Chapter 7

Topology control in sensor, actuator and mobile robot networks

Arnaud Casteigts, Amiya Nayak and Ivan Stojmenović

School of Information Technology and Engineering, University of Ottawa,
800 King Edward, Ottawa, Ontario K1N 6N5, Canada.

Abstract

The efficiency of many sensor network algorithms depends on characteristics of the underlying connectivity, such as the length and density of links. It is therefore a common practice to control the number and nature of links that are to be used among all potentially available links. Such *topology control* can be achieved by modifying the transmission radii, selecting a given subset of the links, or moving some nodes (if such functionality is available). This chapter reviews some of these problems, and related solutions applicable to the context of sensor and actuator networks. Spanning structures and minimum weight connectivity are first discussed, and some applications for power efficient and delay bounded data aggregation are described. The issue of detecting critical nodes and links to build a bi-connected topology is also investigated, with the aim of providing fault-tolerance to the applications, and some recent and prospective works considering bi-connectivity of mobile sensors/actuators and related deployment of sensors, augmentation, area and point coverage are discussed.

7.1 Introduction

Unless indicated otherwise, all the solutions presented in this chapter assume that every node is able to know its own position and the positions of its neighbors using exchanges of `HELLO` messages. These solutions also assume that no obstacles are present and that the network can thus be represented by a *Unit Disc Graph* (UDG for short), that is, a graph where two vertices are joined by an edge (and said *neighbors*) if and only if the distance between the two corresponding nodes is under a given threshold R. The chosen value for R corresponds to the transmission radius, which is generally assumed to be the same for all nodes. Some variations of this model can however be considered to represent obstacles or different transmission radii among nodes, in which case the edges are considered either as directed (*arcs*), or used only if both communication directions are available. The *min-power* graph is an example of such non-symmetrical UDG that represents the cases where each node has the smallest possible transmission radius with respect to the network connectivity. Depending on the scenarios considered, the transmission power can be decided once at the starting time, or adjusted for each message.

The chapter starts with a discussion on the main general approaches used to control the connectivity in static sensor networks. The emphasis is put in particular on the problem of finding minimum transmission radii so that the network is connected (*i.e.*, there exists a path between any two nodes in the network). This problem is actually closely related to the problem of finding a *minimum spanning tree*, since the longest edge of such structure corresponds to the minimal common radius achieving the connectivity. We thus discuss the problem of approximating this structure in a distributed and localized manner. A few applications of the minimum spanning tree in the context of data aggregation are then presented, with a couple of example protocols that exploit the localized approximation discussed before. Finally, we discuss the problem of maintaining a distributed spanning forest over a uncontrolled sensor and actuator topology so that each sensor belongs to the same tree as at least one actuator.

The second part of the chapter (starting at section 7.6) is concerned with problems related to bi-connectivity. The question of detecting local critical nodes and links in a distributed fashion is first covered. We then discuss several scenarios involving bi-connectivity of mobile robots (which may correspond to sensors or actuators, depending on the cases). The first two scenarios address the problem of deploying bi-connected sensors around a given point of interest, whereas the two last scenarios address the problem of bi-connecting a network that is initially 1-connected.

7.2 General approaches in static sensor networks

When the nodes are static, controlling the topology amounts to selecting only a subset of the possible links for effective use, according to some desired criteria (distance between nodes, remaining level of energy, avoidance or favoring of cycles, *etc.*). There are essentially three ways of proceeding:

1. Selecting particular neighbors based on other criteria than the sole distance between them. We refer to [BLRS06] as an example of such selection, where the criterion is to minimize the number of symmetric links so that interferences are bounded. The topology is designed by maintaining the number of neighbors of each node below a value k while guaranteeing the overall connectivity with high probability. The proposed protocol consists in two message exchanges. First, all nodes transmit their ID using the maximal emission power. Upon reception, each node i builds a list L_i containing its k nearest neighbors. Then each node transmits its list using again the maximum power. A link between two nodes is then kept if and only if it belongs to the k closest neighbors of each other. While the value k needed for connectivity with high probability is logarithmic, their experiments showed that $k = 6$ was the 'magic' number above which the network is always connected in practice.

2. Using an adjusted transmission range, possibly being different for each node, and then using all neighbors within that range. While being very energy efficient, the problem of this approach is that medium access (MAC) protocols become complicated, and a vast majority of the existing ones, which assume symmetric and same transmission powers, does not work well in this context.

3. Using an adjusted transmission range, but with the same value for every node. This option is often preferred over the previous one, and can also serve as a prior step to further selection. The main question here is to find the minimal value that still guarantees the network connectivity.

We now discuss the construction of a minimum spanning tree, that one can see as being closely related to the first and the third approaches.

7.3 The Minimum Spanning Tree

Minimum spanning trees (MST for short) are emblematic example of topological structures commonly used in communication networks. In a graph $G = (V, E)$, an *MST* is a connected subset of the graph that contains all the vertices and minimizes the overall sum of its edge lengths (or more generally their *weight*). A well known centralized algorithm to build such structure is the Kruskal's algorithm, which mainly works as follows. All edges of E are sorted in the increasing weight order, and a new empty set E' is created. For each edge e in the ordered set E, e is added to E' if it does not create a loop in $G' = (V, E')$. The resulting graph G' is a minimum spanning tree of G.

The Figure 7.1 gives an example of *MST* that has been built over a random unit disc graph. As usual in wireless networks, the *length* of the edges was used as their weight. Note that if several edges have the same length, then several equivalent *MSTs* may exist, but this non-determinism can be easily broken based on additional parameters (such as a comparison between nodes IDs).

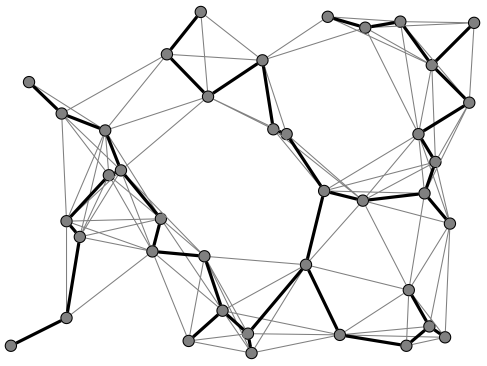


Figure 7.1: A Minimum Spanning Tree on top of a random unit graph

An interesting fact about the minimum spanning tree is that its longest edge also corresponds to the optimal common transmission radius (*i.e.*, the minimal radius such that network connectivity is preserved). This follows directly from Kruskal's algorithm, this edge being the last added edge in E'.

Considering a few related results, Penrose [Pen99] proved that the 'furthest nearest' neighbor of a node in the network, and the longest MST edge, have asymptotically (when n goes to $+\infty$) the same value. The probability of connectedness exhibits a very sharp transition from 0 to 1 just before the critical value [Bet02]. Another interesting fact is that an important amount of energy can be saved if a fewer connectivity is required. Santi and Blough [SB03] showed for example that, in two and three dimensions, halving the critical transmission range of the nodes still keeps 90% of them connected within a same component.

While the problem of building an MST and finding the optimal transmission range are closely related, both of them require to consider global knowledge such as the size and density, or the spacial distribution of nodes. The problem is then to approximate these optima in a distributed fashion. We present below a few approaches for doing so, based on a localized approximation of the minimum spanning tree.

Localized approximation of the Minimum Spanning Tree (LMST)

Localized Minimal Spanning Trees (LMST), introduced in [LHS03] and already discussed in the Chapter 2, are distributed structures that approximate the minimum spanning tree using only local information. More precisely, each node u is assumed to collect position information of its 1-hop neighbors, then to compute the local minimum spanning tree that covers itself and its 1-hop neighbors (including edges between these neighbors, possibly deduced from their positions). Then, an edge (uv) belongs to the *LMST* if and only if is belongs to the local

MST of both u and v. For illustration purpose, we provide on Figure 7.2 the results obtained using this algorithm, as opposed to the optimal solution given on Figure 7.1. As one can see, just a few cycles (3) were created. Considering that only 1-hop information was used, the approximation can be qualified as good.

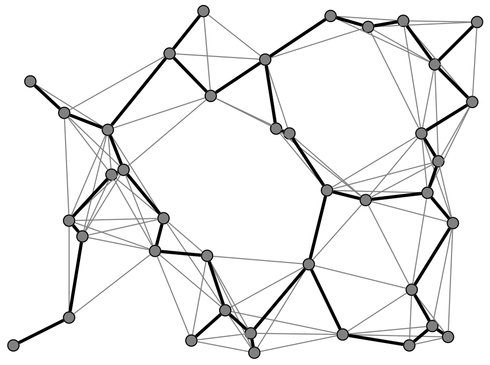


Figure 7.2: Localized Minimum Spanning Tree

Based on the observation of Penrose that the longest edge of the MST corresponds to the optimal transmission range value, Ovalle et al [OSGS05] proposed to use the LMST algorithm to find this value. More precisely, the basic idea was to find the longest LMST edge (that approximate this value) using a wave propagation within. However, it was observed in this paper that even a very small number of additional edges (such as those creating a cycle on Figure 7.2,) representing less than 3% in their experiments (on networks with up to 500 nodes) may extend the range value by about 33%. In turn, these 33% of range value may represent 50% or more increase in energy consumption, depending on the power attenuation factor that is considered. A quasi-localized scheme is thus proposed in the paper to remove additional edges of the *LMST*, using in average less than 7 messages per node. This procedure is a *loop breakage procedure*, which iteratively follows dangling edges from leaves to loops. The loops are then broken by eliminating their longest edge. These procedures continue until they all end up at a same node, considered as a *defacto* leader, that can learn the value of the longest edge in the process, and broadcast it back to the other nodes. This procedure operates only in two dimensions, since it is based on a face routing scheme (see Chapter 4 for details on face routing).

7.4 Data aggregation

A common scenario of sensor networks involves the deployment of hundreds or thousands of low-cost, low-power sensor nodes in a region from where information will be collected periodically. Sensor nodes must sense their nearby environment and send the information to a sink or actuator, from which the collected information can be further processed or made available to the user. In the most basic reporting schemes, each sensor independently sends its data to the associated actuator using routing operations that generates a lot of redundant traffic if the data is geographically or temporally correlated (*e.g.* if two neighboring, or successive, measure values are expected to be close to one another). *Data aggregation* arise from the observation that most of this redundancy could be avoided if the data were partially processed locally to the sensors, for example by averaging it over time or space before forwarding it. This is what *data aggregation* does, by applying fusion/consolidation functions to the data along its way to the sink. Example of such functions include *average, maximum, minimum, sum, count,* or *deviation*, that can be applied either periodically or on-demand.

Once these functions are chosen and combined, the main problem of data aggregation is to build the overlay structure along which data will be effectively aggregated. This structure must be as efficient as possible to allow a fast aggregation while maximizing the lifetime of the network (*i.e.,* the number of aggregation cycles, or *rounds*, before energy depletion).

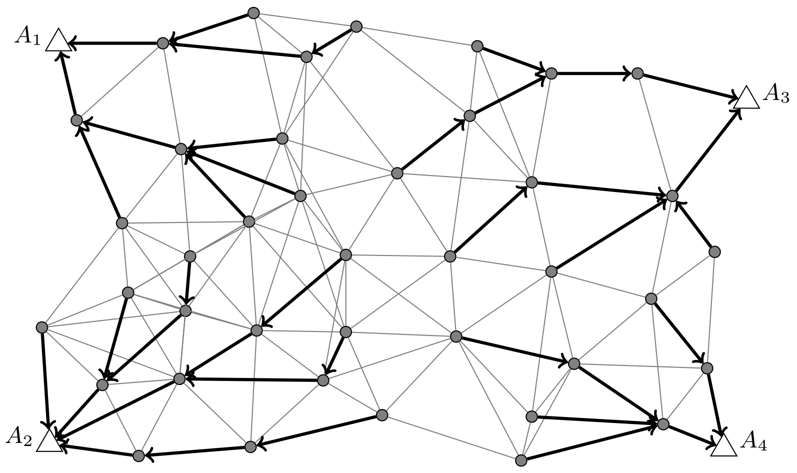


Figure 7.3: Data aggregation trees based on the cumulative distance to the sink.

A very common type of structure in this context is the tree, which represents a natural hierarchical organization where parent nodes collect and aggregate the data coming from their children, before forwarding their own data in turn. At the top of the tree is the sink (or actuator). Figure 7.3 gives an example scenario where sensors are reporting toward four actuators. Here the trees are set up in such a way that the sum of edge lengths between sensors and actuators are minimized. Other criteria such as minimizing the number of *hops* or the favoring the nodes with more remaining energy could be used instead.

In order to be realistic and efficient in the context of sensor and actuator networks, an algorithm to build such structure should have some important properties. First, the algorithm should be distributed since it is extremely energy consuming to calculate the optimum paths in a dynamic network and inform others about the computed paths in a centralized manner. The algorithm must scale well with increasing number of nodes. Another desirable property is robustness, which means that the routing scheme should be resilient to node and link failures. The scheme should also support new node additions to the network, since not all nodes fail at the same time, and some nodes may need to be replaced. In other words, the routing scheme should be self-healing. The final and possibly the most important requirement for a data aggregation scheme for wireless sensor networks is being energy efficient.

One common approach to build an aggregation tree is to flood a packet from the sink (or actuator) so that every node can select a parent among the nodes from which it received the packet. The data is then aggregated along this so-constructed hierarchy. The problem with flooding techniques (some of which are detailed in Chapter 2) is that this may generate a lot of redundant messages, be problematic with respect to possible interferences, are more generally cost an important amount of energy, since several retransmissions happen uselessly. In [TKS07], the idea was proposed to build and maintain an underlying *LMST* that is considered as the topology during the flooding operation, so that the overhead of messages is much reduced (the same principle may also work with other structures such as the *Relative Neighborhood Graph*, as pointed out by the authors).

A set of three protocols, called *Localized Power Efficient Data Aggregation Protocols* (L-PEDAP), are proposed. Their variations correspond to different strategies of parent selection. According to the decisions made during this flooding process, the tree is yielded. The three methods are: 1) choosing the first node from which the flooded packet is received, 2) choosing the node that minimizes the number of hops to the sink, and 3) choose the node that minimizes the total energy consumed over the path to the sink. Note that the first and second methods are nearly equivalent in sensor networks, since the processing time of messages overcomes their physical transmission time over the air, making the transmission time quasi-proportional to the number of hops.

Since the underlying structure (here, the *LMST*) can be locally maintained, this solution accommodates new node arrivals and departures of existing nodes. The authors also propose power-aware versions of these protocols that consider the energy level of nodes while constructing the underlying structure. Finally, the paper derives a theoretical upper-bound for the lifetime in terms of the first node failure, and simulation results show that the protocols achieve up to 90% of this upper bound.

Delay-bounded and power-efficient data aggregation. While the purpose of the LMST is to build an energy-efficient structure by selecting the shortest edges, it is not optimal with respect to delay considerations. In [XLNS09], the authors propose a data aggregation protocol where a LMST is first considered, then modified at reporting time in order to match additional delay constraints. As with most existing solutions for delay-bounded data aggregation, the metric used here to approximate the delay of a communication is the number of hops. Indeed, among the delays experienced by a packet at each hop, the distance travelled over the air is negligible compared to the time required to process it at the nodes, which can be considered equal for each node when all packets are of the same size. The delay experienced by a packet is thus directly proportional to the number of hops it travels. The proposed algorithm, called *Desired Hop Progress* (DHP), allows to respect given delay bounds while using significantly less energy than the known competitor in [MPGA05] (from 25 to 75% less energy consumption and up to 123% network lifetime depending on the configurations). We detail it now.

As with the previous algorithm (L-PEDAP), a tree is built on LMST edges when an actuator (they can be several) floods the network with a request. Thanks to a specific retransmission mechanism, every node computes and memorizes several parameters during this process, the main of which are the hop distance to the sink. Two kind of distances are actually memorized: the *LMST-based* distance (or LD), which represents the number of hops to the actuator using only the LMST edges, and the *UDG-based* distance (or UD), which is the number of hops to the actuator if any edge could be used. These different distances are illustrated on Figure 7.4, where thick edges represent the LMST-based tree, and all edges together the UDG. The aggregation process then starts from the leaves to the actuator as explained below.

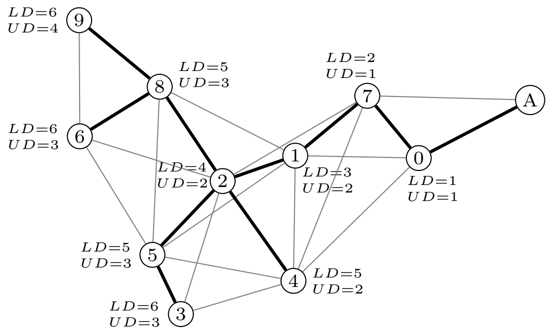


Figure 7.4: Example topology for the DHP protocol.

Every request (packet flooded) contains information about the delay that must be respected during this aggregation. If this delay is achievable using only LMST edges (that is, if for every node, LD is equal to, or lower than the desired delay), then the aggregation occurs as with L-PEDAP using only these edges. Otherwise, the reporting nodes tries to find *shortcuts* outside of the LMST, using the UDG distance. More precisely, before sending the aggregated packet to its parent, every node calculates a particular ratio to decide whether using

LMST edges is sufficient or a shortcut is necessary. This ratio, called *dhp* (for *desired hop progress*), is defined as:

$$dhp = \lceil \frac{LD}{Delay - MED} \rceil,$$

where MED (*most experienced delay*) is the number of hops already done by the childs. The meaning of this ratio is actually to represent the LD distance that should be gained at every next hop so that the delay is finally matched. If this value is higher than 1, then it determines which level of shortcut should be ideally used. Considering the scenario given on Figure 7.4 with a delay limit of 3, this leads to the following execution.

Since node 8 has an UD equal to the required delay, it cannot accept any child (because the delay could not be respected whatever the shortcuts). Node 9 will therefore not be considered for the aggregation. Since nodes 3 and 6 have both a *dhp* value of 2 ($\lceil \frac{6}{3-0} \rceil$), they try to apply a shortcut to gain 2 LMST hops, and thus select node 2 as parent. Nodes 4, 5 and 8 compute their *dhp* (also equal to 2), and thus select node 1 (shortcutting node 2). Node 2 then computes *dhp* (also equal to 2), and takes node 7 as parent. For the same reasons again, node 1 decides that node 0 is its parent. In turn, node 7 finds that its report has to be sent directly to the actuator, as a single last hop is allowed ($Delay - MED = 1$). Finally, node 0 naturally determines the actuator to be its parent. While almost no LMST edges were used in this example, the point of this protocol is that such edges are always used when the delay constraint allows it, which tends to offer both delay-bounded and power-efficient aggregation at the same time.

Two variations of the DHP protocol were also proposed in the same paper (DHPA and DHPAC), to integrate it with *sensor activity scheduling* and *connected dominating set* (CDS), respectively. Detailed discussion on CDS can be found in Chapter 1. The two variants require fewer sensors to report and thus have reduced bandwidth usage and improved energy efficiency. The first variant, DHPA, adopts a localized area coverage algorithm [GCSRS06] for selecting an active node set. Active nodes monitor the environment and generate reports, whereas the others switch to sleep mode for energy saving. The DHP protocol is therefore run only on active nodes. In this area coverage algorithm, each node sets a timeout t to start coverage evaluation and schedule its activity. Considering that nodes with shorter t will have a higher chance to stay active, they define $t = c/E_{rest}$ (c is a constant, and E_{rest} is remaining nodal energy). This definition favors energetic nodes. That is, the more residual energy a node has the more chance the node gets to work. The second variant, DHPAC, is a combination of DHPA and the localized CDS algorithm from [CSR04]. In this combination, the CDS algorithm is run on active nodes determined by the area coverage algorithm. Since each active node either belongs to the CDS or has a direct neighbor in it, non-CDS nodes will report to their closest CDS neighbor, while CDS nodes will run the DHP protocol to organize data aggregation within the CDS.

7.5 Spanning trees in uncontrolled dynamic topologies

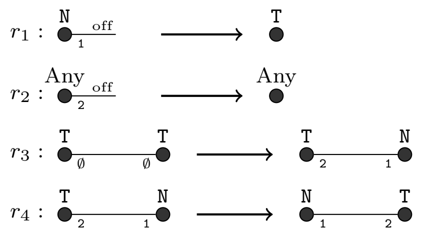
The discussion below addresses the scenarios where sensors are to move in an uncontrolled fashion, which is for example the case when they are carried by some physical actors of the considered scene (*e.g.* animals, vehicles, virtual insects or robots whose movement are to be determined by external parameters). The problem of maintaining distributed spanning trees over dynamic topologies has been extensively studied these past few decades, especially in the two research areas of dynamic graphs and mobile ad hoc networks. To the best of our knowledge, a vast majority of approaches, if not all of them, considered the problem of building a *single* spanning tree to cover the whole network. Generally tackled from the angle of *self-stabilization*, these approaches consider topological events as *faults* that induce a *non-legal state* that the algorithm must correct. The correction is then achieved when the whole network is covered by a single tree, which is the *legal state.* To give a few references on this family of approaches, one can cite the distributed graph algorithm in [AKM$^+$93], which exhibits the shortest construction time 'from scratch' (recently adapted to the message passing model in [BK07]), and [Gae03] that describes a number of comparable approaches.

While most proposed algorithms engage a complete re-construction of the tree after each topological failure, some other more realistic approaches (*e.g.* [BFG$^+$03], [AMZ06]) attempt to correct only the local discrepancies resulting from the link failures. However, these algorithms still require some stabilization time during which the separate subtrees are unavailable and inconsistent. An important consequence is that they simply cannot deal with topologies that change quicker than the stabilization time.

Up to now we discussed scenarios where the connectivity of the whole network was required. This might not always be the case, however. Most applications for sensor and actuator networks do not actually require that a path exists between a sensor and all of the actuators, the point being that sensor must only be able to report information to, or communicate with, at least *one* actuator (or perhaps a few, for fault tolerance). On the other hand, is it of utmost importance that such communication are *always* achievable, *i.e.,* that the underlying supporting structure is always available.

A novel approach addressing highly dynamical topologies was proposed in a recent paper [CCGP09]. The basic idea behind this apprach is to renounce to build a single tree covering the whole network, but instead consider maintaining a forest of several trees that grow opportunistically and that recover a consistent state in one single operation after any topological failure in such a way that the two parts of a given broken tree remain transparently usable. The key point to achieve such property is that both *mergings* and *splittings* of the trees are *purely* localized events that do not generate any wave propagation. The algorithm relies on the circulation of tokens whose number is strictly maintained at one per tree. The difference with other token-based approaches is that the walk of each token is limited to the edges of its tree, which offers some very specific properties. The main of these properties is that every node knows anytime which one of its local edges leads to the token, this edge being simply the one by which the token went out after its previous visit. The consequence is that when an edge of a tree is broken, one of the two endpoint nodes knows that its remaining part of the tree is token free, and that it is now the 'highest' node on the route that led to the lost token. As a consequence, it can locally regenerate a new token and resume the circulation transparently for the other nodes.

The principle of this algorithm can be detailed by a small set of localized modification patterns, depicted on Figure 7.5. Initially, every vertex is a one-vertex tree that has its own token (label `T`). When two tokens are located on neighboring vertices, they are merged and the corresponding edge is marked as a *tree edge* (rule r_3). This marking use a different value on each side (`1` and `2`) to reflect the orientation induced by the remaining token. If no merging is locally achievable, the token is transmitted to any neighbor <u>in</u> the tree (rule r_4), and the orientation mark is updated consequently. For any given node, if the local edge leading to the token is broken, then a new token is regenerated locally (rule r_1). The other side of the broken edge does not perform any particular operation (rule r_2).



<u>initial states:</u>

- `T` for every vertex,
- ∅ for every edge extremity.

<u>meaning of the states:</u>

- `T`: a token is on this node,
- `N`: no token is on this node,
- `1`: this tree edge leads to the token.
- `2`: this tree edge does not.

Figure 7.5: The spanning forest algorithm.

An interesting question for the context of sensor and actuator networks is whether the expected size of the trees is large enough to guarantee that each sensor has at least one actuator in its tree at anytime. In other words, one might want to answer the following question: "*given a number of sensors, their density and the expected rate of topological changes, how many actuators are needed so that the probability to have anytime at least one in each tree is above a given threshold?*". In [CCGP09] the authors provided a first element of answer by characterizing the expected merging time of two given trees, as a function of their size and the number of links available between them. The road between these first results and the complete answer may still be important, though.

7.6 Detection of critical nodes and links

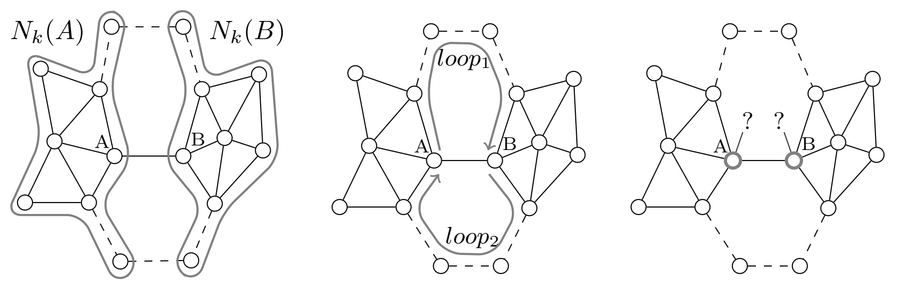
In sensor and actuator networks, the failure of some nodes or links, if generating several partitions of the network, may be fatal for collecting data from the field or performing certain actions on sensors. It is expected however that the network exhibits some *critical connectivity* before partitioning. Recognizing such properties in a timely fashion could allow to perform some data or service replication, so that the network can continue to function after the partition occurred. This kind of detection may also be used at deployment time (*e.g.* while deciding a common communication radius) to ensure that no such critical node or link exist, that is, that the network is *bi-connected*. Both approaches may be considered to add fault tolerance to the network. We discuss below some ideas for detecting critical links and nodes.

Algorithms for detecting critical nodes and links based on global knowledge are well known. However, their use in sensor networks is limited since this requires the entire topology to be known by a single entity, which is not scalable and implies a delay between topological changes and the system reactions. It appears therefore preferable to try to detect critical links and/or nodes in a local and distributed fashion, even if making possibly a few appreciation mistakes that implies a more 'pessimistic view' of the connectivity (an element seen as critical while not being so).

In a global context, a node, or link, is said critical if its removal disconnects the network, that is, if this partitions it into several connected components. The definitions of what critical nodes and links are must be slightly modified in a localized context. As introduced in [JSHS04], a node can be said *locally critical* if its removal disconnects the subgraph of its p-hop neighbors. In the case of links, several definitions can be considered, and three were proposed in the same paper, based on the method used to look at the local connectivity. These methods are illustrated on Figure 7.6. The first method consists in looking at the p-hop neighbors of both endpoints and see if some are in common (Fig. 7.6(a)). The second one is to initiate a face traversal on both sides of the tested link in order to see if the other endpoint is reached before $2p$-hops (Fig. 7.6(b)). Finally, the link can also be decided critical if its endpoints are critical themselves, and this information is already available (Fig. 7.6(c)).

Depending on whether position information is available to the nodes, a variation of each definition can be considered, as discussed in [JSHS04]. Let us consider the simple example given on Figure 7.7, where nodes A and B must determine if their common link is critical, based on a 1-hop neighboring information. If the respective positions of C and D are known, then the fact that a link exists or not between them can be established without additional information, while this is not possible using only topological information. Obviously, such position-based deduction assumes UDG model without obstacle. These two variations are denoted as k-*top*- and k-*pos*-criticality by the authors.

Experiments using random UDGs showed a high correlation between global and local decisions. The only difference is when alternative routes exist but are relatively long. The point is that real critical elements will be detected as such,



(a) $N_k(A) \cap N_k(B) = \emptyset$ (b) $|loop_1| > 2k \wedge |loop_2| > 2k$ (c) $critical(A) \wedge critical(B)$

Figure 7.6: Localized detection of a critical link (here (AB))). For each method, the link is decided critical if the caption formula is true.

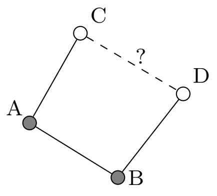


Figure 7.7: Impact of the availability of position information on the detection of criticality.

and any wrong appreciation is only generated by excess of caution (in case of reactive replications) or a slight excess of connectivity (in case of link selection).

These notions can be generalized to the case of critical k-connectivity of the network, as proposed in [JGK$^+$07]. In the first protocol of this paper, each nodes makes criticality decision by verifying whether or not each of its p-hop neighbors has degree (number of neighbors) at least k. The second protocol tests also whether the subgraph of p-hop neighbours of a given nodes is k-connected. The third protocol verifies also whether this subgraph contains any critical nodes.

7.7 Bi-Connected robot team movement for sensor deployment

We consider here the problem of deploying static sensors around a *point of interest* (POI) using a fleet of mobile robots able to carry them. We describe the solution proposed in [Li09]. Here, the number of mobile robots is considered arbitrary (and limited), and each one is initially supplied with an arbitrary number of sensors. The problem is then to deploy collaboratively the sensors so that their topology forms a *triangle tesselation* around the POI. The choice of a triangle tesselation is motivated by its interesting geometrical properties that create a near-optimal coverage by the sensors while making them bi-connected as a by-product. The main concern is to ensure that the robot network remains also bi-connected during the deployment, while minimizing the sum of their moves.

The principal steps of the proposed protocol are as follows. At the beginning, the robots are randomly scattered in the region. They first run an auxiliary protocol such as *Greedy-Rotation-Greedy* (see [LFSS07] or Chapter 10) to gather around the POI into several concentric hexagonal layers that form a triangle tesselation at the local scale. Let us first assume that the number of robots is such that the outmost layer is complete (as with the 7-robots hexagon depicted in the middle of Figure 7.7). Each robot starts by dropping one sensor at its position, then the whole group of robots shifts in one direction, and starts a circular (or more precisely, a hexagonal) course around the already deployed sensors, dropping new sensors along their way. This process repeats until all sensors have been deployed. Note that each circumvolution can deploy several layers, depending on the group diameter.

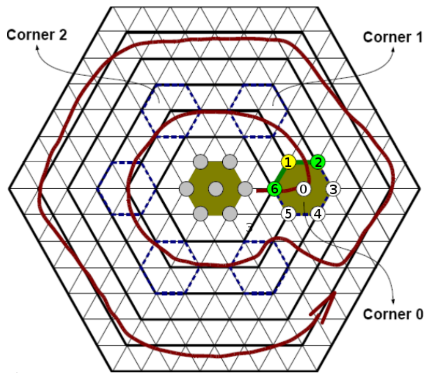
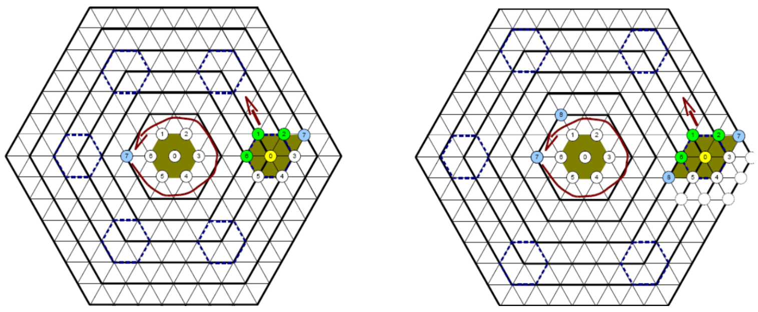


Figure 7.8: Simple case of deployment by mobile robots

More in details, let us call *frontier robots* the robots that are located in the front head of the group with respect to the current direction (that is, robots 2, 1, and 6 on the hexagon representing the group after the initial shift). Frontier robots are those in charge of dropping their sensors when they encounter an empty virtual vertex of the tesselation. Whenever the group arrives at a corner of the hexagon, the frontier nodes change according to the new direction (on the same example, the new frontier robots after the first corner has been reached will be robots 1, 6, and 5). If the frontier robots run out of sensors at some point, the second layer of robots (*e.g.* 5, 0, and 3 between Corner 0 and Corner 1) will take over the task. It is however expected that a re-organization of robots within the group may be required in some situations, especially if the initial supply in sensor is not uniform among robots.

The problem becomes more complex when the number of robots does not correspond to a perfect hexagon. Some sketches of solutions given in [Li09] are depicted on Figures 7.9(a) and 7.9(b). The first picture corresponds to the case where the outmost layer contains only one robot. It is suggested that robots in the inner hexagons behave as previously, while the robot in the outmost layer rotate around the group when a corner is encountered (this robot is expected to be relatively quickly depleted in sensors, thereby forcing more frequent re-organizations of the group). The second picture corresponds to cases where the outmost hexagon contains more than one robot. In this case, the robot team is to move using a different pattern. Here the team is a hexagon with two robots on the sides of the frontier. The other extra robots can be placed around the core, for example, at the positions marked by empty circles. As for the previous case, the outer robots will have to move within the group when a corner is encountered.



(a) One robot in the outmost layer (b) Several robots in the outmost layer

Figure 7.9: Robot team behavior

7.8 Augmentation algorithm for robot self-deployment

In the same vein as the previously discussed scenario, the deployment of a bi-connected network of sensors around a point of interest (POI) is addressed in [FNS09]. The major difference is that sensors are themselves endowed with movement capabilities, and are therefore capable of self-deploying around the POI. Here, the sensors are released one at a time from potentially various remote places.

The main idea of the proposed protocol is to incrementally build a perfect triangle tesselation around the POI (this tesselation structure is chosen here for the same geometrical reasons as previously discussed), while minimizing the sum of sensors moves. This protocol roughly works as follows: the first sensor moves directly to the exact position of the POI. Then, when a new sensor is released, it moves toward the POI until entering the range of a sensor that already belongs to the tesselation. At this point, the already deployed sensor is in charge of finding an appropriate tessellation position to ask the new node to move at.

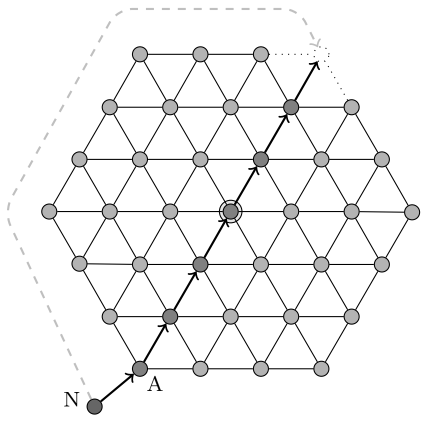


Figure 7.10: Minimizing the sum of movement to place a new node

The main objective here is to minimize the sum of robots movements while keeping the tesselation centered around the POI. The kind of choice resulting from these constraints is illustrated on Figure 7.10, where an empty tessellation vertex is available at the opposite of a new arriving node (node N). Here, we should prefer to shift every node in the diagonal instead of asking the new node to turn around the tessellation (because this latter movement represents a much larger overall distance). However, such decision implies that a tessellation node can be aware of, or inquire for, an empty remote position in the tessellation. Since forcing all nodes to memorize (and synchronize) a global view of the tessellation is not reasonable, the challenge will be to design a distributed protocol where nodes can collaboratively decide what destination can be assigned to a new node. Note that the question of how uniformly the movements are distributed among robots may arise as a second step, since here the already deployed nodes can be asked to move again afterward. Another interesting result could be to characterize an upper bound on the deploying time of one sensor, in order to determine how frequently they can be initially released (this time might increase with the number of sensors previously deployed).

7.9 Bi-Connectivity from connectivity without additional constraints

We now discuss some scenarios where mobile robots were already randomly deployed but still assumed 1-connected. From this initial connected network, the objective is to turn the network bi-connected using only localized movement decisions and minimizing the total movements of robots. The solution presented in this section comes from [DLNS09].

From a *global* point of view, a bi-connected network is a network that does not contain any *critical* nodes nor *critical* links, that is, that remains connected if any node or link is individually removed. Since a link is critical only if at least one of its endpoint nodes is critical, making the network bi-connected comes to turn every critical node into non-critical. By looking at the Figure 7.11, it appears intuitively clear that the *global* criticality of a node (*e.g.* N) cannot be locally decided, since it depends on some remote edges (*e.g.* AB) whose existence is locally unknown.

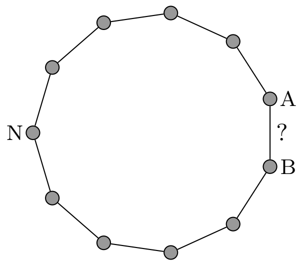


Figure 7.11: Locally critical node *vs.* globally critical node

The algorithm is based on the concept of *p-hop criticality*, already discussed in Section 7.6. More practically, each node is assumed to collect information about its p-hop neighbors through exchanging and relaying *hello* messages over multiple hops. If the p-hop neighborhood of a node n appears to n as disconnected without itself, then n locally decides that it is *p-hop critical* (we will simply say *critical* in the following). In the example depicted on Figure 7.11, N will decide that it is critical (unless $p \geq 5$).

By definition, the movement of any critical node is susceptible of disconnecting the network. The basic idea of this protocol is thus to use only non-critical nodes to create bi-connectivity, while keeping the critical nodes *static* (until they become in turn non-critical and able to move). The algorithm is thus based on a small set of pre-determined actions that critical nodes trigger on their non-critical neighbors according to the situation. In particular, the fact that a critical node has, or has not, other critical neighbors will generate different actions. Let us define a few more concepts before detailing these actions.

A critical node is called *available* if it has *at least one* non-critical neighbor, in other words if it has a neighbor that can move. This notion of *availability* can be used to decide which one among some critical neighbors is to pilot the local actions. More precisely, the pilot node in this case (also called *critical head*) must be *available* and have a *larger* ID than any of its *available* critical neighbors (in case of tie). Considering the example on Figure 7.12(a) (topology on the left), nodes 2, 4 and 5 are all critical. Since 4 is larger than 2 and node

4 is available, node 2 is not a critical head. Since 5 is larger than 4 and 5 is available, node 4 is not a critical head neither. Node 5, here, is the only critical head. Note that if node 2 had a larger ID than node 4, there would have been two critical heads here instead of one (nodes 2 and 5).

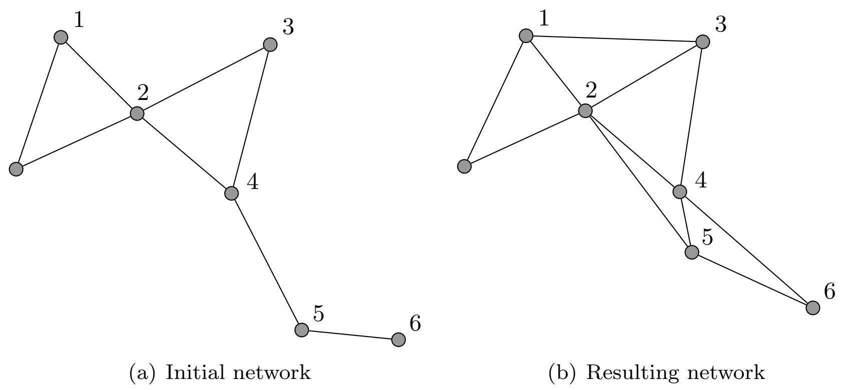


Figure 7.12: Simple scenario for DLNS.

The algorithm works as follows. At initialization stage, each node checks whether it is a p-hop critical node, and will continue to check it after every hello message exchanged. Whenever a node detects that it is p-hop critical, it broadcasts a *critical announcement* packet to all its direct neighbors, including the information about its *availability*. Two cases are then possible:

1. If the node has no critical neighbor, then it selects two of its neighbors n_1 and n_2 that belong to separate 'bi-connected component' and ask them to move towards each other. If the distance between n_1 and n_2 is d, then each node should move a distance of $(d-r)/2$, where r is the communication range. In case of several possible choices, the pair minimizing d is chosen.

2. If the node has one or several other critical neighbors, then it figures out whether it is critical head or not, and if so, it asks a non-critical neighbor to move toward one of the other critical neighbors. Here, the selected neighbor should move a distance of $d-r$. As for the previous case, the pair is selected so that both nodes belong to separate 'bi-connected components' with a distance d as small as possible.

If a node receives a request of movement while being already in the process of moving, it simply ignores it; if it receives several requests at the same time from different neighbors, it considers the request coming from the one with the largest ID. Note that the resulting movements may thus not create a new link at every expected place, which is the case for example when a node moves towards another, that finally moves elsewhere (in case 1). However, since the algorithm is incremental, this situation is likely to be solved at a later iteration.

Considering again the network given on Figure 7.12, we will describe the execution sequence transforming the initial topology (Fig. 7.12(a)) into a bi-connected topology (Fig. 7.12(b)). As explained above, nodes 2, 4, and 5 are initially critical, but only node 5 is a critical head. It is thus the first node to act by asking node 6 to move toward node 4. Then node 5 becomes non-critical and node 4, that has became critical head, asks node 5 to move toward node 2. Finally, node 2 remains the only critical node, and apply the first case by asking nodes 1 and 3 to move toward each other, after what the network is bi-connected.

This algorithm was experimentally compared to the centralized algorithm [BR04]. Simulation results showed that the total distance of movement of robots is significantly lower with the localized algorithm (about 2.5 times for networks with density 10). However, this algorithm does not totally guarantee bi-connectivity, and may even disconnect the network in some very particular case (e.g. when two non-critical nodes happen to be in-between some separate components, and they are both asked to move simultaneously away from each other).

7.10 Bi-Connectivity from connectivity with additional constraints

We are interested here in the same problem as in the previous section, that is to achieve bi-connectivity of mobile robots starting from a random (but connected) topology. Here however, the objective is also to maximize the overall coverage and minimize the network diameter at the same time. Every robot n is assumed to have a *communication* range, and a *coverage* range. The first, denoted $c(n)$, indicates up to which distance other nodes can receive messages from $c(n)$. The second, denoted $s(n)$, is the radius defining the area where the robot is to serve. For example, if n is a sensor, then $s(n)$ corresponds to its *sensing* range. If n is an actuator, then $s(n)$ may correspond to the area in which sensors are monitored by n. The ratio between these ranges is usually considered to satisfy $cr(n) > 2 \times sr(n)$. The *overall coverage* of the network is defined as the <u>union</u> of the coverage areas of all the robots. One can intuitively see that the closer the robots, the smaller the overall coverage due to potential overlappings.

The *diameter* of a network is defined as the 'largest' shortest path between any two nodes. More formally, if $d(u, v)$ is the length of the shortest path between two vertices u and v, then the diameter of a graph $G = (V, E)$ is defined as $max(d(u, v) : u, v \in V)$. Because the diameter bounds the number of hops of transmissions, making it small is crucial for most real-time applications. It can be intuitively seen that the closer the robots, the smaller the diameter of the network. Hence, the concepts of *diameter* and *overall coverage* appear somehow antinomic and may present contradictory objectives for an algorithm.

The Figure 7.13 illustrates this point with a topology of 4 bi-connected robots. On the left (Fig. 7.13(a)), the diameter has been reduced in the extreme (1), which generates a substantial overlap of the coverage areas. If we take the same network and move the nodes away from each other until their coverage areas do not overlap (Fig. 7.13(b)), then the diameter of the network increases (here to 2). However, as depicted on the right (Fig. 7.13(c)), some geometrical organizations such as the *triangle tessellation* seems to offer an interesting tradeoff between diameter and coverage.

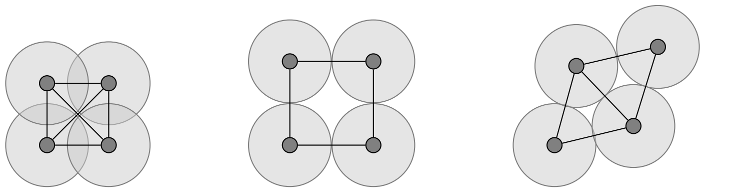

(a) Minimum diameter (b) Maximum coverage (c) Triangle tessellation

Figure 7.13: Three configuration examples.

Hence, it appears a good option to consider this pattern of organization. However, since we consider here a scenario where the nodes are initially already deployed, we do not want to built such perfect topology over the whole network (but rather use it to determine local organization of neighboring nodes). Let us nonetheless continue with some ideas illustrated on a whole tesselation. Using such structure, regulating the tradeoff between *coverage* and *diameter* can be done my merely tunning the distance between neighboring nodes without changing the organization geometry (the smaller distance, the higher priority for a small diameter). This is illustrated on Figure 7.14, where the left picture illustrates the choice to favor coverage (with a diameter of 3), and the right one the choice of reducing the diameter to 2 (but with overlapping coverages). Because the only difference between favoring coverage and diameter lies in the distance between nodes without modification of the topology geometry, this suggest that part of the tradeoff can be achieved using only localized operations (*e.g.* determining the local distance to direct neighbors).

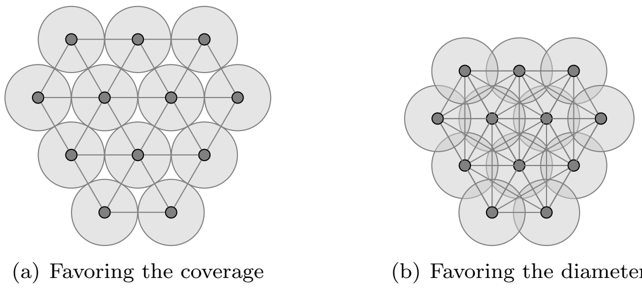

(a) Favoring the coverage (b) Favoring the diameter

Figure 7.14: Different choices of tradeoff

The algorithm presented in the previous section ([DLNS09]) can be adopted so that the resulting network tends to have a smaller diameter and/or a higher coverage depending on a given tradeoff parameter. This adaptation could also apply localized triangle tesselation pattern that helps improving both criteria at the same time.

As a by-product, the original algorithm already reduces the diameter of the network by moving the robot closer to one another. This aspect could however be further improved by changing the way the nodes move. For example, instead of simply asking two nodes to move half the distance toward one another to get connected, one could first check if one of them have no additional neighbor, and then ask only this one to move. More generally, making the nodes with lower degrees move more than the others could help decrease the network diameter (although it could also raise some new problems at the same time).

Once the network becomes bi-connected, a kind of localized triangle tesselation could be achieved by using repulsive forces, equally pushing the nodes away from one another, and therefore increasing the coverage. Here, the range of the force would serve as the tradeoff parameter between coverage and diameter. The Figure 7.15 shows a simple scenario where such forces were applied after the original algorithm. These forces may be applied once the first algorithm terminates, or they could be merged with it.

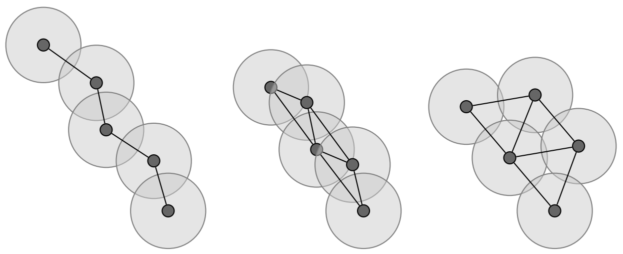

(a) Initial topology (b) After DLNS (c) After repulsion

Figure 7.15: Mixing bi-connectivity and repulsion forces.

The process of applying repulsive forces should however carefully avoid to *bi*-disconnect the network. Note that the use of virtual forces, introduced in [ZC03] in the context of mobile sensor networks, was recently used to maximize the overall coverage of a robot network.

In [GJK08], the authors consider an initial network that is not necessarily connected and propose an algorithm to make it connected. It is assumed that all the nodes know a common location (*e.g.* the base station) toward which they can move. Each robot that is arrived at this location (*e.g.* in range with the base station) floods a packet so that every node that receives the packet is thus aware of being connected with the others. Once all nodes are connected, virtual forces are applied to maximize the coverage of the network.

In [LCLD09], a localized protocol is proposed to bi-connect a network of robots from an initially non-connected topology. The objective is to minimize total moving distance of robots while maximizing sensing coverage of the network. It assumes that robots have a common communication range and a sensing range. Each robot is aware of locations of 1-hop neighbors and boundary of the sensing region. As with the previous solution, all robots are supposed to move toward a common position and maximize their sensing coverage. Here, however, nodes move by following only two kinds of virtual forces. The first is an attraction force that always draws every robot toward the common point of interest, and the second is a repulsive force that is applied between every pair of neighboring nodes. The simultaneous application of both forces (illustrated on one node in Figure 7.16 ends up in a bi-connected triangle tesselation centered on the POI (such as the one on Figure 7.17, that results from effective simulations). This is due to the fact that each node is located on a loop which surrounds the POI when the nodes come into equilibrium. Each node definitively stops moving when a certain number of changes in directions for obtuse angles, called *oscillations*, are detected. The final topology was proved to be bi-connected if the network is stable, and the proposed protocol was shown to be highly scalable. Another advantage of this solution is that no communication is ever required beyond 1-hop.

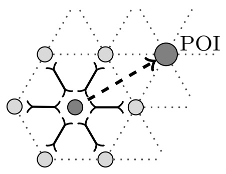


Figure 7.16: Neighboring repulsion and POI attraction.

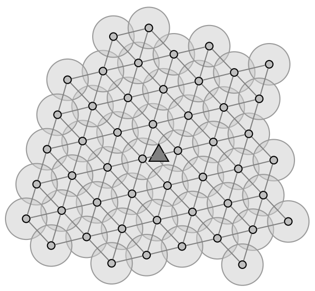


Figure 7.17: Topology resulting from 50 randomly deployed nodes.

Bibliography

[AKM+93] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 652–661, New York, NY, USA, 1993. ACM.

[AMZ06] S. Abbas, M. Mosbah, and A. Zemmari. Distributed computation of a spanning tree in a dynamic graph by mobile agents. *Engineering of Intelligent Systems, 2006 IEEE International Conference on*, pages 1–6, 0-0 2006.

[Bet02] C. Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 80–91, New York, NY, USA, 2002. ACM.

[BFG+03] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *Journal of Parallel and Distributed Computing*, 63:97–104, 2003.

[BK07] J. Burman and S. Kutten. Time optimal asynchronous self-stabilizing spanning tree. In *DISC*, pages 92–107, 2007.

[BLRS06] D.M. Blough, M. Leoncini, G. Resta, and P. Santi. The k-neighbors approach to interference bounded and symmetric topology control in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(9):1267–1282, 2006.

[BR04] P. Basu and J. Redi. Movement control algorithms for realization of fault-tolerant ad hoc robot networks. *Network, IEEE*, 18(4):36–44, July-Aug. 2004.

[CCGP09] A. Casteigts, S. Chaumette, F. Guinand, and Y. Pigné. Distributed maintainance of anytime available spanning trees in dynamic networks. Technical report, RR-1457-09 LaBRI, 2009.

[CSR04] J. Carle and D. Simplot-Ryl. Energy-efficient area monitoring for sensor networks. *Computer*, 37(2):40–46, Feb 2004.

[DLNS09] S. Das, H. Liu, A. Nayak, and I. Stojmenovic. A localized algorithm for bi-connectivity of connected mobile robots. *Telecommunication Systems*, 2009. to appear.

[FNS09] R. Falcon, A. Nayak, and I. Stojmenovic. Reliability oriented robot augmentation protocol. Technical report, University of Ottawa, 2009. In preparation.

[Gae03] F.C. Gaertner. A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms. Technical report, 2003.

[GCSRS06] A. Gallais, J. Carle, D. Simplot-Ryl, and I. Stojmenovic. Localized sensor area coverage with low communication overhead. pages 10–337, March 2006.

[GJK08] T. Guang, S.A. Jarvis, and A.-M. Kermarrec. Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks. pages 429–437, June 2008.

[JGK$^+$07] M. Jorgic, N. Goel, K. Kalaichevan, A. Nayak, and I. Stojmenovic. Localized detection of k-connectivity in wireless ad hoc, actuator and sensor networks. In *Proceedings of 16th International Conference on Computer Communications and Networks*, pages 33–38, 2007.

[JSHS04] M. Jorgic, I. Stojmenovic, M. Hauspie, and D. Simplot-Ryl. Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. In *Proc. of the Third Annual IFIP Mediterranean Ad Hoc Networking Workshop, MedHocNet*, pages 360–371, 2004.

[LCLD09] H. Liu, X. Chu, Y.-W. Leung, and R. Du. An efficient physical model for movement control towards bi- connectivity in robotic sensor networks. In preparation, 2009.

[LFSS07] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Localized self-deployment of mobile sensors for optimal focused-coverage formation. Technical report, Carleton University, Ottawa, 2007.

[LHS03] N. Li, J.C. Hou, and L. Sha. Design and analysis of an MST-based topology control algorithm. In *Proc. of INFOCOM*, 2003.

[Li09] X. Li. Deploying sensors for optimal coverage by bi-connected mobile robot team. Technical report, University of Ottawa, 2009. In preparation.

[MPGA05] T. M, D. Pompili, V.C. G, and I.F. A. A distributed coordination framework for wireless sensor and actor networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 99–110, New York, NY, USA, 2005. ACM.

[OSGS05] F.J. Ovalle-Martinez, I. Stojmenovic, F. Garcia-Nocetti, and J. Solano-Gonzalez. Finding minimum transmission radii for preserving connectivity and constructing minimal spanning trees in ad hoc and sensor networks. *J. Par. Distrib. Comput.*, 65(2):132–141, 2005.

[Pen99] M.D. Penrose. On k-connectivity for a geometric random graph. *Random Struct. Algorithms*, 15(2):145–164, 1999.

[SB03] P. Santi and D.M. Blough. The critical transmitting range for connectivity in sparse wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):25–39, 2003.

[TKS07] H.O. Tan, I. K, and I. Stojmenovic. A distributed and dynamic data gathering protocol for sensor networks. In *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, pages 220–227, Washington, DC, USA, 2007. IEEE Computer Society.

[XLNS09] C. Xu, X. Li, A. Nayak, and I. Stojmenovic. DHP: A delay-constrained power-efficient data aggregation scheme in sensor-actor networks. submitted for publication, 2009.

[ZC03] Y. Zou and K. Chakrabarty. Sensor deployment and target localization based on virtual forces. volume 2, pages 1293–1303 vol.2, March-3 April 2003.