# Detecting Separation in Robotic and Sensor Networks

Chenda Liao[a], Harshavardhan Chenji[b], Prabir Barooah[a], Radu Stoleru[b], Tamás Kalmár-Nagy[c]

[a]Dept. of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL.
[b]Dept. of Computer Science and Engineering, Texas A&M University, College Station, TX.
[c]Dept. of Aerospace Engineering, Texas A&M University, College Station, TX.

**Abstract**

In this paper we consider the problem of monitoring detecting separation of agents from a base station in robotic and sensor networks. Such separation can be caused by mobility and/or failure of the agents. While separation/cut detection may be performed by passing messages between a node and the base in static networks, such a solution is impractical for networks with high mobility, since routes are constantly changing. We propose a distributed algorithm to detect separation from the base station. The algorithm consists of an averaging scheme in which every node updates a scalar state by communicating with its current neighbors. We prove that if a node is permanently disconnected from the base station, its state converges to 0. If a node is connected to the base station in an average sense, even if not connected in any instant, then we show that the expected value of its state converges to a positive number. Therefore, a node can detect if it has been separated from the base station by monitoring its state. The effectiveness of the proposed algorithm is demonstrated through simulations, a real system implementation and experiments involving both static as well as mobile networks.

*Keywords:* mobile ad-hoc network, robotic network, sensor network, fault detection

## 1. Introduction

Sensor and robotic networks is a quickly developing area extending the boundaries of traditional robotics and usual sensor networks [1, 2]. In such a network, static as well as mobile nodes (robots) with varying levels of sensing, communication and actuation capability are used to observe, monitor, and control the state of physical processes. For example, in a scenario depicted in Figure 1, a team of ground robots may serve as information aggregators from a large number of static sensors deployed in an area as well as relays to send the processed data to more maneuverable autonomous aerial vehicles. We refer to all the devices that take part in sharing information, whether static or mobile as *nodes* or *agents*. Thus in Figure 1 the agents are the chopper, the mobile ground vehicles and the static sensors.

The communication topology of a robotic and sensor network is likely to change over time. Such changes can occur not only due to the mobility of the robotic nodes, but are also likely with static agents due to failures. An agent may fail due to various factors such as mechanical/electrical problems, environmental degradation, battery depletion, or hostile tampering. These causes are especially common for networks deployed in harsh and dangerous situations for applications such as forest fire monitoring, battlefield or emergency response operations [3].

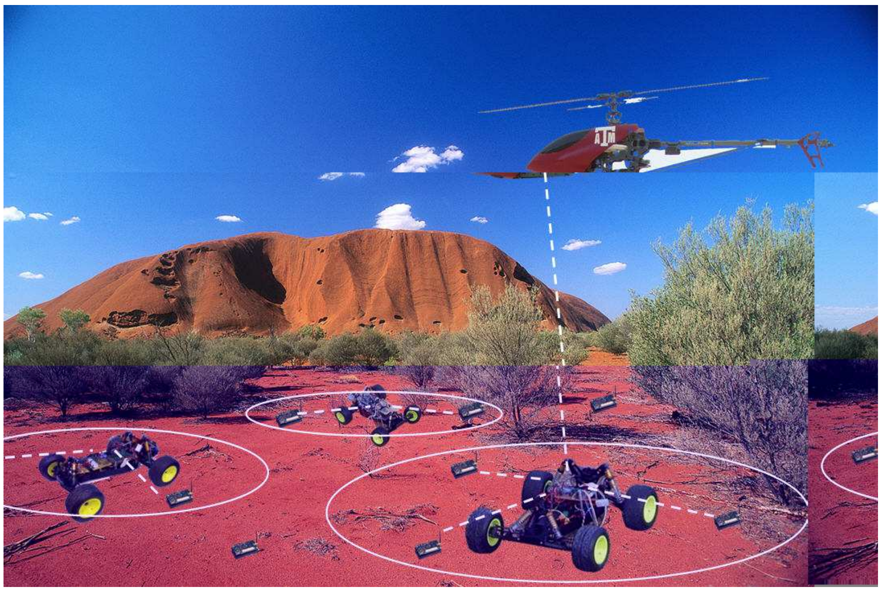


Figure 1: A heterogeneous robotic sensor network: ground robots aggregating information collected from a large number of static sensor nodes and relaying to an aerial robot.

Information exchange through wireless communication is a crucial ingredient for a sensor and robotic network system to carry out the tasks it is deployed for. Long range information exchange between agents is usually achieved by multi-hop wireless communication. In mobile networks, it is quite possible that the agents get separated into two or more sub-networks with no paths for data routing among these sub-networks. We say that a *cut* has occurred in such an event. A node that is disconnected from the base station at some time may also get reconnected later, e.g., when a mobile node moves in such a way that it restores connectivity between two disconnected sets of agents. In a robotic and sensor network, cuts can occur and disappear due to a combination of node mobility and node failure.

Multi-hop transmission typically requires routing data from a source node to a sink node, which is usually the base station. In a network with highly dynamic topology – common for sensor and robotic networks - maintaining updated routing tables, or discovering routing paths on demand, are challenging and energy inefficient tasks [4]. In such situations, sometimes information transfer from a node to a base station is accomplished by waiting till the node comes within range of the base station or to another node that is close to the base station [5]. In either scenario, it is imperative for the agents to know if a cut occurs, so necessary action can be taken. For example, once a cut is detected, mobile robots can attempt to heal the network by repositioning themselves or placing new communication relay nodes in critical regions. There are other advantages of an ability to detect separation from the base. If a node that is separated from the base station initiates data transmission to the base station, it will only lead to wastage of precious on-board energy of itself and its nearby nodes that attempt to route the packets to the base station, since no path to the destination exists. Therefore, after a cut occurs it is better for such nodes not to initiate any long-long information transfer. This requires the nodes to be able to monitor their connectivity to the base station. In addition, any proposed solution for separation detection cannot rely on centralized information processing, since separation will prevent information from reaching the central node.

A *cut* is defined for static networks as the separation of the network into two or more disjoint components [6]. However, for mobile networks we need to distinguish between a node getting disconnected from the rest of the nodes temporarily, for a short time interval, from getting disconnected for a very long time interval, or in the extreme case, all future time. A node may get disconnected temporarily due to mobility, or due to the failure of certain nodes, and then reconnected later. The more dynamic the topology is, the more likely that disconnections and re-connections will occur frequently. Therefore, what is needed is an ability of the nodes to detect if it has been disconnected from the source for a long enough time that will seriously hamper its ability to send data to the base station if the need arises. If the disconnection is for a very short period, that is not a serious cause for concern as the node can simply wait for a brief period to see if it gets connected again, which may occur due to the motion of itself or other nodes. We refer to the first as *intermittent disconnection* while the second is called *permanent disconnection*. The qualifier "permanent" is qualitative, it merely means "long enough" to necessitate some action on the part of the node as discussed earlier.

However, little attention has been paid to the problem of detecting cuts. Notable exceptions are Kleinberg *et al.* [7] - who study the problem of detecting network failures in wired networks – and Shrivastava *et al.* [6] and Barooah [8], who propose algorithms for detecting cuts in wireless sensor networks. The problem of detecting cuts in mobile networks has attracted even less attention. The solutions proposed in [9, 10] require routing packets from the base station to each node periodically. When a node fails to receive this packet for a certain amount of time, it suspects that a cut has occurred. This approach, however, requires routing data between far away nodes, which is challenging in networks of mobile nodes. While algorithms for coordinating the motion of the agents to ensure network connectivity has been developed in recent times (see, for example, [11]), such algorithms cannot guarantee network connectivity under all circumstances. This is especially true when the robotic agents are operating in harsh and uncertain environments. Thus, there is a need to develop distributed algorithms for cut detection in robotic and sensor networks.

In this paper we describe a simple algorithm for cut detection in robotic and sensor networks. The algorithm is applicable to networks made up of static or mobile agents, or a combination of the two. This algorithm – called Distributed Source Separation Detection (DSSD) algorithm – is designed to allow every node to monitor if it is connected to a specially designated node, the so-called *source node*, or if it has been disconnected from the source node. The source node is usually a base station. The reason for this terminology comes from an electrical analogy that the algorithm is based on. The idea is quite simple: imagine the wireless communication network as an electrical circuit where current is injected at the source node. When a cut separates certain nodes from the source node, the potential at these nodes becomes zero. If a node is connected to the source node through multi-hop paths, either always or in a time-average sense, the potential is positive in a time-average sense. In the DSSD algorithm, every nodes updates a scalar (called its *state*) which is an estimate of its virtual potential in the fictitious electrical network. The state update is performed by a distributed algorithm that only requires a node to communicate to its neighbors. By monitoring their computed estimates of the potential, nodes can detect if they are connected to the source node or not. In addition, the iterative scheme used in computing the node potentials is extremely fast, which makes the algorithm scalable to large networks. The proposed DSSD algorithm is fully distributed and asynchronous. It requires communication only between neighbors, multi-hop routing between node pairs that are not neighbors is not needed.

Performance of the algorithm in networks of static nodes was examined through simulations previously in [8]. Here we extend the algorithm to both static and mobile networks. In the mobile case, by modeling the evolution of the network over time as a Markov chain, we show that the node states converge to positive numbers in an expected sense as long a path exists in a time-average sense between the node and the base station. The performance of the algorithm has been tested in simulations as well as in an experimental platform with mobile robots and human agents. These tests demonstrate the effectiveness of the algorithm in detecting separation (and reconnection) of nodes in both static and mobile networks. Since there is no existing prior work on the problem of detecting separation in mobile networks that can operate without multi-hop routing, we donot present comparison of the proposed algorithm with existing algorithms for cut detection.

The rest of the paper is organized as follows. In Section 2 we introduce the DSSD algorithm. The rationale behind the algorithm and its theoretical properties are described in Section 3. Sections 4 and 5 describe results from computer simulations and experimental evaluations. The paper concludes with a summary in Section 6.

## 2. The Distributed Source Separation Detection (DSSD) Algorithm

We introduce some terminology about graphs that will be needed to describe the algorithm and the results precisely (see for example the excellent treatise by Diestel [12]). Given a set $\mathcal{V} = \{v_1, \dots, v_m\}$ of $m$ elements referred to as *vertices*, and a set $\mathcal{E} = \{(v_i, v_j) \,|\, v_i, v_j \in \mathcal{V}\}$ of *edges*, a *graph* $\mathcal{G}$ is defined as the pair $(\mathcal{V}, \mathcal{E})$. A sensor network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose vertex set $\mathcal{V}$ corresponds to the wireless sensor nodes and whose edges $\mathcal{E}$ describe direct communication between nodes. The size of a graph $\mathcal{G}$ is the number of its vertices $|\mathcal{V}|$. The graphs formed by the nodes of the sensor and robotic network are assumed to be *undirected*, i.e. the communication between two nodes is assumed to be symmetric. In the language of graph theory, the edges of an undirected graph are unordered pairs with the symmetry relation $(v_i, v_j) = (v_j, v_i)$. The *neighbors* of vertex $v_i$ is the set $\mathcal{N}_i$ of vertices connected to $v_i$ through an edge, i.e. $\mathcal{N}_i = \{v_j | (v_i, v_j) \in \mathcal{E}\}$. The number of neighbors of a vertex $|\mathcal{N}_i|$ is called its *degree*. A graph is called *connected* if for every pair of its vertices there exists a sequence of edges connecting them.

In a mobile sensor and robotic networks the neighbor relationship can change with time, so the graph in our study are in general time varying: $\mathcal{G}(k) = (\mathcal{V}, \mathcal{E}(k)$, where $k = 0, 1, \dots$ is a discrete time index. Note that we assume the set $\mathcal{V}$ of nodes does not change over time; though certain nodes may fail permanently at some time and thereafter not take part in the operation of the network.

In the proposed scheme, every node $v_i$ maintains a scalar variable $x_i(k)$ in its local memory, which is called its *state*. The base station is designated as the *source node*, though in principle any node can be the source node. The reason for this terminology will be explained soon. For ease of description (and without loss of generality), the index of the source node is taken as 1. The DSSD algorithm is an iterative process consisting of two phases at every discrete step: (i) *State Update* and (ii) *Cut Detection from State*.

DSSD PHASE I (**State update law**): The scalar state $x_i(k)$ assigned to node $v_i$ is iteratively updated (starting with $x_i(0) = 0$) according to the following update rule. Recall that the index $i = 1$ corresponds to the source node.

$$x_i(k+1) = \begin{cases} \frac{1}{d_1(k)+1}\left(\sum\limits_{v\in\mathcal{N}_1(k)} x_i(k) + s\right) & i = 1 \\ \frac{1}{d_i(k)+1}\sum\limits_{v\in\mathcal{N}_i(k)} x_i(k) & i > 1 \end{cases}. \tag{1}$$

where $\mathcal{N}_i(k)$ is the set of neighbors of $v_i$ in graph $\mathcal{G}(k)$ and $d_i(k) := |\mathcal{N}_i(k)|$ is the degree of $v_i$ (number of neighbors) at time $k$, and the $s > 0$ is an arbitrary fixed positive number that is called the *source strength*. The source strength is a design parameter and has to be provided to the source node a-priori.

DSSD PHASE II (**Cut detection from state**): Every node $v_i$ maintains determines its connectivity to the source node by comparing its state $x_i(k)$ to the *cut detection threshold* $\epsilon$ (a small positive number) as follows:

$$\text{cut\_belief}_i(k) = \begin{cases} 0 & x_i(k) > \epsilon \\ 1 & x_i(k) \le \epsilon \end{cases} \tag{2}$$

where $\text{cut\_belief}_i = 1$ means the node believes it is disconnected from the source and 0 means it believes it is connected to the source.

The rationale for the algorithm comes from the interpretation of the states in terms of the potentials in an electrical circuit. If the network does not change with time, then the state of a node that is connected to the source converges to positive number that is equal to its electrical potential in a fictitious electrical network. If a node is disconnected from the source then its state converges to 0. When the network is time-varying, then too the states can be shown to converge in a mean-square sense to either positive numbers or 0 depending on whether the node is connected or not (in some appropriate stochastic sense) to the source. We discuss the details of the electrical analogy and the theoretical performance guarantees that can be provided for the proposed algorithm in the next section.

We note that the cut detection threshold $\epsilon$ is a design parameter, and it has to be provided to all the nodes a-priori. The value of $\epsilon$ chosen depends on the source strength $s$. Smaller the value of $s$, the smaller the value of $\epsilon$ that has to be chosen to avoid false separation detection. We also note that the algorithm as described above assumes that all updates are done synchronously, or, in other words, every node shares the same iteration counter $k$. In practice, the algorithm is executed asynchronously without requiring any clock-synchronization or keeping a common time counter. To achieve this, every node keeps a buffer of the last received states of its neighbors. If a node does not receive messages from a neighbor during a time-out period, it updates its state using the last successfully received state from that neighbor.

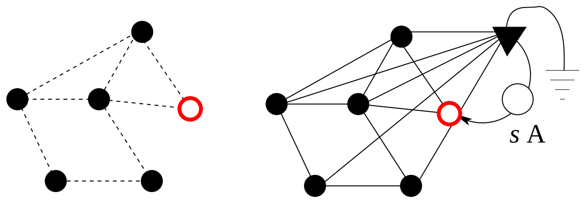


Figure 2: A graph describing a sensor network (left), and the associated electrical network (right). In the electrical network, one node is chosen as the source that injects $s$ Ampere current into the network, and additional nodes are introduced (fictitiously) that are grounded, through which the current flows out of the network. The thick line segments in the electrical network are resistors of 1Ω resistance.

When a node does not receive broadcasts from one of its neighbors for sufficiently long time, it removes that neighbor from its neighbor set and carries on executing the algorithm with the remaining neighbors.

## 3. Algorithm Explanation and Theoretical Results

When the graph does not change with time, i.e., $\mathcal{G}(k) = \mathcal{G}$ for some $\mathcal{G}$, the state update law is an iterative method for computing the node potentials in a fictitious electrical network $\mathcal{G}^e = (\mathcal{V}^e, \mathcal{E}^e)$ that is constructed from the graph $\mathcal{G}$ as follows. First, define $\mathcal{V}^e := V \cup \{g\}$ where $g$ is a fictitious *grounded node* $g$ and next, introduce $n$ additional edges to connect each of the $n$ node in $\mathcal{V}$ to the node $g$ with a single edge. So the edges in $\mathcal{E}^e$ consist of the edges in $\mathcal{E}$ and the newly introduced edges. Now an electrical network $(\mathcal{G}^e, 1)$ is imagined by assigning to every edge of $\mathcal{G}^e$ a unit resistance. Figure 2 shows a physical network and the corresponding fictitious electrical network. It can be shown that in a time invariant network, the node states resulting from the state update law always converge to constants [8]. In fact, the limiting value of a node state in a graph $\mathcal{G}$ is the potential of the node in the fictitious electrical network $\mathcal{G}^e$ in which $s$ Ampere current is injected at the source node and is extracted at a fictitious grounded node; which is always maintained at potential 0 by virtue of being grounded (see Theorem 1 of [8]).

The evolution of the node states for a *static network of nodes* was analyzed in [8], where it was shown that for a time invariant graph that is connected (and therefore every node is connected to the source), the state of every node converges to a positive number (see Theorem 1 of [8]). For nodes that are disconnected from the source, it was shown that their states converge to 0, and in fact this result holds even if the component(s) of nodes that are disconnected from the source are changing with time due to mobility etc. We state the precise result below:

**Theorem 1.** *[8] Let the nodes of a sensor network with an initially connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ execute the DSSD algorithm starting at time $k = 0$ with initial condition $x_i(0) = 0,\ i = 1, \dots, |\mathcal{V}|$. If a node $v_i$ gets disconnected from the source at a time $\tau > 0$ and stays disconnected for all future time, then its state $x_i(k)$ converges to $0$ as $k \to \infty$.*

This result is useful in detecting disconnection from the source; if the state of node converges to 0 (which can be determined by checking if it becomes smaller than a threshold), then the node detects that it is disconnected from the source. This partially explains the logic behind the Phase II of the algorithm.

To detect connectivity to the source, we have to answer the question: how do the states of nodes that are *intermittently connected* to the source due to their own – or other nodes' – mobility evolve? In a mobile network the graph at any time instant is essentially random since it depends on which nodes are within range of which other nodes at that instant, which is difficult to predict in all but the most trivial situations. We therefore model the evolution of the graph $G(k)$ as a stochastic process. In that case the state of every node also is a random variable, whose value depends on the evolution of the graphs. Assuming the evolution of the graph can be described by a Markov chain, we then show that the node states converge in the mean square sense. Meaning, the mean and variance of each node's state converge to specific values. We also provide formulas for these values.

We consider the case when the sequence of graphs $\{\mathcal{G}(k)\}_{k=0}^{\infty}$ can be modeled as the realization of a Markov chain, whose state space $\mathbb{G} := \{\mathcal{G}_1, \dots, \mathcal{G}_N\}$ is the set of graphs that can be formed by the mobile nodes due to their motion. The network at time $k$ can be any one of the elements of the set $\mathbb{G}$, i.e., $\mathcal{G}(k) \in \mathbb{G}$. The Markovian property means that if $\mathcal{G} \in \mathbb{G}$, then $\Pr(\mathcal{G}(k+1) = \mathcal{G} | \mathcal{G}(k)) = \Pr(\mathcal{G}(k+1) = \mathcal{G} | \mathcal{G}(k), \mathcal{G}(k-1), \dots, \mathcal{G}(0))$, where $\Pr(\cdot)$ denotes probability. A simple example in which the time variation of the graphs satisfies the Markovian property is that of a network of static nodes with unreliable communication links such that each link can fail temporarily, and the failure of each edge at every time instant $k$ is independent of the failures of other links and the probability of its failure is time-invariant. Another example is a network of mobile agents whose motion obeys first order dynamics with range-determined communication. Specifically, suppose the position of node $v_i$ at time $k$, denoted by $p_i(k)$, is restricted to lie on the unit sphere $\mathbf{S}^2 = \{x \in \mathbb{R}^3 | \|x\| = 1\}$, and suppose the position evolution obeys: $p_i(k+1) = f(p_i(k) + \Delta_i(k))$, where $\Delta_i(k)$ is a stationary zero-mean white noise sequence for every $i$, and $\mathrm{E}[\Delta_i(k)\Delta_j(k)^T] = 0$ unless $v_i = v_j$. The function $f(\cdot) : \mathbb{R}^3 \to \mathbf{S}^2$ is a projection function onto the unit-sphere. In addition, suppose $(v_i, v_j) \in \mathcal{E}(k)$ if and only if the geodesic distance between them is less than or equal to some predetermined range. In this case, prediction of $\mathcal{G}(k+1)$ given $\mathcal{G}(k)$ cannot be improved by the knowledge of the graphs observed prior to $k$: $\mathcal{G}(k-1), \dots, \mathcal{G}(0)$, and hence the change in the graph sequence satisfies the Markovian property. If no restriction is placed on the motion of the nodes or edge formation, the number of graphs in the set $\mathbb{G}$ is the total number of distinct graphs possible with $n$ nodes. In that case, $N = 2^{\frac{1}{2}n(n-1)}$, where $N := |\mathbb{G}|$. If certain nodes are restricted to move only within certain geographic areas, $N$ is less than this maximum number.

We assume that the Markov chain that governs the evolution of the graphs $\{G(k)\}_{k=0}^{\infty}$ is homogeneous, and denote the transition probability matrix of the chain by $\mathcal{P}$. The following result states under what conditions the node states converge in the mean square sense and when the limiting mean values are positive. The proof of the result is provided in Section 3.1. In the statement of theorem, the *state vector* is the vector of all the node states: $\mathbf{x}(k) := [x_1(k), \dots, x_n(k)]^T$, and the *union graph* $\hat{\mathcal{G}} := \cup_{i=1}^{N} \mathcal{G}_i$ is the graph obtained by taking the union of all graphs in the set $\mathbb{G}$, i.e., $\hat{\mathcal{G}} = (\mathcal{V}, \cup_{i=1}^{N} \mathcal{E}_i)$.

**Theorem 2.** *When the temporal evolution of the graph* $\mathcal{G}(k)$ *is governed by a Markov chain that is ergodic, the state vector* $\mathbf{x}(k)$ *converges in the mean square sense. More precisely, for every initial condition* $\mathbf{x}(0)$, *there exists vector*

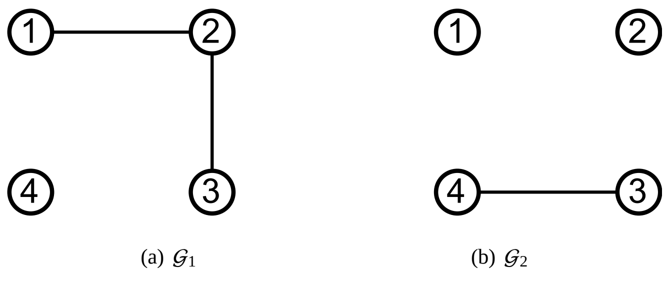


Figure 3: An example of two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ that appear at different times due to the motion of 4 mobile nodes. If these two graphs form the set of all possible graphs that can ever appear, then $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$. Note that neither $\mathcal{G}_1$ nor $\mathcal{G}_2$ is connected, but the union graph $\cup_i \mathcal{G}_i$ is.

*$\mu \in \mathbb{R}^n$ and a symmetric positive semi-definite matrix $\mathbf{Q}$ so that $\mathrm{E}[\mathbf{x}(k)] \to \mu$ and $\mathrm{E}[\mathbf{x}(k)\mathbf{x}(k)^T] \to \mathbf{Q}$. Moreover, the vector $\mu$ is entry-wise non-negative. If $P$ is entry-wise positive, we have $\mu(i) > 0$ if and only if there is a path from node $v_i$ to the source node $v_1$ (with index 1) in the union graph $\hat{\mathcal{G}}$.*

Theorems 2 and 1 together explain the rationale behind Phase II of the DSSD algorithm. The state of a node converges to 0 if and only if it is permanently disconnected from the source. If it is connected to the source in the union graph, meaning it is connected in a time-average sense (even if it is not connected in every time instant), then the expected value of its state converges to a positive number. As a result, a node can detect if it is connected to the source or not simply by checking if its state has converged to 0, which is done by comparing the state to the threshold $\epsilon$.

A closed-form expression for the limiting mean of the nodes states, i.e., the vector $\mu$, and its correlation matrix $\mathbf{Q}$, is also provided in Lemma 1 in Section 3.1. We refrain from stating them here as the expressions require significant amount of terminology to be introduced. The ergodic property of the Markov chain assumed in Theorem 2 ensures that the chain has a steady state distribution which is an entry-wise positive vector. Intuitively, ergodicity of the chain means that every graph in the set $\mathbb{G}$ appears infinitely often as time progresses. In other words, the network $\mathcal{G}(k)$ does not get stuck in one of the graphs in the set $\mathbb{G}$. As a result, if a node is connected to the source in the union graph, even if it is not connected to the source in any one of the graphs that can ever appear, there is still a path for information flow between the source node and this node over time. Figure 3 shows a simple example in which the network $\mathcal{G}(k)$ can be only one of the two graphs shown in the figure. Node 4 is disconnected from the source at every $k$, but it is connected in a time-average sense since information from the source node 1 can flow to 2 in one time instant when the graph $\mathcal{G}_1$ occurs and then from 2 to 4 in another time instant when $\mathcal{G}_2$ occurs. In a such a situation the theorem guarantees that the expected value of the state of the node 4 will converge to positive number. Thus, node 4 can detect that it is connected to the source in a time-average sense. On the other hand, if a node is not connected to the source in the union graph there is no path for information flow between itself and the source over all time. This is equivalent to the node being permanently disconnected from the source, so the result that the mean of the node's state converges to 0 is consistent with the result of Theorem 1. The condition that $\mathcal{P}$ is entry-wise positive means that there is a non-zero probability that the network at time $k$ can transition into any one of the possible graphs at time $k+1$, even though most of these probabilities maybe small. We believe this sufficient condition is merely an artifact of our proof technique, and in fact, this is not a necessary condition for convergence to occur.

**Remark 1.** *An advantage of the DSSD algorithm is that its convergence rate for a time-invariant network is independent of the number of agents in the network (Lemma 1 in [8]). This makes the algorithm scalable to large networks. The convergence rate of mean and variance of the nodes states to their limiting values in the mobile case, however, requires further research.*

### *3.1. Proof of Theorem 2*

We start with some notation. let $D(k)$ be the *degree matrix* of $\mathcal{G}(k)$, i.e., $D(k) := \text{diag}(d_1(k), \ldots, d_n(k))$. If node $i$ fails at $k_0$, we assign $d_i(k) = 0$ and $\mathcal{N}_i(k) = \phi$ (the empty set) for $k \geq k_0$. Let $A(k)$ be the *adjacency matrix* of the graph $\mathcal{G}(k)$, i.e., $A_{i,j}(k) = 1$ if $(i, j) \in \mathcal{E}(k)$, and 0 otherwise. With these matrices, (1) can be compactly written as:

$$\mathbf{x}(k+1) = (D(k) + I)^{-1} \left(A(k)\mathbf{x}(k) + s\, e_1\right) \tag{3}$$

where $e_1 = [1, 0, \ldots, 0]^T$. Recall that the source node has been indexed as node $v_1$. The above can be written as

$$\mathbf{x}(k+1) = J(k)\mathbf{x}(k) + B(k)w(k) \tag{4}$$

where the matrices $J, B$ and the vector $w$ are defined as

$$J(k) := (D(k) + I)^{-1}A(k), \qquad B(k) = (D(k) + I)^{-1}, \qquad w(k) := se_1. \tag{5}$$

Under the assumption that the temporal evolution of the graphs $\mathcal{G}(k)$ is governed by a Markov chain, we can write (3) in the standard notation of *jump-linear system* [13]

$$\mathbf{x}(k+1) = J_{\mathcal{G}(k)}\mathbf{x}(k) + B_{\mathcal{G}(k)}w(k) \tag{6}$$

where $J_{\mathcal{G}(k)} = J(k)$, $B_{\mathcal{G}(k)} = B(k)$, and $w(k) = w = se_1$. This notation is used to emphasize that the state and input matrices $J$ and $B$ of the linear system (6) change randomly, and the transition is governed by a Markov chain.

Let $\mu(k) := \text{E}[\mathbf{x}(k)]$, $\mathbf{Q}(k) := \text{E}[\mathbf{x}(k)\mathbf{x}(k)^T]$ be the mean and correlation of the state vector $\mathbf{x}(k)$, respectively. We need the following definitions and terminology from [13] to provide expressions for these quantities as well as to state conditions for the convergence of the mean and correlation. Let $\mathbb{R}^{m\times n}$ be the space of $m \times n$ real matrices. Let $\mathbb{H}^{m\times n}$ be the set of all N-sequences of real $m \times n$ matrices $Y_i \in \mathbb{R}^{m\times n}$. That is, if $Y \in \mathbb{H}^{m\times n}$ then $Y = (Y_1, Y_2, \ldots, Y_N)$, where each $Y_i$ is an $m \times n$ matrix. The operators $\varphi$ and $\hat{\varphi}$ are defined as follows: let $(y_i)_j \in \mathbb{R}^m$ be the $j$-th column of $Y_i \in \mathbb{R}^{m\times n}$, then

$$\varphi(Y_i) := \begin{pmatrix} (y_i)_1 \\ \vdots \\ (y_i)_n \end{pmatrix} \qquad \text{and} \qquad \hat{\varphi}(Y) := \begin{pmatrix} \varphi(Y_1) \\ \vdots \\ \varphi(Y_N) \end{pmatrix} \tag{7}$$

Hence, $\varphi(Y_i) \in \mathbb{R}^{mn}$ and $\hat{\varphi}(Y) \in \mathbb{R}^{Nmn}$. Similarly, define an inverse function $\hat{\varphi}^{-1} : \mathbb{R}^{Nmn} \to \mathbb{H}^{m\times n}$ that produces an element of $\mathbb{H}^{m\times n}$ given a vector in $\mathbb{R}^{Nmn}$. For $X_i \in \mathbb{R}^{n\times n}$ for $i = 1, \ldots, N$, define

$$diag[X_i] := \begin{pmatrix} X_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & X_N \end{pmatrix} \in \mathbb{R}^{Nn\times Nn}. \tag{8}$$

For a set of square matrices $C_{ij} \in \mathbb{R}^{m\times m}$, $i, j = 1, \ldots, N$, we also use the notation $C = [C_{ij}]$ to denote the following matrix:

$$C = [C_{ij}] := \begin{bmatrix} C_{11} & C_{12} & \ldots & C_{1N} \\ C_{21} & C_{22} & \ldots & C_{2N} \\ \ldots & \ldots & \ldots & \ldots \\ C_{N1} & C_{N2} & \ldots & C_{NN} \end{bmatrix} \in \mathbb{R}^{Nm\times Nm}$$

In context of the jump linear system (6), define the matrices

$$\begin{aligned} \mathcal{C} &:= (\mathcal{P}^T \otimes I_n) diag[J_i] \in \mathbb{R}^{Nn} \\ \mathcal{D} &:= (\mathcal{P}^T \otimes I_{n^2}) diag[J_i \otimes J_i] \in \mathbb{R}^{Nn^2}, \end{aligned} \tag{9}$$

where $I_n$ is the $n\times n$ identity matrix, $\mathcal{P}$ is the transition probability matrix of the Markov chain and $\otimes$ denotes Kronecker product. Note that the matrices $\mathcal{C}$ and $\mathcal{D}$ can be expressed, using the notation introduced above, as

$$\mathcal{C} = [p_{ji} J_j] \qquad \mathcal{D} = [p_{ji} F_j] \text{ where } F_i := J_i \otimes J_i, \tag{10}$$

where $p_{ij}$ is $(i, j)$-th entry of $\mathcal{P}$ and $\pi \in \mathbb{R}^{1\times N}$ is the stationary distribution of the Markov chain, which exists due to the ergodicity directly followed by assumption of positive $\mathcal{P}$.

For a matrix $X$, we write $X \geq 0$ to mean that $X$ is entry-wise non-negative and write $X \succ 0$ to mean $X \geq 0$ and $X \neq 0$. If every entry of $X$ is positive, we write $X \gg 0$. For a matrix $X$, $X \geq (>) 0$ means it is positive semi-definite (definite). For two matrices $X$ and $Y$ of compatible dimension, we write $X \geq Y$ if $X_{ij} \geq Y_{ij}$ for all $i$, $j$, and write $X \succ Y$ if $X \geq Y$ and $X \neq Y$. For a vector $x$, we write $x \geq 0$ to mean $x$ is entry-wise non-negative, $x \succ 0$ to mean $x$ is entry-wise non-negative and at least one entry is positive, and $x \gg 0$ to mean every entry of $x$ is positive. The fact that both $J$ and $B$ are entry-wise non-negative will be useful later.

We will also need the following technical results to prove Theorem 2.

**Proposition 1** ([14], Theorem 3.2)**.** *Let $C = [C_{ij}] \in \mathbb{R}^{mn\times mn}$ be a block $m \times m$ matrix, where $C_{ij}$ are non-negative $n \times n$ matrices for all $i, j = 1, \ldots, m$ and let $\tilde{C} = [\|C_{ij}\|] \in \mathbb{R}^{m\times m}$, where $\|\cdot\|$ is either the induced 1-norm ($\|\cdot\|_1$) or the induced $\infty$-norm ( $\|\cdot\|_\infty$). Then $\rho(C) \leq \rho(\tilde{C})$.*

**Proposition 2.** *Let $\mathcal{G}$ be an undirected graph with n nodes, and let $J = (D + I)^{-1}A$, where D and A are the degree and adjacency matrices, respectively. $\|J\|_\infty < 1$ and $\|J \otimes J\|_\infty < 1$.*

*Proof.* Based on examining the structure of $J$, we see that $J_{ij} = \dfrac{1}{d_i + 1} 1_{\{\mathcal{N}_i\}}(j)$, where $1_{\{A\}}(x)$ is 1 if $x \in A$ and 0 otherwise. Obviously, $J$ is non-negative matrix and so is $J \otimes J$. Meanwhile, each row sum of $J$ is at most $n - 1/n$, where $n$ is the number of nodes. Therefore $\|J\|_\infty \leq \frac{n-1}{n} < 1$, which leads to $\|J \otimes J\|_\infty = \|J\|_\infty \|J\|_\infty < 1$, where the equality is a result of properties of the Kronecker product [15, Section 2.5]. □

**Proposition 3.** *If the temporal evolution of the graph $\mathcal{G}(k)$ is governed by a Markov chain that is ergodic, then we have $\rho(C) < 1$ and $\rho(\mathcal{D}) < 1$, where $C, \mathcal{D}$ are defined in* (9).

*Proof.* Since $\rho(C) = \rho([p_{ji} J_j])$ (see (10)), we obtain by applying Proposition 1 that

$$\rho(C) \leq \rho(\left[\|p_{ji} J_j\|_\infty\right]) = \rho(\left[p_{ji} \|J_j\|_\infty\right]),$$

where the equality follows from $p_{ij}$'s being probabilities and therefore non-negative. Since $\|J\| < 1$ (Proposition 2), it follows that $P^T > [p_{ji}\|J_j\|_\infty]$. Since both $P^T$ and $[p_{ji}\|J_j\|_\infty]$ are non-negative, and $P^T$ is irreducible (which follows from the ergodic assumption of the Markov chain), it follows from Corollary 1.5 of [16, pg. 27] that $\rho([p_{ji}\|J_j\|_\infty]) < \rho(\mathcal{P}^T) = \rho(\mathcal{P}) = 1$, the last equality being a property of a transition probability matrix. This proves that $\rho(C) < 1$.

To show that $\rho(\mathcal{D}) < 1$, since $\rho(\mathcal{D}) = \rho([p_{ji} F_j])$ (see (10)), we obtain by applying Proposition 1 that

$$\rho(\mathcal{D}) \leq \rho(\left[\|p_{ji} F_j\|_\infty\right]) = \rho(\left[p_{ji} \|F_j\|_\infty\right]),$$

where the equality follows from $p_{ij}$'s being probabilities and therefore non-negative. Since the scalars $\|F_j\|_\infty$ satisfy $\|F_j\|_\infty < 1$ for each $j$ (see Proposition 2), it follows that $P^T > [p_{ji}\|F_j\|_\infty]$. Since both $P^T$ and $[p_{ji}\|F_j\|_\infty]$ are non-negative, and $P^T$ is irreducible (which follows from the ergodic assumption of the Markov chain), it follows from Corollary 1.5 of [16, pg. 27] that $\rho([p_{ji}\|F_j\|_\infty]) < \rho(\mathcal{P}^T) = \rho(\mathcal{P}) = 1$, the last equality being a property of a transition probability matrix. This proves that $\rho(\mathcal{D}) < 1$. □

The proof of Theorem 2 will require the following result.

**Lemma 1.** *Consider the jump linear system* (6) *with the an underlying Markov chain that is ergodic. If $\rho(\mathcal{D}) < 1$, where $\mathcal{D}$ is defined in* (9)*, then the state $\mathbf{x}(k)$ of the system* (6) *converges in the mean square sense, i.e., $\mu(k) \to \mu$ and $\mathbf{Q}(k) \to Q$, where $\mu$ and $Q$ are given by*

$$\mu := \sum_{i=1}^{N} q^{(i)} \quad \mathbf{Q} := \sum_{i=1}^{N} Q_i. \tag{11}$$

*where*

$$[q^{(1)^T}, \ldots, q^{(N)^T}]^T = q := (I - C)^{-1} \psi \quad (q \in \mathbb{R}^{Nn})$$
$$(Q_1, \ldots, Q_N) = Q := \hat{\varphi}^{-1}\left((I - \mathcal{D})^{-1} \hat{\varphi}(R(q))\right), \quad (Q \in \mathbb{H}^{n \times n})$$

*where*

$$\psi := [\psi_1^T, \dots, \psi_N^T]^T \in \mathbb{R}^{Nn} \qquad \textit{and} \qquad \psi := \sum_{i=1}^{N} p_{ij} B_i w \pi_i \in \mathbb{R}^n$$

$$R(q) := (R_1(q), \dots, R_N(q)) \in \mathbb{H}^{n \times n} \quad \textit{and} \quad R_j(q) := \sum_{i=1}^{N} p_{ij}(B_i w w^T B_i^T \pi_i + J_i q^{(i)} w^T B_i^T + B_i w {q^{(i)}}^T J_i^T)) \in \mathbb{R}^{n\times n},$$

*and $C$ is defined in* (9)*, $\pi_i$ is the i-th entry of the steady state distribution of the Markov chain, and $J_i, B_i$ are the system matrices in* (6)*. Moreover, $\mu \geq 0$.*

*Proof.* The first statement about mean square convergence follows from standard results in jump linear systems, as do the expressions for the mean and correlation; see [13, Proposition 3.37]. Note that the existence of the steady state distribution $\pi$ follows from the ergodicity of the Markov chain.

To show that $\mu$ is entry-wise non-negative, note that since $\rho(C) < 1$ (Proposition 3), we have $\mathcal{M} := (I - C)^{-1} = \sum_{k=0}^{\infty} C^k$. Thus, $\mathcal{M} \geq 0$ since $C$ is non-negative (which follows from the fact that $\mathcal{P} \succ 0$ and $J_i \geq 0$'s). It follows from the expression for $\psi$ that it is also non-negative vector. This shows that $q = (I - C)^{-1}\psi \geq 0$, which implies $\mu \geq 0$. □

Now we are ready to prove Theorem 2

*Proof of Theorem 2.* It follows from Proposition 3 that under the hypothesis of the Markov chain being ergodic, we have $\rho(\mathcal{D}) < 1$. It then follows from Lemma 1 that the state converges in the mean square sense, which proves the first statement of the theorem. Note that the limiting mean and correlation of the state is also provided by Lemma 1.

We already know from Lemma 1 that $\mu \geq 0$. To prove the last statement of the theorem, that $\mu(u) > 0$ if and only if there is a path between node $u$ and the source node 1 in the union graph, we have to look at the structure of the tall vector $q$ in (11) more carefully, since $q$ completely determines $\mu$. With some abuse of notation, from now on the source node will be referred to as node 1 instead of $v_1$. Note that $\pi \gg 0$ which follows from ergodicity, $\mathcal{P} \gg 0$ by assumption, $B_i$ is a diagonal matrix with positive diagonal entries for every $i$ (follows from its definition), and $w = se_1$, where $s > 0$ and $e_1 = [1, 0, \dots, 0]^T \in \mathbb{R}^n$. It is easy to show now that the $\psi_j = ae_1$ for some $a > 0$. Thus, $\psi = a[{e_1}^T, \dots, {e_1}^T]^T \in \mathbb{R}^{Nn}$. Since $\rho(C) < 1$ (Proposition 3), $\mathcal{M} := (I - C)^{-1} = \sum\limits_{k=0}^{\infty} C^k$. Now, we express the matrix $\mathcal{M}$ in terms of its blocks: $\mathcal{M} = [\mathcal{M}^{(ij)}]$, where $\mathcal{M}^{(ij)}$ are $n \times n$ matrices. Then, $q$ can be rewritten as,

$$q = \begin{bmatrix} q^{(1)} \\ q^{(2)} \\ \vdots \\ q^{(N)} \end{bmatrix} = \begin{bmatrix} \mathcal{M}^{(11)} & \mathcal{M}^{(12)} & \dots & \mathcal{M}^{(1N)} \\ \mathcal{M}^{(21)} & \mathcal{M}^{(22)} & \dots & \mathcal{M}^{(2N)} \\ \vdots & \vdots & \dots & \vdots \\ \mathcal{M}^{(N1)} & \mathcal{M}^{(N2)} & \dots & \mathcal{M}^{(NN)} \end{bmatrix} \begin{bmatrix} ae_1 \\ ae_1 \\ \vdots \\ ae_1 \end{bmatrix} = a[\mathcal{M}^{(ij)} e_1], \quad (a > 0)$$

Therefore, $q^{(i)} = \sum_{j=1}^N \mathcal{M}^{(ij)} e_1 = \sum_{j=1}^N \mathcal{M}^{(ij)}_{:1}$, where the subscript : 1 denotes the first column of the corresponding matrix. Hence, the $u$-th entry of $q^{(i)}$ is $q^{(i)}(u) = \sum_{j=1}^N \mathcal{M}^{(ij)}_{u1}$. Recall that $\mu = \sum_{i=1}^N q^{(i)}$. Therefore $\mu(u) = 0$ if and only if $q^{(i)}(u) = 0$ for $i = 1, \dots, N$, which is also equivalent to $\sum_{i=1}^N \sum_{j=1}^N \mathcal{M}^{(ij)}_{u1} = 0$.

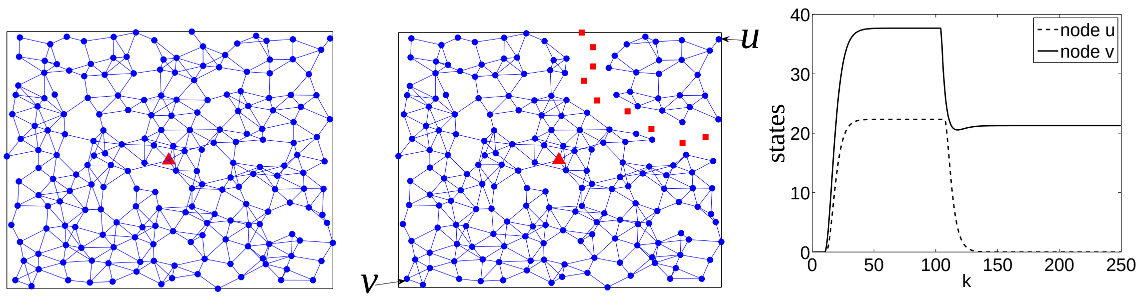
The subsequent discussion requires introducing directed graphs associated with matrices. For every $\ell \times \ell$ matrix $A$, define $\vec{\mathcal{G}}(A) = (\mathcal{V}, \vec{\mathcal{E}})$ be the directed graph corresponding to $A$ as follows: the node set $V$ is the index set $\mathcal{V} = \{1, \ldots, \ell\}$ and the edge set is defined by $(i, j) \in \vec{\mathcal{E}}$ if and only if $A_{i,j} \neq 0$ [17]. It is a standard result in graph theory that the number of walks from a vertex $i$ to vertex $j$ in a directed graph of length $r$ is the $(i, j)$-th element of $A^r$, where $A$ is the adjacency matrix of the graph [18, pp. 165]. Since $\mathcal{M} = \sum\limits_{k=0}^{\infty} C^k$, it follows from the preceding discussion that the $(i, j)$-th entry of $\mathcal{M}$ is positive if and only if there exists a path from the vertex $i$ to vertex $j$ in the directed graph $\vec{\mathcal{G}}(C)$. Note that the graph $\vec{\mathcal{G}}(C)$ contains $Nn$ nodes. We can group $Nn$ nodes into $N$ clusters such that each cluster, containing $n$ nodes, can be thought of as copies of the $n$ nodes in the sensor and robot network. To prevent confusion between the vertices in $\vec{\mathcal{G}}(C)$ and node set $\mathcal{V}$ of the original network, we use $v^{(i)}$ to denote a node in the graph $\vec{\mathcal{G}}(C)$ that is the $i$-th copy of the node $v$ in $\mathcal{V}$, where $i = 1, \ldots, N$.

Therefore $\sum_{i=1}^{N} \sum_{j=1}^{N} \mathcal{M}_{u1}^{(ij)} = 0$ is equivalent to there being no directed path from any of the $u$'s copies $(u^{(i)}, i = 1, \ldots, N)$ to any of 1's copies $(1^{(i)}, i = 1, \ldots, N)$ in the directed graph $\vec{\mathcal{G}}(C)$. Otherwise, $q(u) > 0$. Since existence of an edge from $i$ to $j$ in $\vec{\mathcal{G}}(A)$ only depends on whether the $i, j$-th entry of $A$ is non-zero, and does not depend on the specific value of the entry, it is convenient to define $\overline{A}$ be a matrix associated with the matrix $A$, such that $\overline{A}_{ij} = 1$ if $A_{ij} \neq 0$ and $\overline{A}_{ij} = 0$ if $A_{ij} = 0$. Since $\mathcal{P} \gg 0$, we have

$$\overline{C} = [\overline{p_{ij} J_j}] = \begin{bmatrix} \overline{J_1} & \overline{J_2} & \cdots & \overline{J_N} \\ \overline{J_1} & \overline{J_2} & \cdots & \overline{J_N} \\ \vdots & \vdots & \cdots & \vdots \\ \overline{J_1} & \overline{J_2} & \cdots & \overline{J_N} \end{bmatrix}_{Nn}.$$

It can be seen in a straightforward manner upon examining the matrix $\overline{C}$ that if there is an edge between nodes $u$ and $v$ in the $i$-th graph $\mathcal{G}_i$, i.e., $(u, v) \in \mathcal{E}^{(i)}$, then $(u^{(j)}, v^{(i)})$ for all $j = 1, \ldots, N$, and $(v^{(j)}, u^{(i)})$ for all $j = 1, \ldots, N$, i.e., there are edges in $\vec{\mathcal{G}}(C)$ from all copies of $u$ to $v^{(i)}$, the $i$-th copy of $v$, and from all copies of $v$ to $u^{(i)}$, the $i$-th copy of $u$.

Now we will show that if an arbitrary node $u$ is connected to 1 in the union graph $\cup_{i=1}^{N} \mathcal{G}_i$, then there is a path from a copy of $u$ to a copy of 1 in the directed graph $\vec{\mathcal{G}}(C)$, otherwise not. To see that this is the case, we first take an example: consider a path of length 2 from $u$ to 1 in the union graph that involves two edges in two distinct graphs: $(u, v) \in \mathcal{E}^{(2)}$ and $(v, 1) \in \mathcal{E}^{(1)}$. From the preceding discussion, we have that $(v, 1) \in \mathcal{E}^{(1)} \Rightarrow (v^{(1)}, 1^{(1)}), (v^{(2)}, 1^{(1)}) \in \vec{\mathcal{E}}(C)$, and $(u, v) \in \mathcal{E}^{(2)} \Rightarrow (u^{(1)}, v^{(2)}), (u^{(2)}, v^{(2)}) \in \vec{\mathcal{E}}(C)$. Thus a path from a copy of $u$ to a copy of 1 in $\vec{\mathcal{G}}(C)$ is $p = \{(u^{(1)}, v^{(2)}), (v^{(2)}, 1^{(1)})\}$. This argument works as long as there is a path from $u$ to 1 in the union graph, irrespective of how long the path is. This shows that $u$ is connected to 1 in the union graph, then there is a path from at least one of its copies to one of 1's copies in the directed graph $\vec{\mathcal{G}}(C)$, which means $q(u) > 0$. If, however, $u$ is not connected to 1 in the union graph, we can show that there is no path from any of $u$'s copies to any of 1's copies. This can be shown by considering the set of all nodes that do not have paths to 1 in the union graph and the set of nodes that do separately; see [19] for details. This concludes the proof of the last statement of the theorem. $\square$



(a) $\mathcal{G}$ before cut (b) $\mathcal{G}(k)$ for $k > 100$ (c) $x_u(k)$ and $x_v(k)$ vs. $k$

Figure 4: (a)-(b): A sensor network with 200 static nodes, shown before and after a cut occurs due to the failure of the nodes shown as red squares. The cut occurs at $k = 100$. (c): The states of two nodes $u$ and $v$ as a function of iteration number. The source node is at the center (triangle), and the source strength is chosen as $s = 5 \times 10^4$.

## 4. Simulation Tests

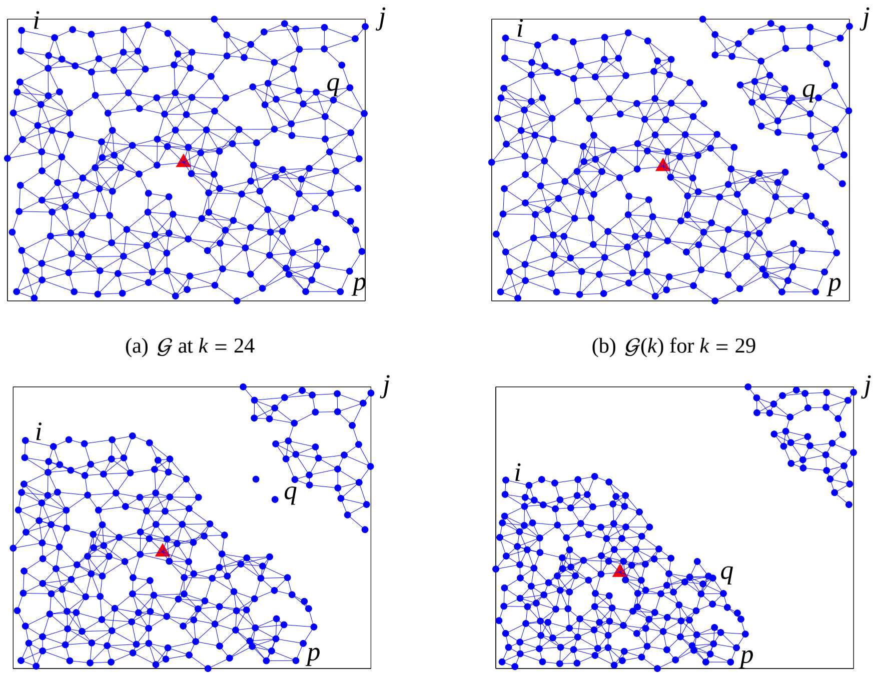
The DSSD algorithm was tested in a MATLAB$^{\text{TM}}$ simulation for a network consisting of 200 agents initially deployed in a unit square at random. Two agents can only establish direct communication if their Euclidean distance is less than 0.11. The source strength and cut detection threshold was $s = 5 \times 10^5$ and $\epsilon = 10^{-2}$, respectively. Since there is no existing prior work on the problem of detecting separation in mobile networks that can operate without multi-hop routing, we do not provide simulation comparison with existing algorithms. Note that the solutions proposed in [9, 10] require routing between the nodes and the base station, which is challenging in sensor and robotic networks in which the topology can change with time quickly.

### *4.1. Performance of DSSD in a static network*

The first set of simulations is conducted with 200 static nodes (see Figure 4(a)). The center node (symbolized by a triangle) is the source node. Simulations are run in a synchronous manner and a neighbor is removed from the list of neighbors of a node the first time it failed to receive messages from that neighbor. At $k = 100$ the nodes shown as red squares in Figure 4(b) fail, leading to a cut in the network. Figure 4(c-d) show the time evolution of the states (calculated using (1)) of the four nodes $u$, $v$, $w$, and $z$. Node $v$ is the only one among the four that is separated from the source after the cut occurs. Initially, the states of every node increase from 0 and then settle down to their steady state value. After the cut occurs, the state of node $v$ decreases towards 0. When the state of node $v$ decreases below the preset threshold $\epsilon$, it declares itself cut from the source. This occurs at $k = 133$, thus the delay between the occurrence of the cut and its detection by $v$ is 33 time-steps.

### *4.2. Performance of DSSD in a mobile network*

Figures 5(a-d) show four snapshots of a communication network of 200 mobile agents. The agents are divided into two groups, though there is no clear spatial separation between the two groups initially. The position of agent $u$,



(a) $\mathcal{G}$ at $k = 24$ (b) $\mathcal{G}(k)$ for $k = 29$

(c) $\mathcal{G}(k)$ for $k = 56$ (d) $\mathcal{G}(k)$ for $k = 99$

Figure 5: Four snapshots of a network of 200 mobile agents.

denoted by $Z_u$, is updated according to:

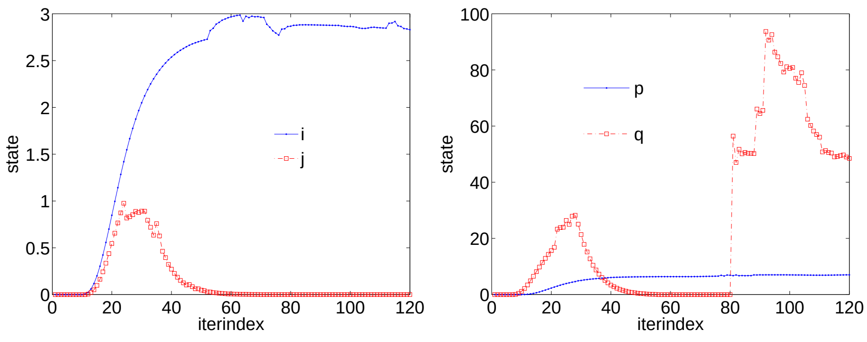
$$Z_u(k+1) = Z_u(k) + \begin{bmatrix} \delta Z_{ux}(k) \\ \delta Z_{uy}(k) \end{bmatrix} \tag{12}$$

where $\delta Z_{ux}(k)$, $\delta Z_{uy}(k)$, for every $u$ and $k$, are independent random numbers. For agents in the first group, both $\delta Z_{ux}$ and $\delta Z_{uy}$ are normally distributed with mean 0.003 and variance 0.0003. For the second group, $\delta Z_{ux}$, $\delta Z_{uy}$ are normally distributed with mean $-0.003$ and variance 0.0003. The motion of the agents results in the network composed of two disjoint components at $k = 28$, four components at $k = 56$, and then again two components at $k = 80$.

The evolution of the states of four agents $i$, $j$, $p$, and $q$ are shown in Figure 6(a-b). The loss of connectivity of agent $q$ from the source occurs at $k = 28$ and is detected at $k = 55$. Connectivity to the source is regained at $k = 80$ and is detected at $k = 81$ (when the states became greater than $\epsilon$). These simulations provide evidence that the algorithm is indeed effective in detecting disconnections and re-connections, irrespective of whether the network is made up of static or mobile agents.

## 5. System Implementation and Experimental Evaluation

In this section we describe the implementation, deployment and performance evaluation of a separation detection system for robotic sensor networks based on the DSSD algorithm. We implemented the system, using the nesC



(a) states of node $i$ and $j$ (b) states of node $p$ and $q$

Figure 6: The states of four mobile nodes $i$, $j$, $p$, $q$ (as a function of time) in the network shown in Figure 5.

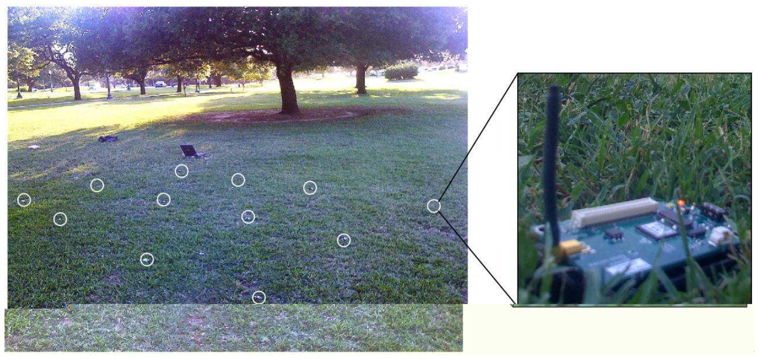
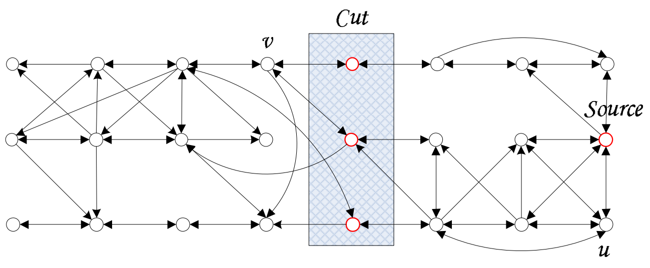


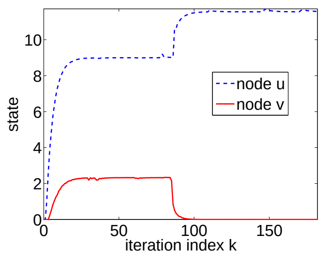
Figure 7: Partial view of the 24 node outdoor system deployment.

language, on Berkeley motes [20] running the TinyOS operating system [21]. The code uses 16KB of program memory and 719B of RAM. The separation detection system executes in two phases: Reliable Neighbor Discovery, and the DSSD algorithm.

In the Reliable Neighbor Discovery Phase each node broadcasts a set of beacons in a small, fixed, time interval. Upon receiving a beacon from node $v_i$, a node updates the number of beacons received from node $v_i$. Next, an iteration of the DSSD algorithm executes. To determine whether a communication link is established, each node first computes for each of its neighbors the Packet Reception Ratio (PRR), defined as the ratio of the number of successfully received beacons received from, to the total number of beacons sent by, a neighbor. A neighbor is deemed reliable if the PRR $> 0.8$. After receiving state information from neighbors, a node updates its state according to Equation (1) and broadcasts its new state. When broadcast from a neighbor is not received for 2 iterations, the last reported state of the neighbor is used for calculating the state. A neighbor from which broadcast is not received for 4 iterations is permanently removed from the neighbor table. The state is stored in the 512KB on-board flash memory at each iteration (for a total of about 1.6KB for 200 iterations) for post-deployment analysis. In order to monitor connectivity information each node broadcasts its neighbor table along with the state.

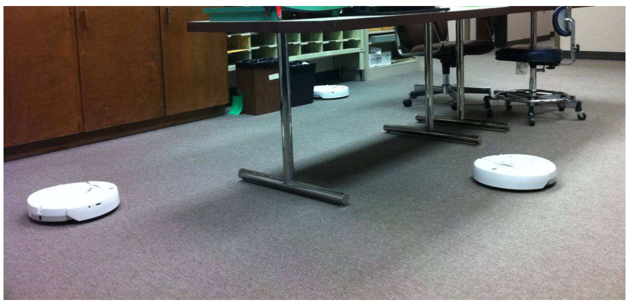


(a) The graph

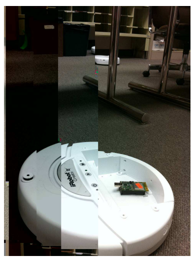


(b) State histories

Figure 8: (a) The network topology during the outdoor deployment. (b) The states of nodes $u$ and $v$ (as labeled in (a)), which are connected and disconnected, respectively, from the source after the cut has occurred.



(a)



(b)

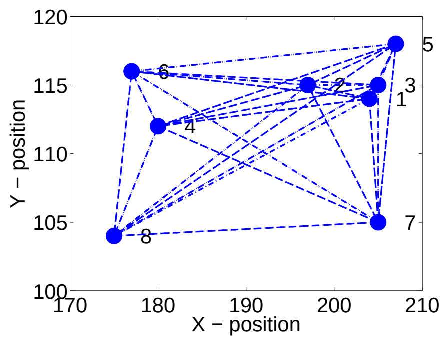
Figure 9: (a) Test set-up for mobile network experiments. The human agents are not shown. (b) Mobile node consisting of a Berkeley mote on a Roomba robot.

To ensure a lock-step execution of the algorithm, all nodes are started at approximately the same time. For this, a mote acting as a base station, connected to a laptop, broadcasts a “system start” message, which is resent by each sensor node at most once. The base station is also used for monitoring the execution of the algorithm and monitoring the inter-mote communication.

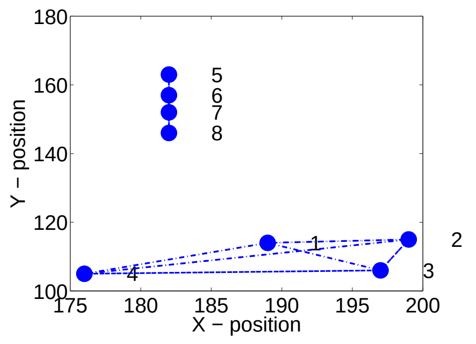
## *5.1. Experimental Performance Evaluation in Static Network*

For evaluating the performance of our separation detection system in static networks, we deployed a network of 24 motes in a 13×5m$^2$ outdoor field at Texas A&M University. Because the motes were positioned on the ground the radio range was reduced considerably with a one-hop distance of about 1.5m. The network connectivity is depicted in Figure 8(a). A partial view of the outdoor deployment is shown in Figure 7.

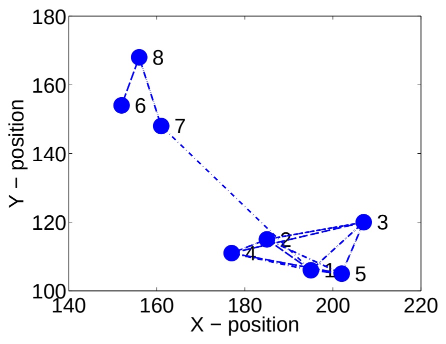
In our deployment, the source strength was specified as $s = 100$, the iteration length was 5sec (this value could be reduced easily to as small as 200 msec) and the cut detection threshold was $\epsilon = 0.01$. Experimental results for two of the sensor nodes deployed are shown in Figure 8. After about 30 iterations the states of all nodes converged. At iteration $k = 83$ a cut is created by turning off motes inside the rectangle labeled “Cut” in Figure 8(a). Figures



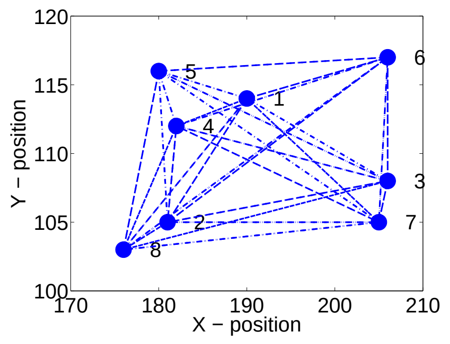
(a) $\mathcal{G}(k)$ at $k = 110$



(b) $\mathcal{G}(k)$ for $k = 150$



(c) $\mathcal{G}(k)$ for $k = 175$



(d) $\mathcal{G}(k)$ for $k = 250$

Figure 10: Four snapshots of a network of 8 mobile agents and the state evolution for them resulting from the DSSD algorithm. The dashed lines represent communication links.

8(b) and 8(c) show the states for nodes $u$ and $v$, as depicted in Figure 8(a), which were connected and disconnected, respectively, from the source node after the cut. The evolution of their states follows the aforementioned experimental scenario. Node $v$ declares itself cut from the source at $k = 100$, since its state falls below the threshold 0.01 at that time.

## *5.2. Experimental Performance Evaluation in Mobile Network*

For evaluating the performance of our separation detection system in mobile networks we deployed 8 sensor nodes in an indoor environment, with 4 of the nodes residing on Roomba robots and 4 on human subjects. The scenario we emulated was that of a robotic-assisted emergency response team. Figure 9 shows part of the test set-up with the mobile nodes.

The network topologies as well as the locations of the nodes at a few time instants are shown in Figure 10. As we can see from the figure, the topology of the network varied greatly over time due to the mobility of the nodes.

Figure 11 shows the time-traces of the node states during the experiment. The network is connected until time $k = 120$, and the states of all nodes converges to positive numbers; see Figure 11(a). This is consistent with the prediction of theorem 2. At approximately iteration $k = 120$, four of the nodes (nodes 5 through 8), carried by human

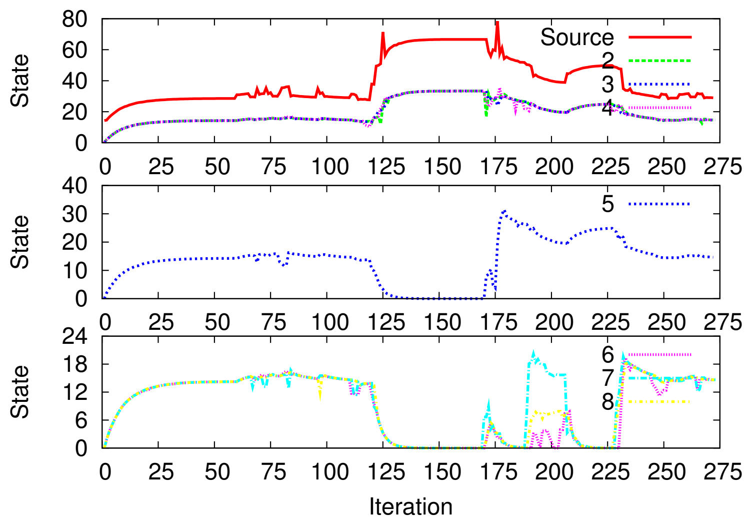


Figure 11: The states of nodes 1 through 8 in the mobile network experiment with Roomba robots and human agents.

subjects, are disconnected from the rest of the network, and in particular, from the source node 1. A sample network topology during the time interval $k = 120$ to $k = 170$ is shown in Figure 10(b). As we can see from Figure 11(b-c), the states of the disconnected nodes 5 through 8 converge to zero. The nodes 5, 6, 7, 8 detect that they are separated from the source, at times $k = 145, 145, 143, 143$ respectively, when their states become lower than the threshold $\epsilon = 0.01$. At approximately iteration $k = 170$, node 5 joins back the sub-network formed by nodes 1-4. As a result of node 5 moving, node 7 becomes a bridge between the two sub-networks. Hence, after iteration $k = 170$, the states of nodes 6, 7 and 8 become positive (hence, a fully connected network). However, this re-connection is temporary, and nodes 6 through 8 again become disconnected from the source after some time, which is seen in their states. Another temporary connection occurs between the set of nodes 6-8 and the set of nodes 1-5, during the time interval $k = 180$ through $k = 210$, followed by a separation. Finally, after iteration $k = 260$, the network becomes connected again, as shown in Figure 10(d). As a result, the states of all the nodes become positive after time $k = 225$, and they detect their re-connections to the source.

## 6. Conclusions

In this paper we introduced the Distributed Source Separation Detection (DSSD) algorithm to detect network separation in robotic and sensor networks. Simulations and hardware experiments demonstrated the efficacy of the algorithm. DSSD requires communication only between neighbors, which avoids the need for routing, making it particularly suitable for mobile networks. The algorithm is distributed, doesn't require time synchronization among nodes, and the computations involved are simple. The DSSD algorithm is applicable to a heterogeneous network of static as well as mobile nodes, with varying levels of resources, precisely the kind envisioned for robotic and sensor networks.

## References

[1] G. McKee, P. Schenker, Proceedings of SPIE 4196 (2003) 197.

[2] M. A. Hsieh, L. Chaimowicz, A. Cowley, B. Grocholsky, J. Keller, V. Kumar, C. J. Taylor, Y. Endo, R. Arkin, B. Jung, D. F. Wolf, G. S. Sukhatme, D. MacKenzie, Journal of Field Robotics 24 (2007) 991–1014.

[3] V. Kumar, D. Rus, S. Singh, IEEE Pervasive Computing 3 (2004) 24–33.

[4] C. Perkins, E. Belding-Royer, S. Das, in: IETF RFC, RFC Editor, United States, 2003.

[5] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, D. Rubenstein, in: Tenth International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS-X).

[6] N. Shrivastava, S. Suri, C. D. Tóth, ACM Trans. Sen. Netw. 4 (2008) 1–25.

[7] J. Kleinberg, Internet Mathematics 1 (2003) 37–56.

[8] P. Barooah, in: 47th IEEE Conference on Decision and Control, pp. 1097 – 1102.

[9] M. Hauspie, J. Carle, D. Simplot, in: 2nd Mediterranean Workshop on Ad-Hoc Networks, pp. 25–27.

[10] H. Ritter, R. Winter, J. Schiller, T. Zippan, in: NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, ACM, New York, NY, USA, 2004, pp. 169–169.

[11] M. A. Hsieh, A. Cowley, V. Kumar, C. J. Taylor, Journal of Robotic Systems 25 (2007) 111 – 131. Special Issue on Search and Rescue Robots.

[12] R. Diestel, Graph theory, Springer New York, 2000.

[13] O. Costa, M. Fragoso, R. Marques, Discrete-Time Markov Jump Linear Systems, Probability and its Applications, Springer, 2004.

[14] M.-Q. Chen, X. Li, Linear Algebra and its Applications (2004).

[15] A. N. Langville, W. J. Stewart, Journal of Computational and Applied Mathematics 167 (2004) 429–447.

[16] A. Berman, R. J. Plemmons, Nonnegative Matrices in the Mathematical Sciences, Computer Science and Applied Mathematics, Academic Press, 1979.

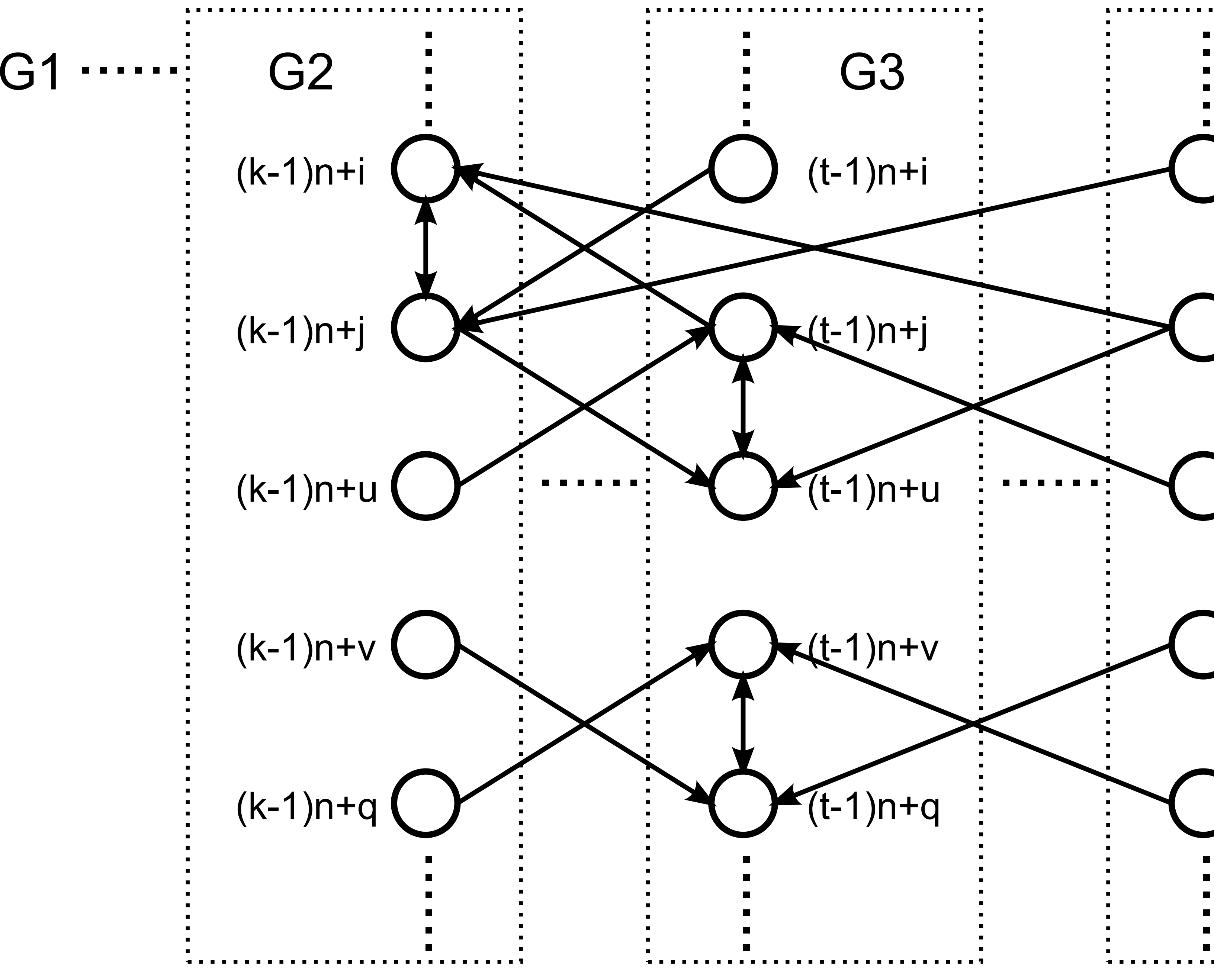
[17] C. D. Meyer, Matrix Analysis and Applied Linear Algebra, SIAM: Society for Industrial and Applied Mathematics, 2001.

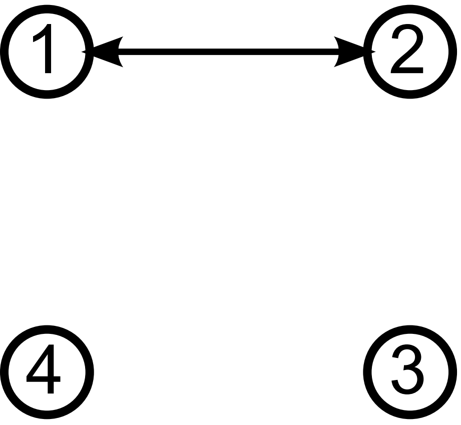
[18] C. Godsil, G. Royle, Algebraic Graph Theory, Graduate Texts in Mathematics, Springer, 2001.

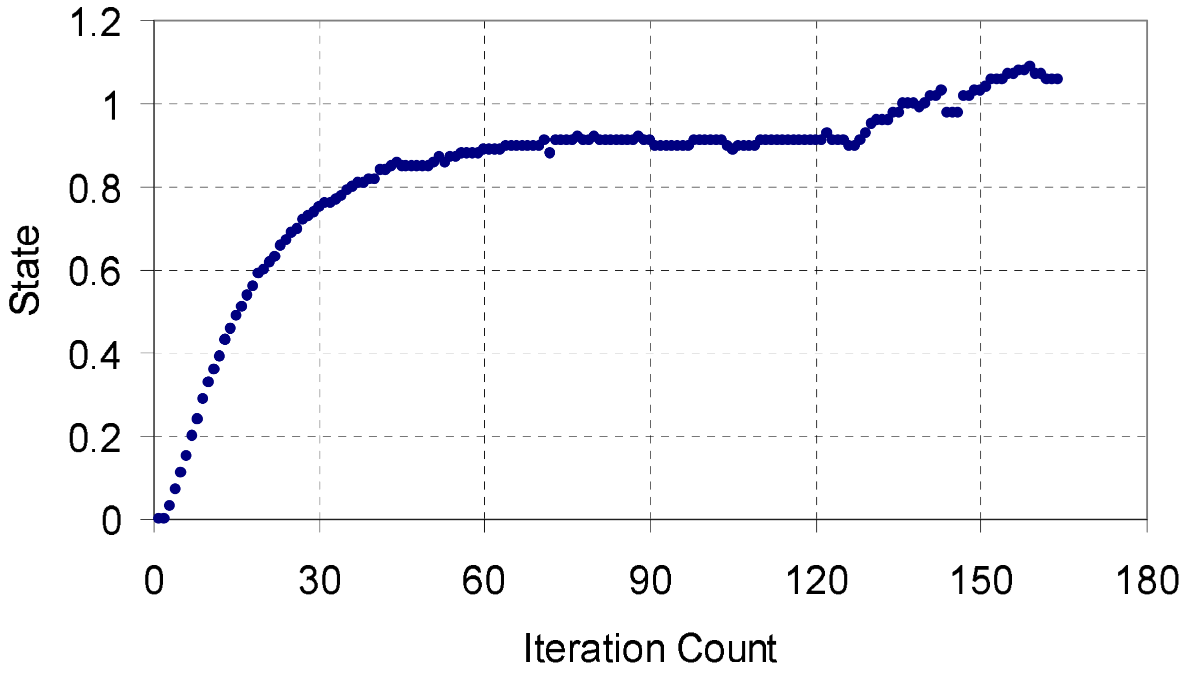
[19] C. Liao, H. Chenji, P. Barooah, R. Stoleru, T. Kalmár-Nagy, Detecting Separation in Robotic and Sensor Networks, Technical Report, University of Florida, 2011. Http://plaza.ufl.edu/cdliao/.

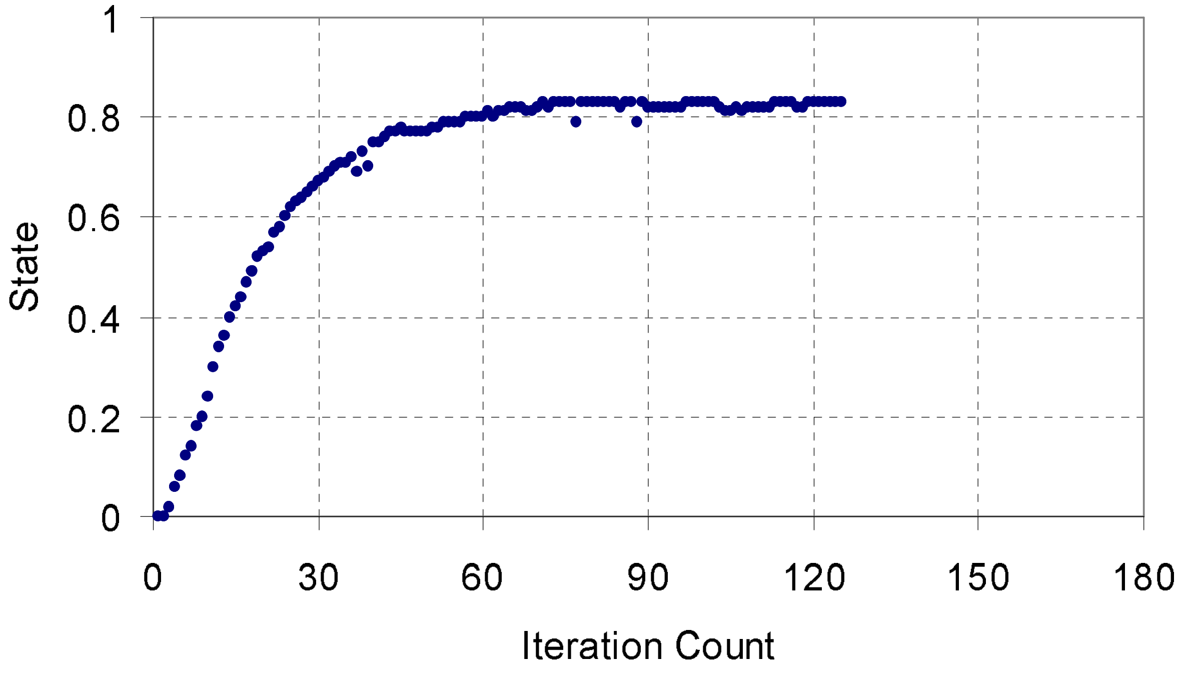
[20] C. Inc., http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAZ_Datasheet.pdf, 2004.

[21] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, in: ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, volume 34, ACM Press, 2000, pp. 93–104.

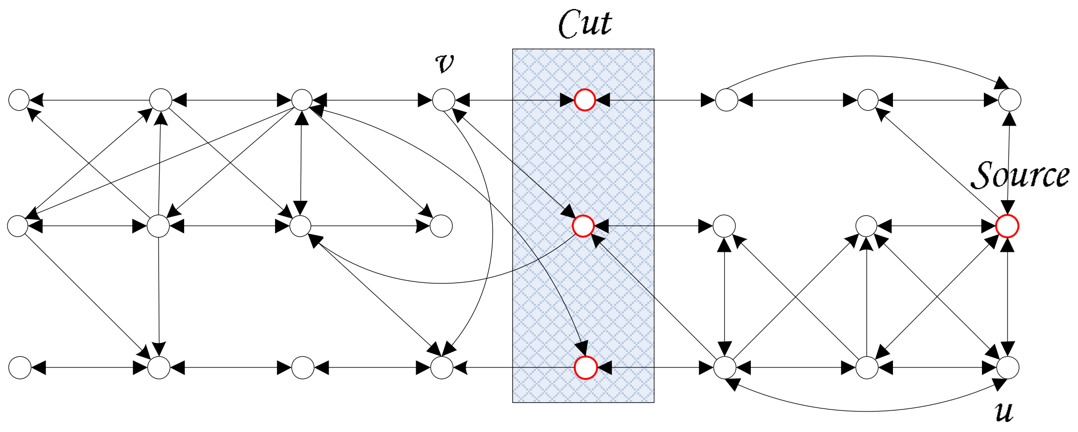

G1
G2
G3
(k-1)n+i
(k-1)n+j
(k-1)n+u
(k-1)n+v
(k-1)n+q
(t-1)n+i
(t-1)n+j
(t-1)n+u
(t-1)n+v
(t-1)n+q

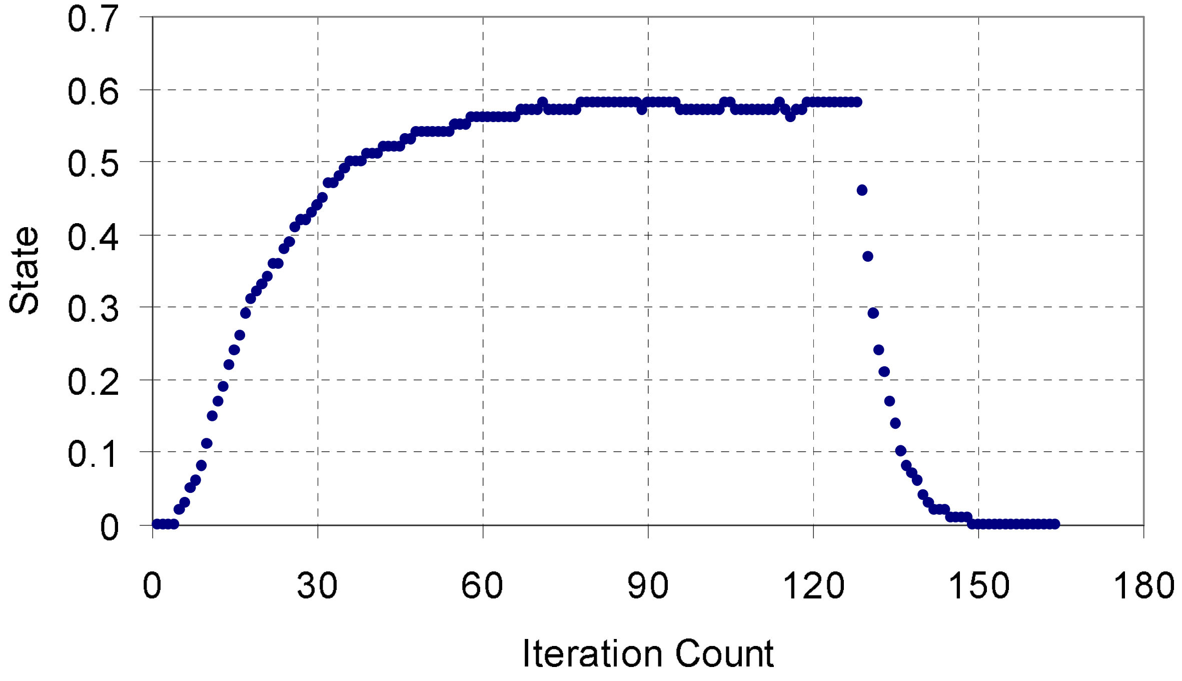

1
2
4
3

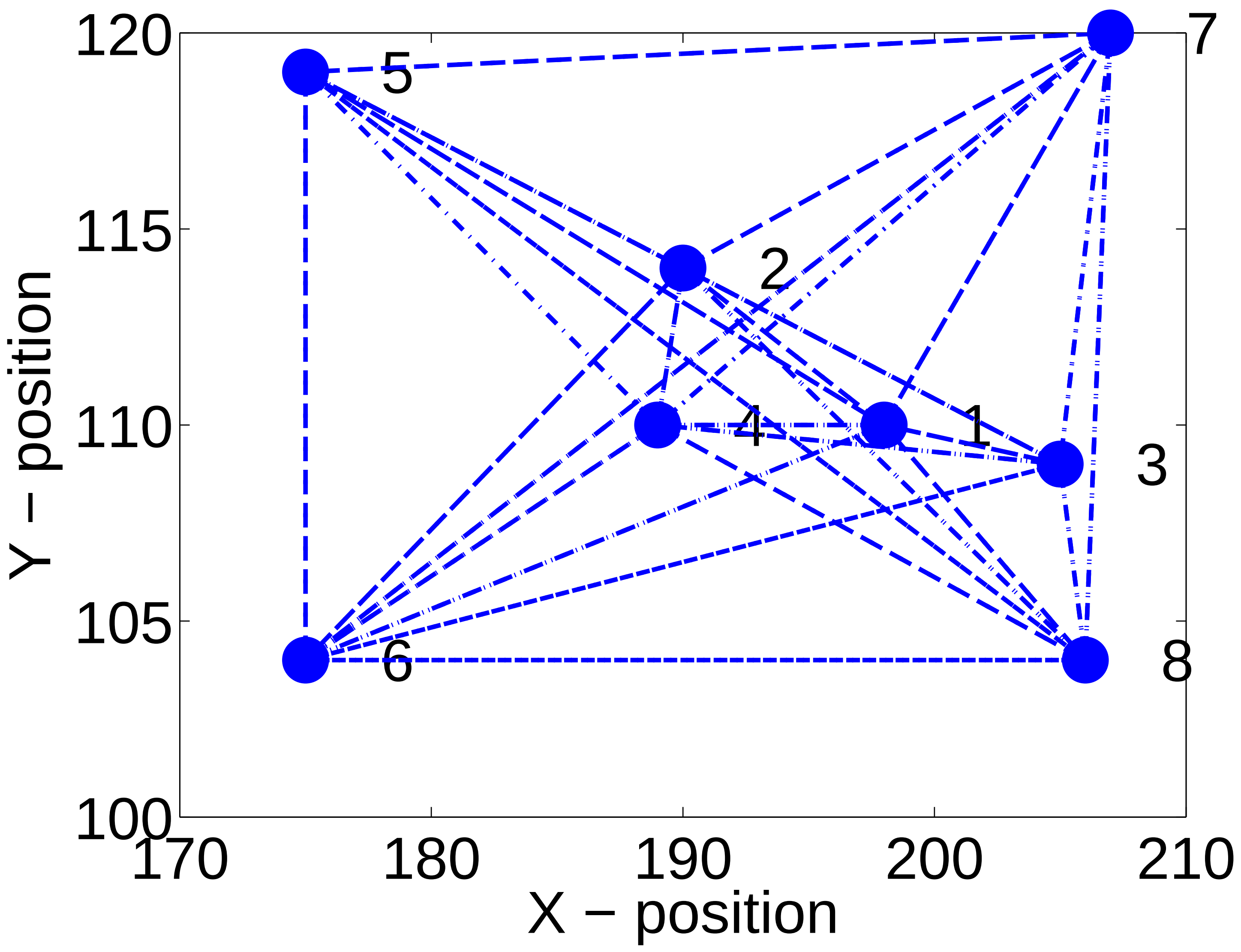

1.2
1
0.8
0.6
0.4
0.2
0
State
0
30
60
90
120
150
180
Iteration Count



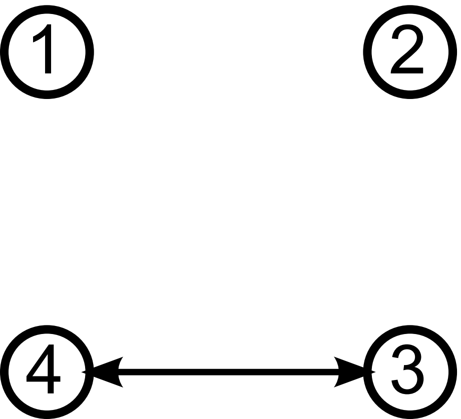
1
0.8
0.6
0.4
0.2
0
State
0
30
60
90
120
150
180
Iteration Count

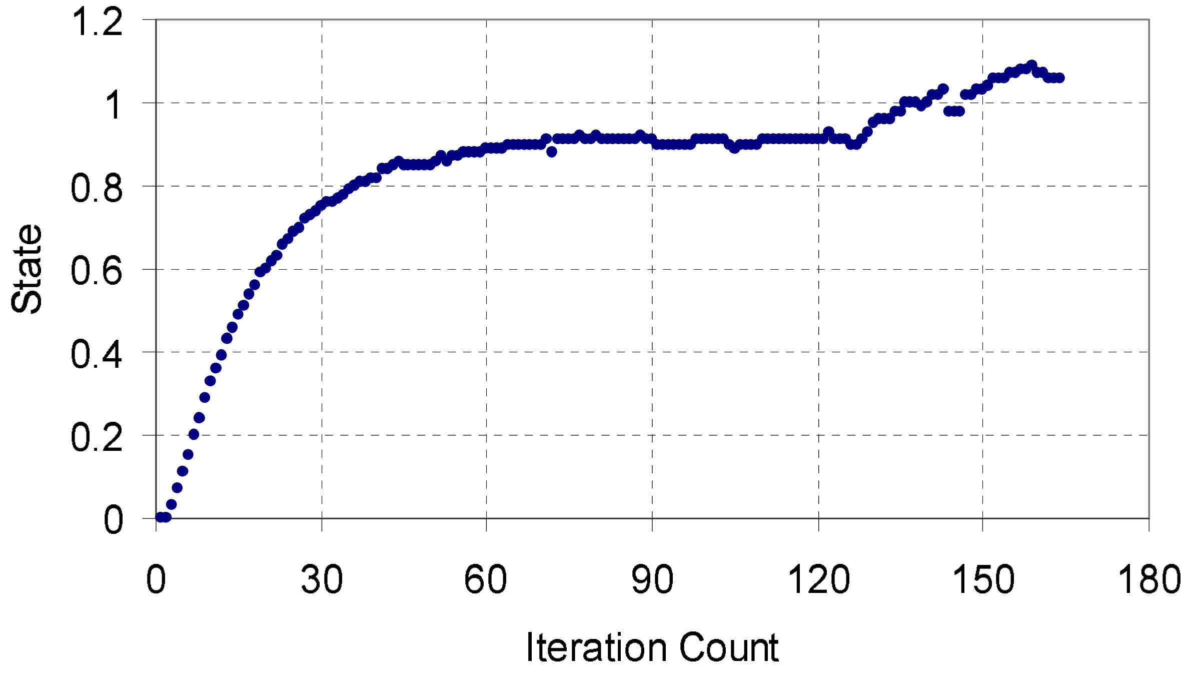

Cut
v
Source
u


State
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
30
60
90
120
150
180
Iteration Count

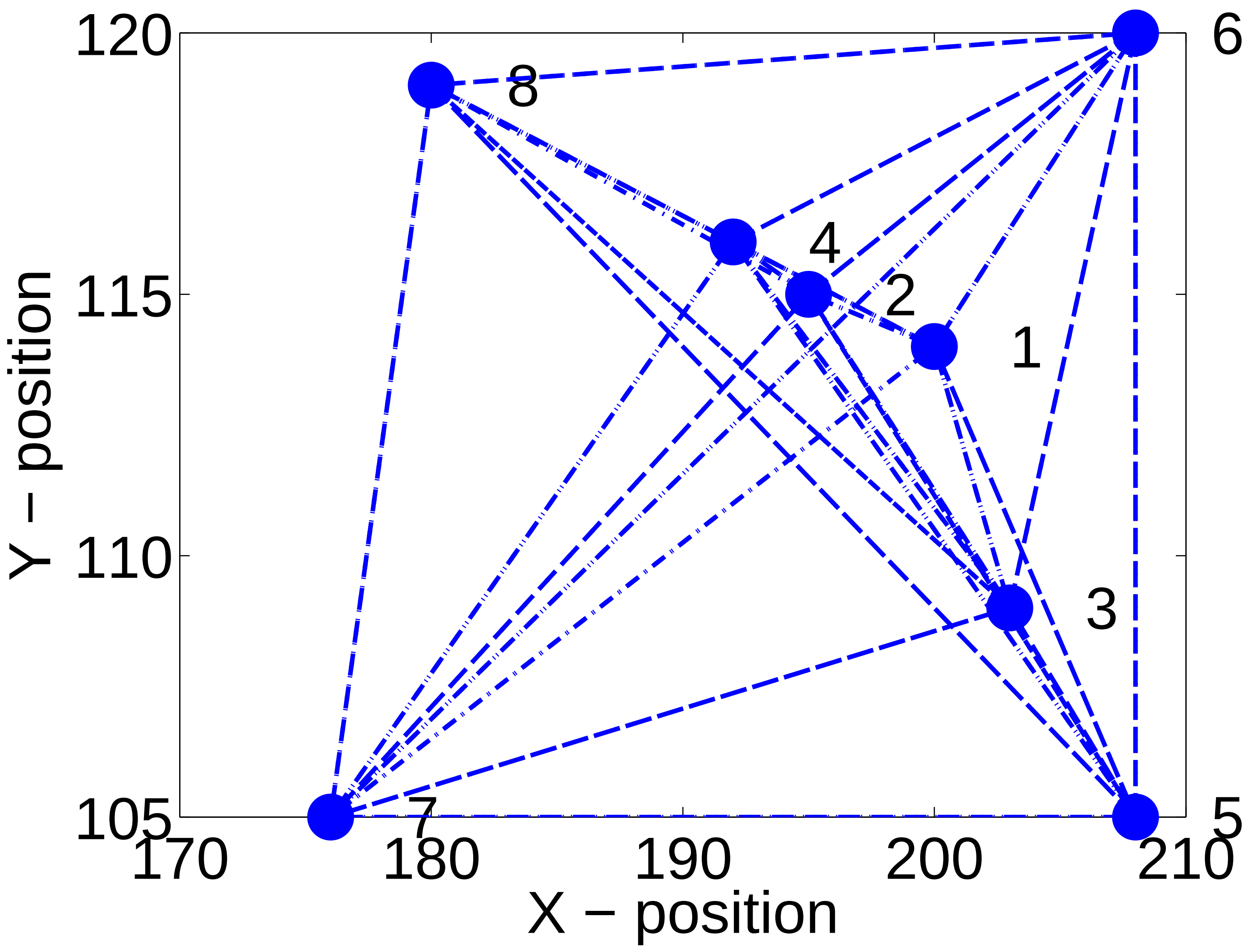


120
115
110
105
100
170
180
190
200
210
Y − position
X − position
1
2
3
4
5
6
7
8

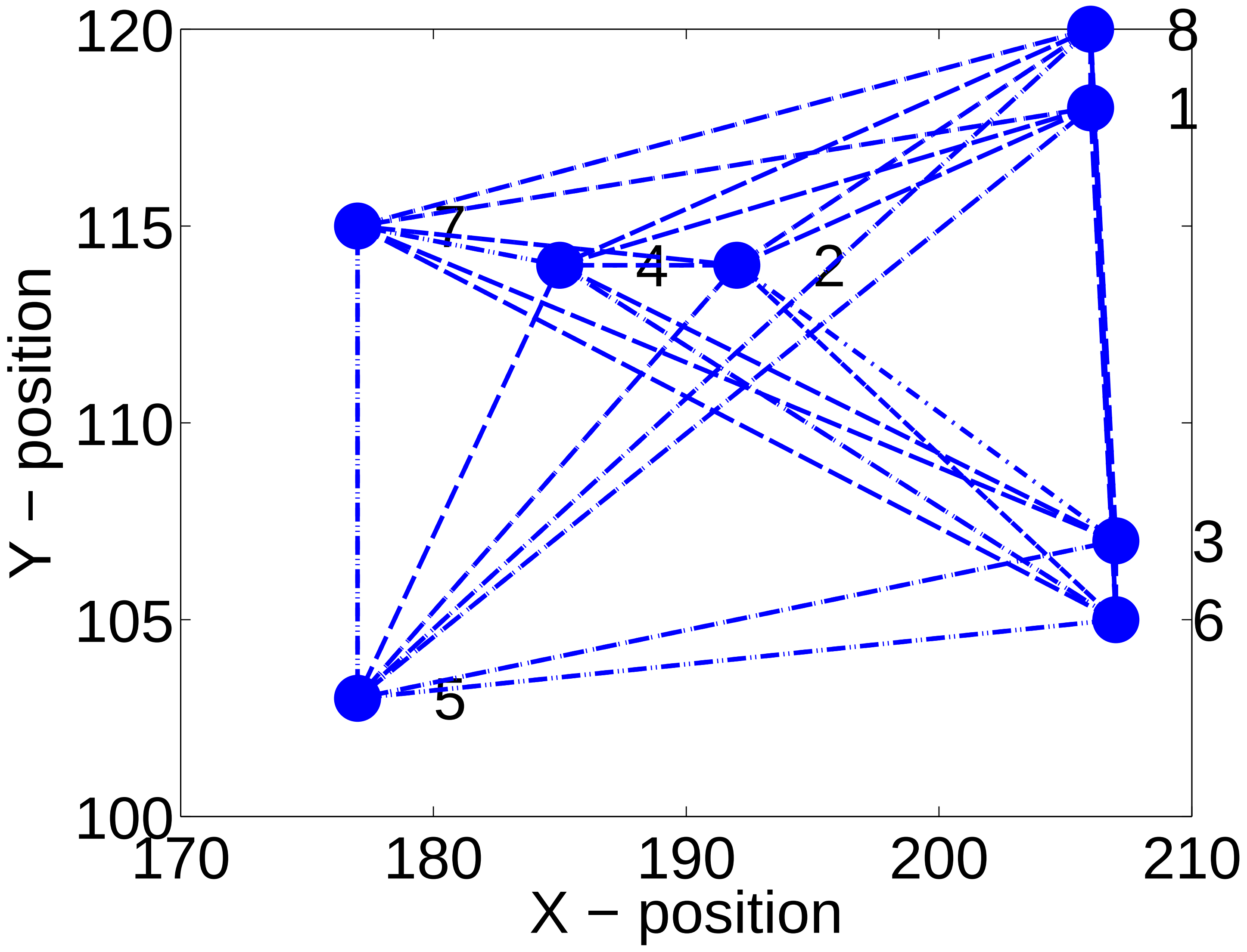

1
2
4
3



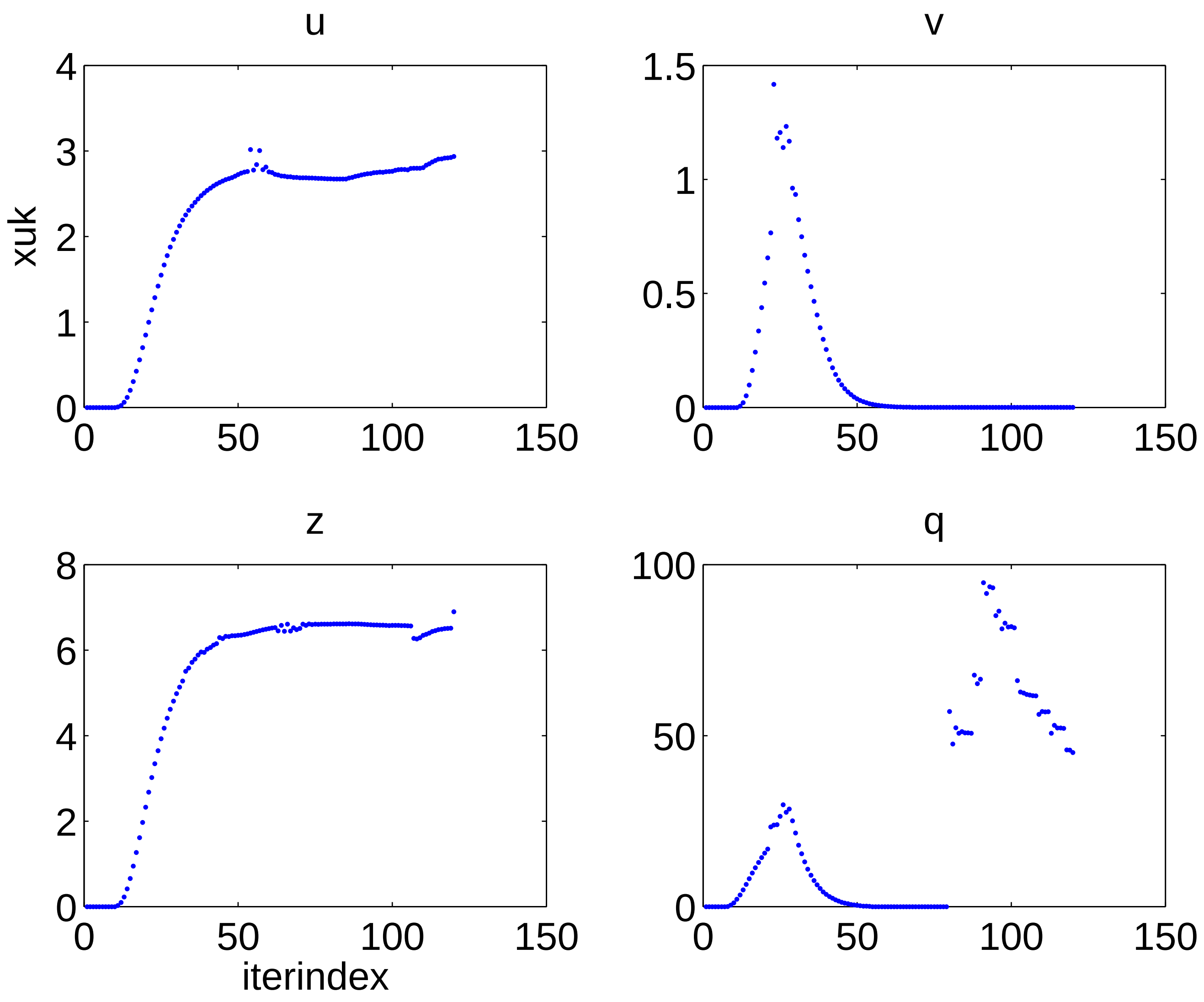
1.2
1
0.8
0.6
0.4
0.2
0
State
0
30
60
90
120
150
180
Iteration Count

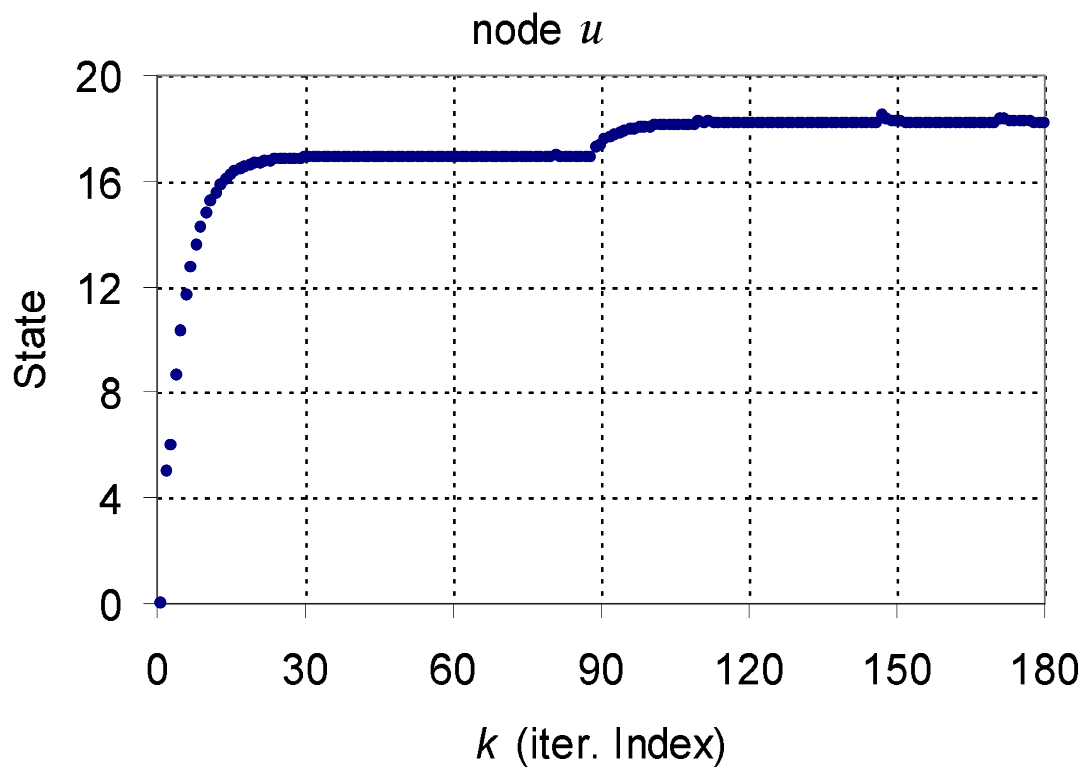


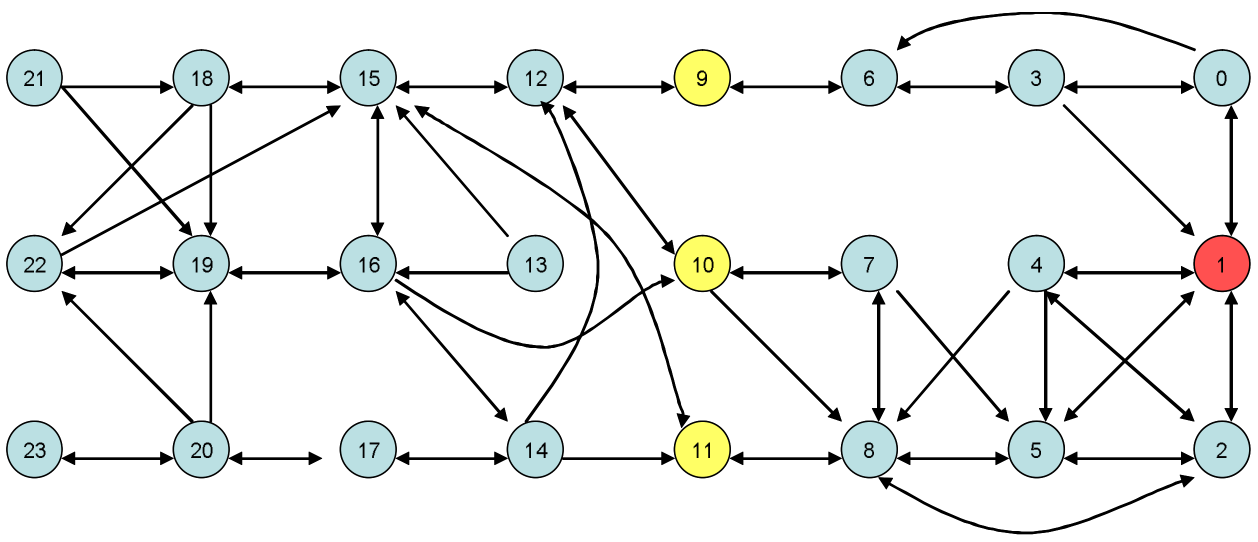
120
115
110
105
170
180
190
200
210
Y − position
X − position
1
2
3
4
5
6
7
8

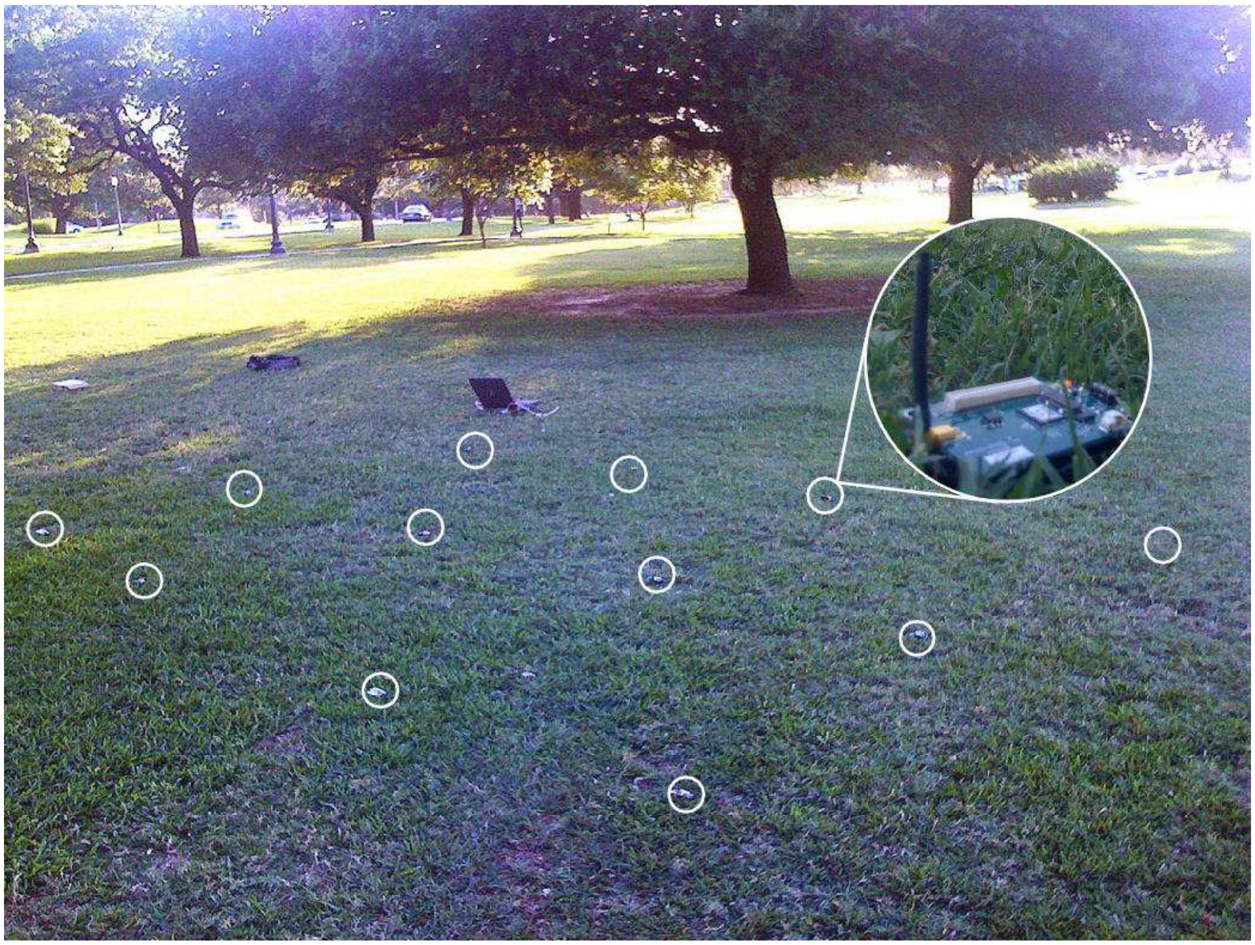


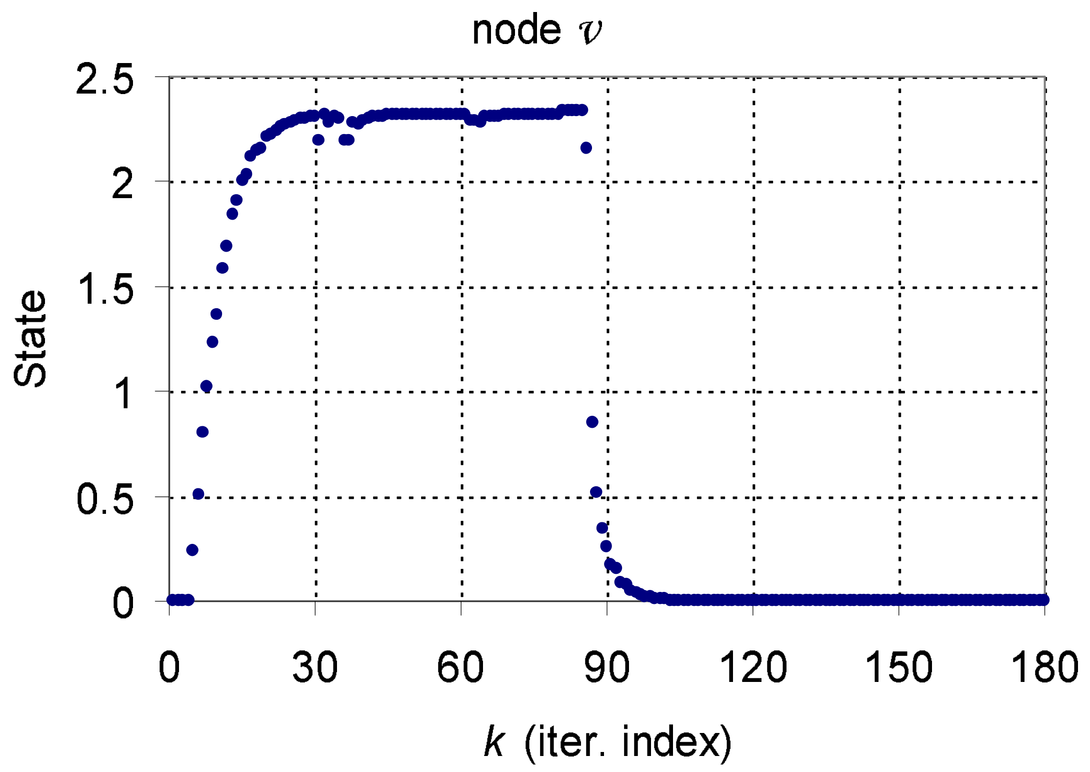
120
115
110
105
100
170
180
190
200
210
Y − position
X − position
8
1
7
4
2
3
6
5

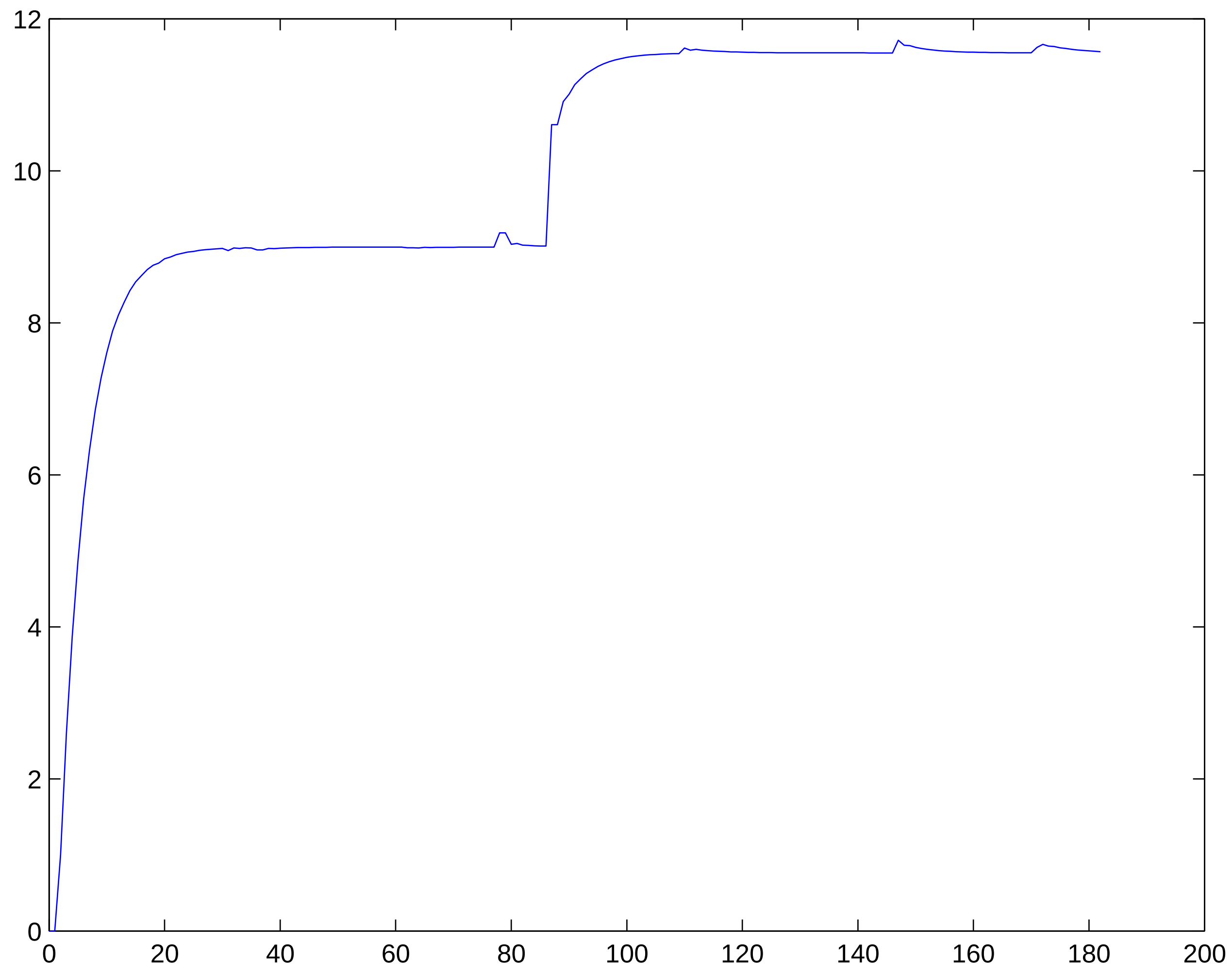

u
v
z
q
xuk
iterindex
0
1
2
3
4
0.5
1.5
6
8
50
100
150


node $u$
20
16
12
8
4
0
State
0
30
60
90
120
150
180
$k$ (iter. Index)

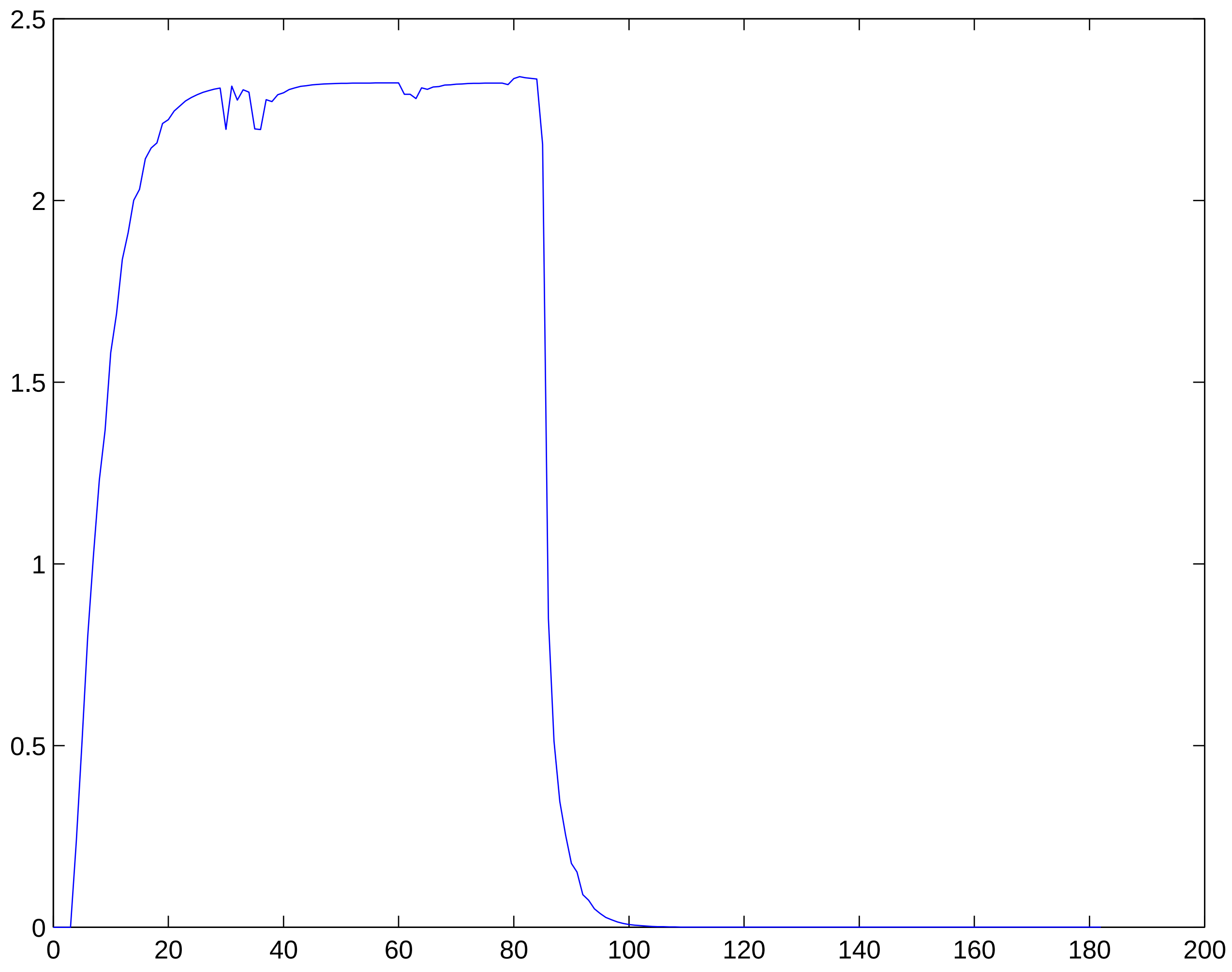

21
18
15
12
9
6
3
0
22
19
16
13
10
7
4
1
23
20
17
14
11
8
5
2



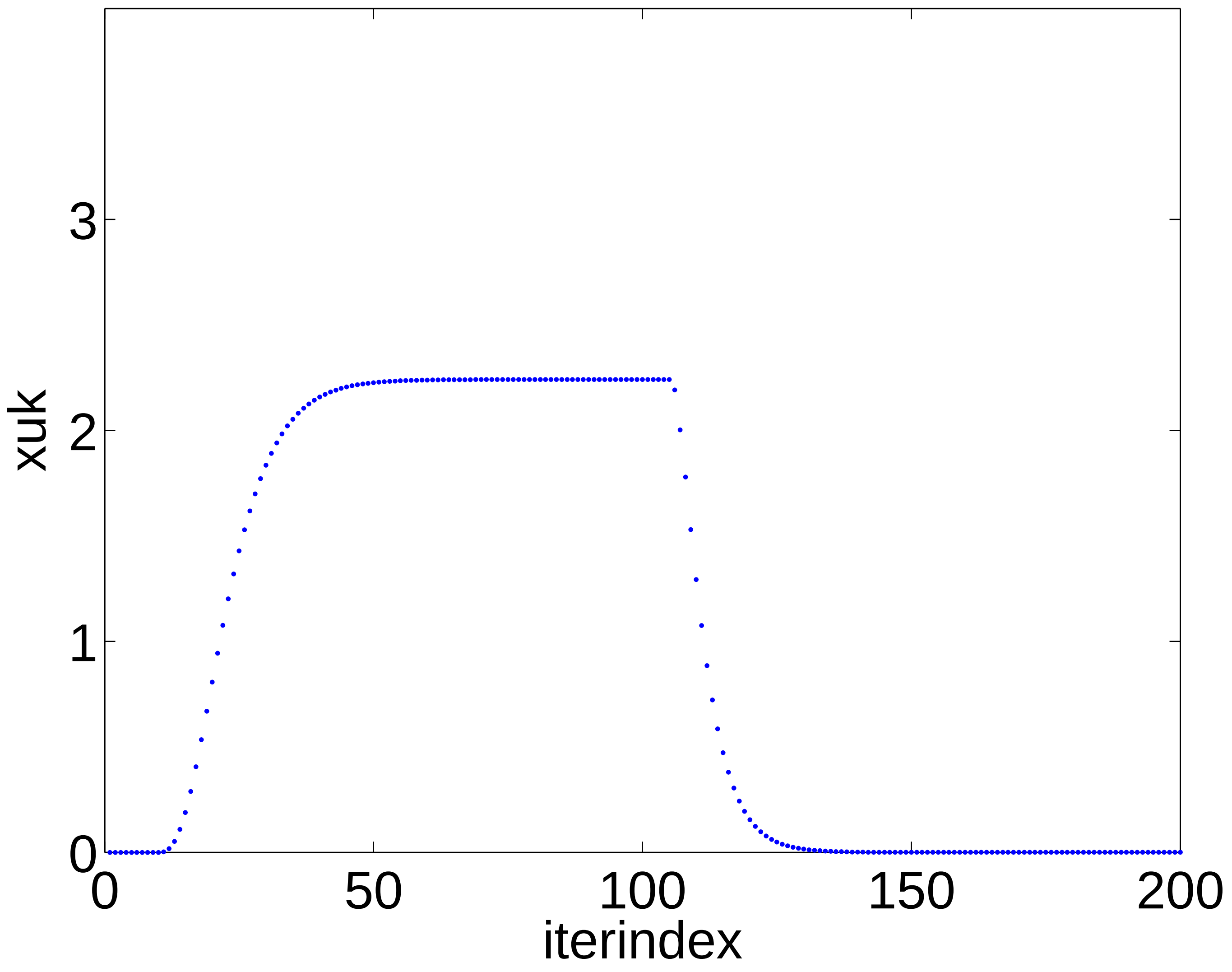

node $v$
State
2.5
2
1.5
1
0.5
0
0
30
60
90
120
150
180
$k$ (iter. index)

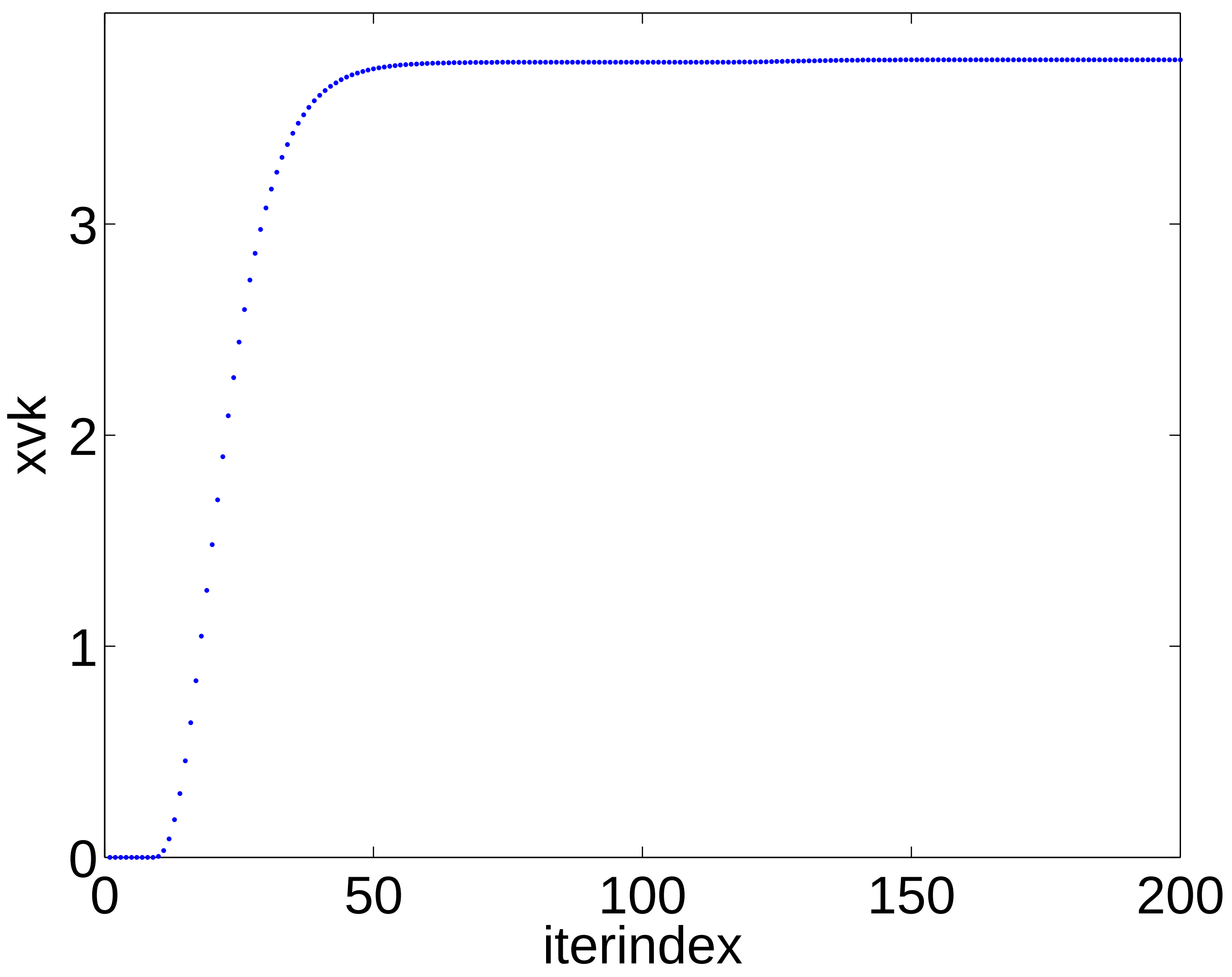


0
2
4
6
8
10
12
0
20
40
60
80
100
120
140
160
180
200

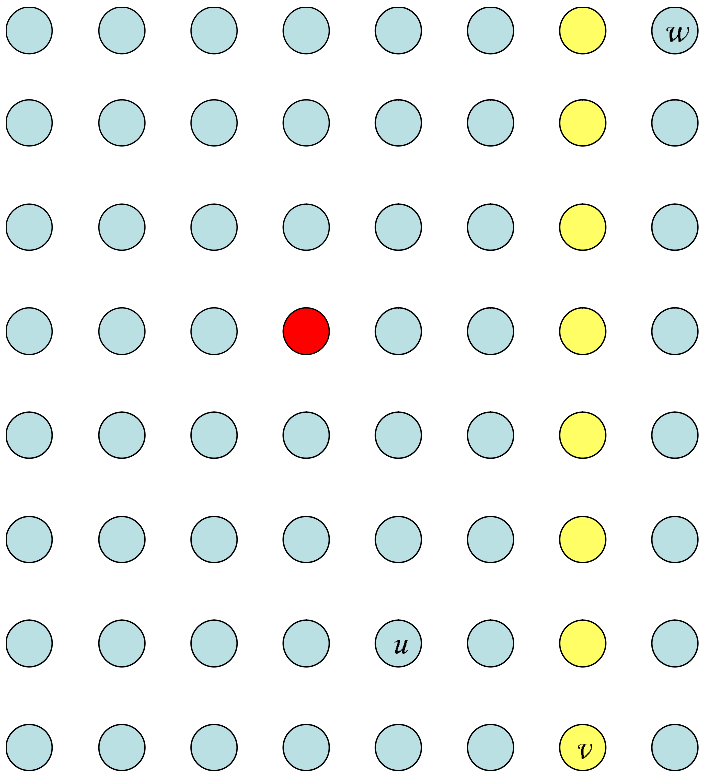


2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
140
160
180
200


xuk
3
2
1
0
0
50
100
150
200
iterindex



xvk
iterindex
0
1
2
3
0
50
100
150
200


w
u
v