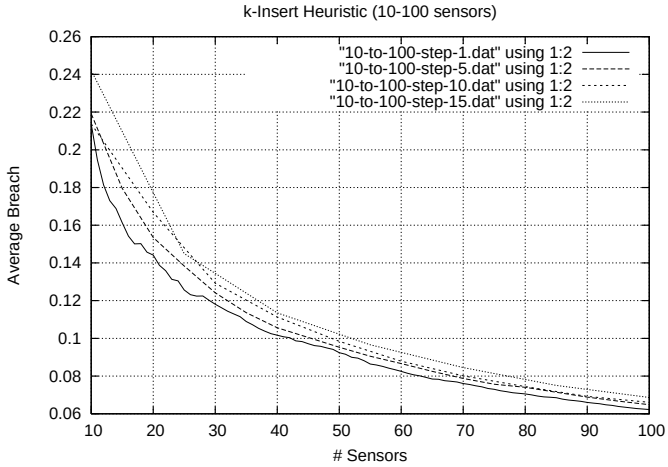

Figure 14: ***MinimiseAvgBreach* Performance.** 10 Sensors to 100 Sensors

Theorem 15 *Procedure **MinimiseAvgBreach** achieves its purpose in $O(N^2)$ time.*

Proof. **MinimiseAvgBreach** maximises the improvement in $avgBreach_{max}$. This follows from the fact that S_Δ is composed of voronoi vertices forming one endpoint of each of the k heaviest critical edges. Each such voronoi vertex belongs to the critical region for sensor insertion (Theorem 7).

Step 1 of the algorithm takes $O(N^2)$. Step 2 takes $O(k) = O(N)$, because $k < N$. Hence, the overall time-complexity is $O(N^2)$. □

11.3 Experimental Data

We carried out simulations to study the effectiveness of Algorithm 6 in reducing average maximal breach. We started with random arrangements of 10 sensors, and then repeatedly applied **MinimiseAvgBreach**, with step-size $k = 1$, 5, 10 and 15 respectively, to arrive at a configuration with 100 sensors. Fig. 14 shows how the value of average maximal breach reduced after each call to **MinimiseAvgBreach**. Fig. 15 shows the reduction in average breach when we started with 50 sensors, and reached 200, with step-sizes $k = 5$, 10, 25 and 50 respectively.

We see in Fig. 14 and 15 a validation of our intuitive strategy **MinimiseAvgBreach**. Another point can be noted. The final values of breach for $k = 1$ and $k = 15$ do not differ by a big margin. But the actual execution time (as noted during our simulations, but not reported here) is a lot longer when $k = 1$. The convergence rate is far more rapid for $k = 15$; though asymptotically, the run-time is the same for any $k \leq N$.

References

[1] S. Megerian, F. Koushanfar, M. Potkonjak, M.B. Srivastava; *Worst and Best-Case Coverage in Sensor Networks*; IEEE Transactions on Mobile Computing, vol.4, no.1, pp. 84- 92, Jan.-Feb.

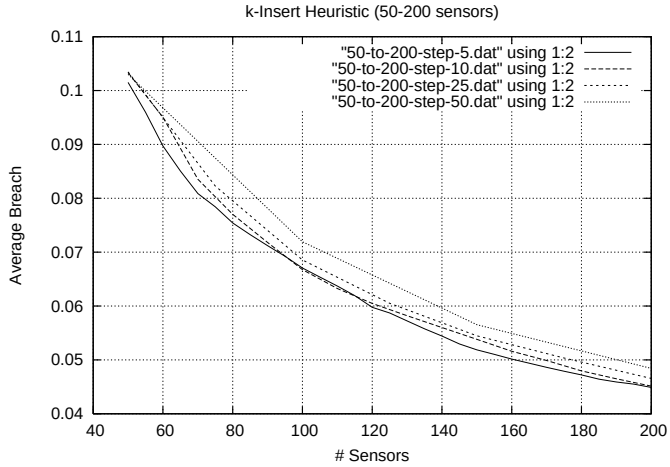

Figure 15: ***MinimiseAvgBreach* Performance.** 50 Sensors to 200 Sensors

2005. [5]

[2] S. Megerian, F. Koushanfar, G. Qu, M. Potkonjak; *Exposure in Wireless Sensor Networks*; in Proceedings of the Seventh International Conference on Mobile Computing and Networking (MobiCom '01); ACM Press; July, 2001.

[3] G. Veltri, Q. Huang, G. Qu, M. Potkonjak; *Minimal and maximal exposure path algorithms for wireless embedded sensor networks*; in Proceedings of the First International Conference on Embedded Networked Sensor Systems; ACM Press; pp. 40 - 50; 2003.

[4] Xiang-Yang Li, Peng-Jun Wan, Frieder, O.; *Coverage in Wireless Ad-hoc oc Sensor Networks*; IEEE Transactions on Computers, vol.52, no.6, pp. 53- 763, June 2003.

[5] Chi-Fu Huang, Yu-Chee Tseng; *The Coverage Problem in Wireless Sensor Networks*; in Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications; ACM Press; September, 2003.

[6] S. Meguerdichian, F. Koushanfar, M. Potkonjak, M.B. Srivastava; *Coverage Problems in Wireless Ad-hoc Sensor Networks*; in Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2001; IEEE CS Press; April, 2001.

[7] Santosh Kumar, Ten H. Lai, Jzsef Balogh; *On k-coverage in a Mostly Sleeping Sensor Network*; in Proceedings of the 10th annual international conference on Mobile computing and networking, MobiCom 2004; ACM Press; September 2004.

[8] Santosh Kumar, Ten H. Lai, Anish Arora; *Barrier Coverage with Wireless Sensors*; in Proceedings of the 11th annual international conference on Mobile computing and networking, MobiCom 2005; ACM Press; August, 2005.

[5] Actually, there are two versions of this paper - a 2001 conference paper [6] and a 2005 enhanced journal paper [1]. [4] refers to [6].

[9] M. de Berg, M. Van Kreveld, M. Overmars, O. Schwarzkopf; *Computational Geometry Algorithms and Applications*; Springer-Verlag, 1997.

[10] F. Aurenhammer; *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*; ACM Computing Surveys, Vol. 23, No. 3, September 1991.

[11] K. Mehlhorn, S. Näher; *The* LEDA *Platform of Combinatorial and Geometric Computing*; Cambridge University Press, 1999.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein; *Introduction to Algorithms, 2nd Edition*; MIT Press, 2001.

[13] H. Melissen; *Packing and Covering with Circles*; Ph.D. thesis, University of Utrecht, 1997.

[14] R. Fowler, M. Paterson, S. Tanimoto; *Optimal Packing and Covering in the Plane are NP-Complete*; Information Processing Letters, Vol. 12, No. 3, pp. 133-137; 1981.

[15] D. S. Hochbaum and W. Maass; *Approximation schemes for covering and packing problems in image processing and VLSI*; Journal of the ACM, Vol. 32, pp. 130-136; 1985.

[16] C.H. Papadimitriou, K. Steiglitz; *Combinatorial Optimization: Algorithms and Complexity*; Chapter 12; Prentice Hall; 2001.

[17] C.H. Papadimitriou; *Computational Complexity*; Chapter 9; Addison-Wesley; 1993.

[18] M.R. Garey, D.S. Johnson; *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W.H. Freeman; 1979.

[19] G.T. Toussaint; *The Relative Neighborhood Graph of a Finite Planar Set*; Pattern Recognition, Vol. 12, No. 4, pp. 261-268; 1980.

[20] K.R. Gabriel and R.R. Sokal; *A New Statistical Approach to Geographic Variation Analysis*; Systematic Zoology, Vol. 18, pp. 259-278; 1969.

[21] F. Zhao, L.J Guibas; *Wireless Sensor Networks: An Information Processing Approach*; Morgan Kaufmann; 2004.

[22] M. Weiser; *The Computer for the Twenty-First Century*; Scientific American, pp. 94-100; September 1991.

[23] M. Cagalj, J-P Hubaux, C. Enz; *Minumum-Energy Broadcast in All-Wireless Networks: NP-Completeness and Distribution Issues*; in Proceedings of the 8^{th} Annual International Conference on Mobile Computing and Networking (MobiCom 2002), ACM Press; September, 2002.

[24] C. Gui, Prasant Mohapatra; *Power Conservation and Quality of Surveillance in Target Tracking Sensor Networks*; in Proceedings of the 10^{th} Annual International Conference on Mobile Computing and Networking (MobiCom 2004), ACM Press; September, 2004.

[25] J. Urrutia; *Routing with guaranteed delivery in geometric and wireless networks*; Handbook of Wireless Networks and Mobile Computing, pp. 393-406; John Wiley & Sons; 2002.

[26] G.J. Pottie, W.J Kaiser; *Wireless Integrated Network Sensors*; Communications of the ACM, Vol. 43, pp. 51-58; May, 2000.

[27] G. Borriello, R. Want; *Embedded Computation Meets the World Wide Web*; Communications of the ACM, Vol. 43, May 2000.

[28] W. Ye, J. Heidemann, D. Estrin; *An Energy Efficient MAC Protocol for Wireless Sensor Networks*; in Proceedings of the 21^{st} Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2002), Vol, 3, pp. 3-12; IEEE CS Press; June, 2002.

[29] A. Savvides, C.C. Han, M.B. Srivastava; *Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors*; in Proceedings of the 7^{th} Annual International Conference on Mobile Computing and Networking (MobiCom 2001), ACM Press; July, 2001.

[30] E. Kranakis, D. Krizanc, J. Urrutia; *Coverage and Connectivity in Networks with Directional Sensors*; in Proceedings Euro-Par Conference, Vol. 3149, Springer Verlag, LNCS; 2004.

[31] P.J. Slater; *R-Domination in Graphs*; Journal of the ACM, Vol. 23, pp. 446-450; 1976.

[32] Abhay Kumar; *Coverage in a Wireless Sensor Network for Detection of an Intruder Moving in a Straight Line*; M.Tech. (Computer Science) Thesis, Indian Statistical Institute; 2006.

[33] D. Hochbaum, W. Mass; *Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI*; Journal of ACM, Vol. 32 (Issue 1), pp. 130-136; 1985.