Desynchronization for Sensor Networks with Arbitrary Graph Topology

Arik Motskin

December 5, 2007

1 Introduction

This project addresses the problem of distributed *desynchronization* of sensor networks with arbitrary graph topology. The motivation for desynchronization is wide-ranging, including the establishment of collision-free communication windows, coordinated sleep schedules, task allocation, and periodic resource sharing. Although work has been done on this problem in the context of complete graphs (see below), the approaches taken so far do not extend to arbitrary graphs. To tackle this problem, we frame the desynchronization problem as a discrete one, where being desynchronized is akin to finding a proper vertex coloring over the connectivity graph.

Much of this paper focuses on developing and analyzing novel algorithms for distributed vertex coloring, where the emphasis is on reducing the communication and computational resources required to perform this task quickly and efficiently. Our main contribution is an analysis of a series of novel randomized algorithms that converge to a proper coloring in $O(\log n)$ rounds with high probability, using the bare minimum in communication and computational resources.

The plan for the paper is as follows. After discussing existing desynchronization work in Section 1.1, we examine distributed vertex coloring and examine the spectrum of potential communication and computational models for solving this problem in Section 2. In Sections 3 and 4, we present and analyze the Neighbor Counting and Neighbor Detection algorithms, whose analysis forms the bulk of the technical work in this paper. Section 5 includes experimental performance of this algorithm, while Section 6 offers an alternative algorithm with different properties. Section 7 shows how to convert a distributed vertex coloring algorithm into a desynchronization protocol for arbitrary graphs, and we conclude in Section 8.

1.1 Existing Work

Although much past work in sensor networks has focused on various aspects of synchronization (clock synchronization, firefly-like pulse synchronization, etc.), recent work has shifted focus to the problem of *desynchronization*, as evidenced by the novel DESYNC paradigm of [1]. There, sensor nodes are modeled as oscillators with frequency $\omega = \frac{1}{T}$. Each period, nodes must select a phase at which to fire so as to adequately distance their firing time from that of their neighbors. Since the network topology considered in that work is a complete graph, each node can hear the firing time of all other nodes, and must make a decision (without message-passing) as to how to set its firing time based solely on what it has heard. At convergence, each node will have claimed a $\frac{1}{n}$ slice of the length T period. The paper presents a lightweight, distributed, self-maintaining protocol, that is shown in [6] to always converge to ε-desynchrony after $O\left(n^2 \log \frac{1}{\varepsilon}\right)$ periods. Moreover, the DESYNC protocol has the property that sensor nodes do not have to be synchronized to a global clock, since all phase selections are made on a relative-time basis.

On the other hand, the DESYNC protocol requires each node to be awake for the entire period of length T, since a node must hear the firing just before and after its own firing, but has no prediction of when this might occur. Most problematic, however, is that the DESYNC protocol does not extend to arbitary graphs; since real-life sensor network deployments do not form complete communication graphs in general, there is obvious impetus to construct protocols that work on networks with non-trivial topologies.

This paper addresses these issues by treating desynchronization as a discrete problem on the network of sensor nodes. A solution to the minimum vertex coloring problem would lend itself naturally to optimal sensor network desynchronization, where each node's assigned color would correspond to a fraction of the length T period. In a proper coloring using m colors, each node could be guaranteed a size $\frac{T}{m}$ slice of the period for itself. As minimum vertex coloring is NP-hard, we cannot hope to find the optimal solution; moreover, our desynchronization algorithm should be *distributed*, making the problem even more difficult. This paper focuses on designing distributed vertex coloring algorithms with good convergence behavior and reasonable bounds on the number of colors used.

2 Distributed Vertex Coloring

When judging the performance of a distributed vertex coloring algorithm on a graph with n nodes, we recall Brooks' Theorem, which states that any graph besides the complete graph and the odd cycle can be colored with Δ colors (where Δ is the maximum node degree). We also recall the trivial centralized algorithm which uses $\Delta + 1$ colors and completes in n rounds. With that in mind, we view Δ to be a reasonable benchmark for the minimum number of colors required for a distributed algorithm to converge quickly to a proper coloring.

But how well can a distributed algorithm do, both in the number of colors used and in the speed of convergence? Past work, and results presented in this paper, can be characterized by the amount of communication (between nodes) and computation (within a node) that an algorithm permits nodes to engage in. As sensor nodes have limited battery life for communication and processing capability for computation, lightweight algorithms are not only preferable but necessary for real-life deployment. We remark that there is a natural tradeoff between, on the one hand, richness of actions available to nodes, and on the other, the performance of the algorithm. As allowable node behavior becomes more restrictive, either i) the time until convergence will lengthen, or ii) number of colors required will increase. Below, we characterize a spectrum of computational models, from most rich to most naive. Each model encodes the actions available to nodes during each round of a vertex coloring algorithm; for all algorithms, we assume that a color is selected at the conclusion of each round (either a new color or the same one just used), and that once a proper coloring has been reached the nodes must know not to switch colors.

MODEL 1 – Rich communication and computation

In every round, each node can:

a) execute arbitrary amount of computation

b) send and receive arbitrary number of messages to and from its neighbors

[5] displays a deterministic distributed algorithm using $O(\Delta)$ colors that converges in $O\left(\log^*\left(\frac{n}{\Delta}\right)\right)$ rounds.

MODEL 2 – Moderate communication and computation

In every round, each node can:

a) maintain a palette of potential and available colors

b) observe whether it has a color conflict with any neighbor

c) send and receive a *done* message once a node decides to *permanently* select a color.

[4] and [2] demonstrate a simple and intuitive algorithm using only $\Delta + 1$ colors that converges in $O(\log n)$ rounds with high probability. Interestingly, for this result to hold, nodes need not know the max degree Δ of the network, but only their own degree.

MODEL 3 – Minimal communication and computation

In every round, each node can:

a) observe the number of neighbors in conflict with it

b) select next color based on above observation

In the next section, we present an algorithm under this model – dubbed Neighbor Counting – using $O(\Delta)$ colors that converges in $O(\log n)$ rounds with high probability. We observe that in moving from Model 2 to the far more restrictive Model 3, we require slightly more colors in order to achieve similar time of convergence bounds, and nodes must all know the max degree Δ. On the other hand, Neighbor Counting critically improves on the algorithm in [4] by being *self-maintaining*; that is, if new links are formed or errors are introduced, the Neighbor Counting protocol easily handles and fixes such situations.

MODEL 4 – Minimal communication and computation, also minimal monitoring

In every round, each node can:

a) observe only whether any neighbor has a conflict with it

b) select next color based on this single bit of information.

With only a small tweak to Neighbor Counting, we present an algorithm under this most restrictive model called Neighbor Detection that, besides different constants, guarantees the same time of convergence and number of color bounds as Neighbor Counting. This is an important improvement, since in the context of sensor networks, observing *whether* there are conflicts is a much easier task than *counting* the number of conflicts.

2.1 Notations and Assumptions

We assume throughout that the graph of n nodes has a maximum node degree of $\Delta \geq 2$. Otherwise, any reasonable procedure reaches a proper coloring quickly with 2 colors.

3 Neighbor Counting

In this section we introduce our novel, distributed vertex-coloring algorithm – dubbed the Neighbor Counting algorithm – which satisfies the Model 3 characterization of communication and computation.

3.1 The Algorithm

Neighbor Counting works on the principle that since both computational and communication resources are scarce, nodes must make severe simplifying assumptions about the state of the graph when selecting their colors. Since message-passing is forbidden, nodes are forced to simply observe the colors chosen by their neighbors, oblivious to what color patterns may be unfolding only two hops away; since non-trivial computation is forbidden, nodes must decide by only basic computations what color to select next round.

The simplifying assumption made by nodes in Neighbor Counting is to treat the local neighborhood as a complete graph. If I am colored green and three of my neighbors are as well, it is reasonable from my myopic perspective to hope that only one of us will remain green next round, at least in expectation. This would be a great strategy if the green-colored nodes indeed form a 4-clique (which they do not, in general). The algorithm proceeds such that if a node observes k of her neighbors with the same color as her, she will remain at her color with probability $\frac{1}{k+1}$, and switch to *another* color uniformly at random with the remaining probability. The algorithm works as follows:

1) At time 1, each node chooses a color uniformly at random (out of m possible).

2) At outset of each round, each node counts the number of its neighbors with the same color as it, say k of them.

3) At end of round, node remains at current color with probability $\frac{1}{k+1}$ and switches to one of other $m - 1$ colors uniformly at random.

We remark that this protocol works on arbitrary graphs, but nodes must know in advance the number of colors m being used. We will see in the next section that in order to achieve good convergence time, we will set m to be a function of the maximum degree Δ. One can imagine that in an actual deployment, if a higher degree is observed than the one assumed *a priori*, a flood warning of a higher Δ can be initiated.

We observe that a node which has no similarly colored neighbors at time t may very well conflict with a neighbor at some later time. This is a direct result of the naive computation and communication model employed by Neighbor Counting. Nodes are not permitted to broadcast that they have found a good color. While it may seem that as a result of this sacrifice, the speed to convergence may suffer, we will see in the next section that Neighbor Counting algorithm can still converge in $O(\log n)$ rounds.

On the other hand, Neighbor Counting has the advantage of being *self-maintaining* in the event of node failures, link failures, or the appearance of new nodes or links. The protocol easily handles any of these issues, since no node ever settles on a “permanent” color.

3.2 Main Convergence Result

The goal of this section is to show that the Neighbor Counting algorithm converges to a proper vertex coloring in $O(\log n)$ rounds with high probability. Suppose there are $k\Delta$ colors, for $k \geq 1$. Define a node v to be *good* in round t if each of its neighbors is colored differently from v during that round; otherwise the node is *bad*. Let the random variable X_t be the number of good nodes (out of n total nodes) at round t. Since the algorithm halts at a proper coloring – ie. when all nodes are good – we wish to characterize the time it takes for X_t to reach n. This is done, in part, by proving a lower bound on the probability that a good node *remains* good from round to round. The key to proving our main result is to demonstrate that this probability actually converges to 1 as the number of good nodes approaches n.

Fix time t. Suppose that good node v, currently colored c, has Δ bad neighbors. We wish to lower bound the probability that v will also be good *next* round.

$$\begin{aligned}
\mathbb{P}(\text{node } v \text{ remains good next round}) &= \mathbb{P}(\text{none of } v\text{'s neighbors switch to c}) \\
&\geq \left[1 - \left(\frac{\Delta - 1}{\Delta}\right)\left(\frac{1}{k\Delta - 1}\right)\right]^{\Delta} \\
&\geq \left[1 - \frac{1}{k\Delta}\right]^{\Delta} \\
&= \left[\left(1 - \frac{1}{k\Delta}\right)^{k\Delta}\right]^{\frac{1}{k}} \qquad (1) \\
&\geq \left[e^{-1}\left(1 - \frac{1}{k\Delta}\right)\right]^{\frac{1}{k}} \\
&\geq \left[e^{-1}\left(1 - \frac{1}{2k}\right)\right]^{\frac{1}{k}} \\
&= p^*
\end{aligned}$$

Clearly, nodes that have fewer than Δ bad neighbors also remain good with probability at least p^*. We remark that by choosing k high enough, we can make p^* as close to 1 as we wish, say $p^* = 0.85$ (if $k = 7$). We now wish to lower bound the probability that a node u that is currently *bad*, will become good next round. By definition, u has at least one node with the same color, so it will switch colors next round with probability at least $\frac{1}{2}$.

$$\begin{aligned}
&\mathbb{P}(\text{node } u \text{ becomes good next round}) \\
&\qquad \geq \mathbb{P}(u \text{ switches to a color that is currently unclaimed, and} \\
&\qquad\qquad \text{which no neighboring node will also claim next round}) \\
&\qquad \geq \left(\frac{1}{2}\right)\left(\frac{k\Delta - (\Delta - 1)}{k\Delta - 1}\right)\mathbb{P}(\text{no neighboring node} \\
&\qquad\qquad\qquad \text{switches to } u\text{'s color}) \\
&\qquad \geq \left(\frac{k-1}{2k}\right)\mathbb{P}(\text{no neighboring nodes switch to } u\text{'s color}) \\
&\qquad = \left(\frac{k-1}{2k}\right)\mathbb{P}(\text{each of } \Delta \text{ neighboring nodes don't switch to } u\text{'s color}) \\
&\qquad \geq \left(\frac{k-1}{2k}\right)p^* \\
&\qquad = q^*
\end{aligned} \tag{2}$$

Again, we remark that by choosing k high enough, we can make q^* as close to $\frac{1}{2}$ as we wish, say 0.35 (if $k = 7$).

Lemma 1
If $X_t \leq \frac{2n}{3}$*, then* $\mathbb{E}[X_{t+1}|X_t] \geq X_t + 0.05(n - X_t)$.

Proof. By Equations 1 and 2, $\mathbb{E}[X_{t+1}|X_t] \geq 0.85X_t + 0.35(n - X_t) = (0.85X_t + 0.3(n - X_t)) + 0.05(n - X_t)$. Since $X_t \leq \frac{2n}{3}$, we have that $X_t \leq 2(n - X_t)$. Therefore, $\mathbb{E}[X_{t+1}|X_t] \geq (0.85X_t + 0.15X_t) + 0.05(n - X_t) = X_t + 0.05(n - X_t)$. ∎

Next we will prove a similar result for $\frac{n(\Delta-1)}{\Delta} \geq X_t > \frac{2n}{3}$. We remark that the bounds in Equations 1 and 2 will no longer be sufficient.

Suppose that there are currently $n\frac{\Delta-1}{\Delta} \geq X_t = n\frac{w}{w+1}$ good nodes, for $w > 2$. Since each bad node can be a neighbor to at most Δ good nodes (in fact, to at most $\Delta - 1$), there are at most $n\frac{\Delta}{w+1}$ bad neighbors, distributed among $n\frac{w}{w+1}$ good nodes. Therefore, the average number of bad neighbors each good node has is at most $\frac{\Delta}{w}$. Since we wish to lower bound in expectation the number of good nodes that will remain good next round, we note by Equation 1 that in the worst case all good nodes have the same number of bad neighbors as each other. That is, in the worst case, each good node has $\lceil\frac{\Delta}{w}\rceil$ bad neighbors. Therefore we alter Equation 1 for the current situation where good nodes have at most $\lceil\frac{\Delta}{w}\rceil$ bad neighbors:

$$\begin{aligned}
\mathbb{P}(\text{good node remains good next round}) &= \mathbb{P}\left(\text{none of v's } \left\lceil \frac{\Delta}{w} \right\rceil \text{ bad neighbors switch to c}\right) \\
&\geq \left[1 - \left(\frac{\Delta - 1}{\Delta}\right)\left(\frac{1}{k\Delta - 1}\right)\right]^{\lceil \frac{\Delta}{w} \rceil} \\
&= \left(\left[1 - \left(\frac{\Delta - 1}{\Delta}\right)\left(\frac{1}{k\Delta - 1}\right)\right]^{\Delta}\right)^{\frac{\lceil \frac{\Delta}{w} \rceil}{\Delta}} \\
&\geq (p^*)^{\frac{\lceil \frac{\Delta}{w} \rceil}{\Delta}} \\
&\geq (p^*)^{\frac{2}{w}}
\end{aligned} \tag{3}$$

since we know that $\frac{2}{w} \geq \frac{\lceil \frac{\Delta}{w} \rceil}{\Delta}$ when $\Delta \geq w$, as it is here.

Lemma 2
If $n\frac{\Delta-1}{\Delta} \geq X_t > \frac{2n}{3}$, then $\mathbb{E}[X_{t+1}|X_t] \geq X_t + 0.05(n - X_t)$.

Proof. Suppose $X_t = n\frac{w}{w+1}$ where $(\Delta - 1) \geq w > 2$. Then by Equations 2 and 3,

$$\begin{aligned}
\mathbb{E}[X_{t+1}|X_t] &\geq (0.85)^{\frac{2}{w}}\left(\frac{w}{w+1}\right)n + (0.35)\left(\frac{1}{w+1}\right)n \\
&= \left(\sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{2\log .85}{w}\right)^i\right)\left(\frac{w}{w+1}\right)n + (0.35)\left(\frac{1}{w+1}\right)n \\
&\geq \left(1 + \frac{2\log .85}{w}\right)\left(\frac{w}{w+1}\right)n + (0.35)\left(\frac{1}{w+1}\right)n \\
&= \left(\frac{w}{w+1}\right)n + \frac{\left(\frac{2\log .85}{w}\right) + 0.35}{w+1}n \\
&\geq \left(\frac{w}{w+1}\right)n + \frac{0.05}{w+1}n \\
&= X_t + 0.05(n - X_t)
\end{aligned} \tag{4}$$

■

Finally, we will prove a similar result for $n \geq X_t > \frac{n(\Delta-1)}{\Delta}$.

Lemma 3
If $n \geq X_t > n\frac{\Delta-1}{\Delta}$, then $\mathbb{E}[X_{t+1}|X_t] \geq X_t + 0.05(n - X_t)$.

Proof. We recall that a bad node can be a neighbor to at most $\Delta - 1$ good nodes, and that in the worst case each good node has an equal number of bad neighbors. Therefore if $X_t > n\frac{\Delta-1}{\Delta}$, then in the worst case $(\Delta - 1)(n - X_t)$ good nodes have a single bad neighbor, while the remaining good nodes have no bad neighbors.

$$\begin{aligned}
\mathbb{E}[X_{t+1}|X_t] &= [(\Delta - 1)(n - X_t)]\left(1 - \frac{1}{\Delta k}\right) + [X_t + (n - X_t)(\Delta - 1)] + 0.35(N - X_t) \\
&= (n - X_t)(\Delta - 1)\left(-\frac{1}{\Delta k}\right) + X_t + 0.35(n - X_t) \\
&= [X_t + 0.05(n - X_t)] + (n - X_t)\left(\frac{1}{\Delta k} - \frac{1}{k} + 0.3\right) \\
&\geq X_t + 0.05(n - X_t)
\end{aligned} \tag{5}$$

as long as $k \geq \frac{10}{3}$. ■

Proposition 1
If X_t is the number of good nodes at time t, then $\mathbb{E}[X_{t+1}|X_t] \geq X_t + 0.05(n - X_t)$ for all values of X_t in $[0, n]$.

Our final task is to show that the algorithm converges quickly to a proper coloring. In particular, we prove the following general result:

Proposition 2
Let $n > 0$ be an integer, and X_t be an integer random variable in $[0, n]$ for $t \in \{1, 2, 3, \ldots\}$. Define variable T to be the smallest t such that $X_t = n$. Suppose there exists constant $0 < c < 1$ such that $\mathbb{E}[X_{t+1}|X_t] \geq X_t + c(n - X_t)$. Then

$$\mathbb{P}(T \geq \lceil \log_b n \rceil + k + 1) \leq (1 - c)^k$$

where $b = \frac{1}{1-c}$.

This is an adapation of the classic result on probablistic recurrence relations in [3]. Proposition 2 generalizes the constant c case of that result by no longer requiring the restriction that X_t be a monotone non-decreasing random variable. Our proof of this result uses a straightforward application of the Markov inequality.

Proof. Suppose in the worst case that $X_1 = 0$. Fix integer $k \geq 0$. By the law of iterated expectations, $\mathbb{E}[X_{\lceil \log_b n \rceil + k + 1}|X_1] \geq n - (1 - c)^k$. Define $Y = n - X_{\lceil \log_b n \rceil + k + 1}$. By the Markov inequality,

$$\begin{aligned}
\mathbb{P}[Y \geq 1] &\leq \mathbb{E}[Y] \\
&\leq (1 - c)^k
\end{aligned} \tag{6}$$

Since $Y < 1 \Leftrightarrow X_{\lceil \log_b n \rceil + k + 1} = n$, this proves the theorem. ■

Proposition 2 implies the result we seek for convergence of the Neighbor Counting Algorithm.

Theorem 1
On a graph with n nodes, the Neighbor Counting algorithm converges to a proper coloring in $O(\log n)$ rounds with high probability when using $O(\Delta)$ colors.

In particular, if T is the number of rounds until convergence, then using $k\Delta$ colors,

$$\mathbb{P}(T \geq \lceil \log_b n \rceil + w + 1) \leq (1-c)^w \tag{7}$$

where $b = \frac{20}{19}$, $c = 0.05$ and $k \geq 7$. We observe that due to the coarseness of the bounds used in our proof, in practice the $O(\log n)$ convergence result will also hold for smaller values of k, or larger values of b and c. Moreover, there is a natural tradeoff between number of colors used and time until convergence, which will be examined experimentally in Section 5.

4 Neighbor Detection

The previous algorithm required that at each round, nodes *count* the number of neighbors in conflict with them. It is a far less demanding task if nodes are to only detect *whether* they are in conflict with any neighbor. The following Neighbor Detection algorithm achieves the same bounds for distributed vertex coloring as does Neighbor Counting, but with different constants.

1) At time 1, each node chooses a color uniformly at random (out of m possible).

2) At outset of each round, each node checks whether any of its neighbors have the same color as it.

3) At end of round, node remains at current color if it had no conflict; otherwise, it switches to one of other $m-1$ colors uniformly at random.

The following analysis establishes the same bounds as those for Neighbor Counting. Suppose there are $k\Delta$ colors. Corresponding to equation 1, if v is currently a good node,

$$\begin{aligned}
\mathbb{P}(\text{node } v \text{ remains good next round}) &= \mathbb{P}(\text{none of } v\text{'s neighbors switch to c}) \\
&\geq \left[1 - \left(\frac{1}{k\Delta - 1}\right)\right]^{\Delta} \\
&\geq \left[1 - \frac{1}{(k-1)\,\Delta}\right]^{\Delta} \\
&\geq \left[e^{-1}\left(1 - \frac{1}{2\,(k-1)}\right)\right]^{\frac{1}{k-1}} \\
&= p^*
\end{aligned} \tag{8}$$

Here, by choosing $k = 5$, we get $p^* > 0.75$. Now consider a bad node u. Corresponding to the analysis in equation 2,

$$\begin{aligned}
&\mathbb{P}(\text{node } u \text{ becomes good next round}) \\
&\qquad \geq \mathbb{P}(u \text{ switches to a color that is currently unclaimed, and} \\
&\qquad\qquad \text{which no neighboring node will also claim next round}) \\
&\qquad \geq \left(\frac{k\Delta - (\Delta - 1)}{k\Delta - 1}\right) \mathbb{P}(\text{no neighboring node} \\
&\qquad\qquad\qquad \text{switches to } u\text{'s color}) \qquad (9) \\
&\qquad \geq \left(\frac{k-1}{k}\right) p^* \\
&\qquad = q^*
\end{aligned}$$

Setting $k = 5$ achieves a $q^* > 0.6$. It can be shown, following the exact same analysis as for Neighbor Counting, and using values of $p^* > 0.75$ and $q^* > 0.6$, that Lemmas 1, 2, 3 and Theorem 1 hold, except we only require that $k = 5$ (the same b and c values still hold). Thus, while Neighbor Counting requires 7Δ colors to achieve the convergence speed in Theorem 1, Neighbor Counting requires only 5Δ colors. This gain is due entirely to the fact that, with a fixed k, although good nodes have a slightly smaller probability of remaining good in Neighbor Detection than in Neighbor Counting, bad nodes have a much greater probability of *becoming* good. We confirm experimentally that Neighbor Detection is the faster algorithm in section 5.

5 Experimental Results

To examine the tradeoff between number of colors and time of convergence, we compared the performance of the Neighbor Counting and Neighbor Detection distributed vertex coloring algorithms empirically. In the experiment, 500 nodes were placed uniformly at random in the unit square, and connectivity was determined by a unit disk graph model. For both algorithms, 1000 trials of the algorithm were run for each $k \in \{1, 2, \ldots, 20\}$, and the mean number of rounds until convergence was computed. Several radius lengths for the unit disk graph model were tested – figures 1 and 2 show the results for unit disk graph radii of 0.05 and 0.1 (with $\Delta = 10$ and 27 respectively). The results presented here are typical for all radii tested.

The results indicate that Neighbor Detection converges on average $1.5 - 2$ times faster than Neighbor Counting, and that for both the function appears concave. We note that no matter what algorithm is being used, there will be at least 1 round, since the nodes take at least a single round to verify that there are no conflicts.

6 A Third Option

In this section, we present another algorithm, loosely based on the "wake-up" algorithms presented in [4] and [2]. The strength of those protocols is how few colors are required: using only $\Delta + 1$ colors, the algorithms converges in $O(\log n)$ rounds with high probability. However, what was gained in color efficiency was lost in communication efficiency, as they require nodes to broadcast a "done" message to their neighbors, who must in turn update and maintain an internal palette of possible colors. Moreover, those algorithms were not as amenable to self-maintenance, since nodes settle on a *permanent* color after the first time that they have avoided a conflict with their neighbors. On the other hand, while the Neighbor Counting and Detection algorithms require minimal node computation and communication and are self-maintaining, they require more colors – $O(\Delta)$ – in order to converge quickly.

The following ε*-Wake-up* algorithm strikes a middle ground between these two paradigms. Unlike our previous analysis, we will be able to establish an explicit tradeoff between speed of convergence and number of colors used. Suppose for $\varepsilon > 0$ that there are $m = (\varepsilon + 1)(\Delta + 1)$ available colors. Each node begins as uncolored. At each round, the algorithm proceeds as follows:

1) If node v is currently asleep, it "wakes-up" and selects a color c uniformly at random (out of m possible).

2) If no neighbors are also colored c, node v chooses c as its permanent color, and it calls itself *permanent*. Node v never goes back to sleep, and does not inform its neighbors that it has chosen a permanent color.

3) If some neighbor *is* also colored c, node v goes back to sleep and awaits the next round.

It is clear from the algorithm that if two neighbors u and v are permanent, then their colors do not conflict. Moreover, once all nodes are permanent, the algorithm has converged to a proper coloring. We now establish a lower bound on the probability that an uncolored node becomes permanent during a round.

$$\begin{aligned} \mathbb{P}(\text{uncolored node } v \text{ becomes permanent}) &= \mathbb{P}(v \text{ selects a color that no one else selects}) \\ &\geq \left(\frac{(\varepsilon + 1)(\Delta + 1) - \Delta}{(\varepsilon + 1)(\Delta + 1)} \right) \\ &\geq \left(\frac{\varepsilon}{\varepsilon + 1} \right) \end{aligned} \tag{10}$$

Then by Proposition 2, we get that if T is the number of rounds until convergence to a proper coloring, $\varepsilon > 0$, and using $(\varepsilon + 1)(\Delta + 1)$ colors,

$$\mathbb{P}(T \geq \lceil \log_b n \rceil + w) \leq (1 - c)^w \tag{11}$$

where $b = \varepsilon + 1$ and $c = \frac{\varepsilon}{\varepsilon+1}$.

It is notable that the probability bound and time of convergence result holds even if individual nodes do not know the maximum degree Δ. In total, there is a list of $(\varepsilon + 1)(\Delta + 1)$ potential colors that can be used, but a particular node u that has degree $\Delta_u \leq \Delta$ needs to only consider the first $(\varepsilon + 1)(\Delta_u + 1)$ members of that list when selecting a color. This is consistent with the work in [4] and [2], whereas nodes in the Neighbor Counting and Neighbor Detection algorithms need to select from the entire list of $O(\Delta)$ colors.

7 Converting to a Desynchronizing Algorithm

We now show how to convert a distributed vertex coloring algorithm into a sensor network desynchronizing protocol. As in [1], we will consider nodes to be oscillators with frequency $\omega = \frac{1}{T}$. If a coloring algorithm uses m colors, then each such color will correspond to a length $\frac{T}{m}$ interval of the length T period. If, in the coloring algorithm, some node v claimed the i^{th} color, then this will correspond to node v firing for the duration of the i^{th} interval of the period.

We assume in the ensuing discussion that each node can measure the length of time T and $\frac{T}{m}$, and knowing the start and end times of a period, can choose to fire for the duration of the i^{th} interval of length $\frac{T}{m}$. For purpose of analysis, assume that there exists a global clock (with which the nodes are not necessarily synchronized) that keeps track of the beginning of each “global” round of length T at times $\{0, T, 2T, 3T, \ldots\}$. Also, assume that each node v has its own internal clock which keeps track of what it *thinks* is the beginning of each round at times $\{s_v, s_v + T, s_v + 2T, s_v + 3T, \ldots\}$, where $0 \leq s_v < T$.

If every node’s clock is synchronized to the global clock (ie. where $s_v = 0$ for all nodes v), then the vertex coloring algorithms are easily adapted to a desynchronization protocol. Every period, nodes choose a size $\frac{T}{m}$ interval of the period during which to fire; depending on whether there is a conflict or not, a node may choose a different interval next period according to whatever vertex coloring procedure is being followed (Neighbor Detection, Neighbor Counting, etc.). The system is said to be desynchronized when for every node, the selected firing interval does not overlap with any of its neighbors’ firing intervals. Each node has, in a sense, staked out a length $\frac{T}{m}$ interval for itself. Accordingly, all previous convergence results hold (including the values of the constants b, c and k); that is, the sensor networks converges to desynchronization in $O(\log n)$ periods when the period T is divided into $O(\Delta)$ firing intervals.

If each node’s clock is not necessarily synchronized to the global clock, we run the desynchronizing algorithm as above, but the analysis is complicated by two issues: the beginning and end of the *firing intervals* of each node do not align with those of its neighbors, and the beginning and end of the *period* of each node does not align with that of its neighbors. For the first issue, we remark that if all nodes use firing intervals of the same length (which we assume they do), then the i^{th} interval of node v can overlap with at most 2 of a neighbor’s firing intervals. Then if the second issue did not exist (ie. beginning and end of each node’s period was synchronized), this would imply that we require *double* the number of colors as in the synchronized case in order to converge in the same number of rounds.

With regards to the second issue, we perform the following accounting to keep track of “good” and “bad” nodes: at global time t, we say a node v is good if its firing interval did not overlap with the firing interval of its neighbors during its *last period that ended before time t*. Otherwise, it is bad. With this definition, it can be seen that if the clock synchronized case used m firing intervals of length $\frac{T}{m}$, then by using $2m$ firing intervals of length $\frac{T}{2m}$ in the unsynchronized case, the probability that a good node at time t is also good at time $t + T$ is lower bounded by the probability of that event in the synchronized case. The same argument holds for showing a lower bound on the probability that a bad node at time t will become good at time $t + T$. That is, to achieve the same convergence results as the clock synchronized case, we require just twice as many firing intervals.

Recall that in the DESYNC protocol of [1], nodes were required to be awake for the entire period of length T in order to hear the firing times of their neighbors. For our procedure, a node is required to be awake only during its own firing interval, since it must only check whether neighbors' intervals have overlapped with its own – this property holds irrespective of whether nodes' clocks are or are not synchronized to a global clock.

8 Conclusion

In this project we address the problem of desynchronizing sensor networks with arbitrary graph topologies by abstracting the problem to a distributed vertex coloring one. We put forth and analyzed three algorithms that achieve this goal: Neighbor Counting, Neighbor Detection, and the ε-Wake-up algorithm, each of which converges to a proper vertex coloring in $O(\log n)$ rounds with high probability. Compared to existing work, each of these algorithms uses a very restricted computational and communication model, achieving quick convergence despite the limited actions available to each node. Neighbor Counting and Detection both have the property of being self-maintaining (if new links or nodes appear, the algorithm easily adjusts the coloring); on the other hand, while ε-Wake-up is not self-maintaining, it has a convenient tradeoff lever (parameterized by ε) that characterizes the speed of the algorithm compared to the number of colors used (and it can achieve quick convergence using fewer colors than Neighbor Counting or Detection).

Moreover, although Neighbor Detection uses a more restrictive computational model than Neighbor Counting, it converges to a proper coloring faster (by a constant factor), indicating that the option in Neighbor Counting to *count* the number of conflicts (instead of simply detecting *whether* there is a conflict) may not add any computational value to algorithms in this context. Finally, we showed how vertex coloring algorithms can be transformed into sensor network desynchronization protocols, noting that if all nodes' clocks are synchronized to a global clock, convergence to desynchrony follows directly from the vertex coloring convergence results, while unsynchronized clocks achieve the same speed of convergence as long as the number of available "colors" (firing intervals) is doubled.

References

[1] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal, *Desync: self-organizing desynchronization and tdma on wireless sensor networks*, IPSN, 2007, pp. 11–20.

[2] Ojvind Johansson, *Simple distributed* $\delta + 1$*-coloring of graphs*, Information Processing Letters **70(5)** (1999), 229–232.

[3] Richard M. Karp, *Probabilistic recurrence relations*, Journal of the ACM **41** (1994), 1136–1150.

[4] Michael Luby, *Removing randomness in parallel computation without a processor penalty*, IEEE Symposium on Foundations of Computer Science, 1988, pp. 162–173.

[5] Gianluca De Marco and Andrzej Pelc, *Fast distributed graph coloring with $\mathcal{O}(\delta)$ colors*, SODA (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2001, pp. 630–635.

[6] Ankit Patel, Julius Degesys, and Radhika Nagpal, *Desynchronization: The theory of self-organizing algorithms for round-robin scheduling*, SASO, 2007, pp. 87–96.

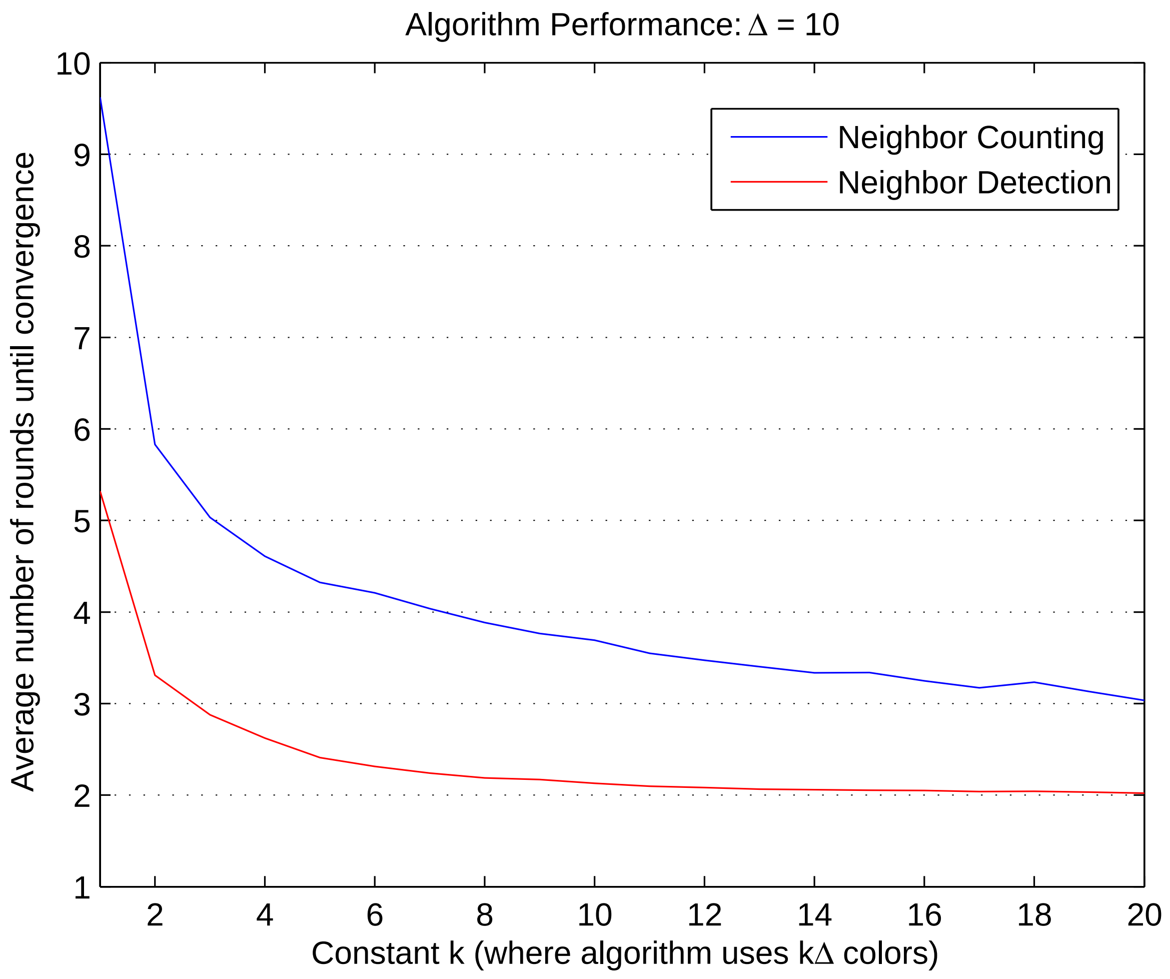
Algorithm Performance: Δ = 10
Neighbor Counting
Neighbor Detection
Average number of rounds until convergence
Constant k (where algorithm uses kΔ colors)
1
2
3
4
5
6
7
8
9
10
2
4
6
8
10
12
14
16
18
20

FIGURE 1

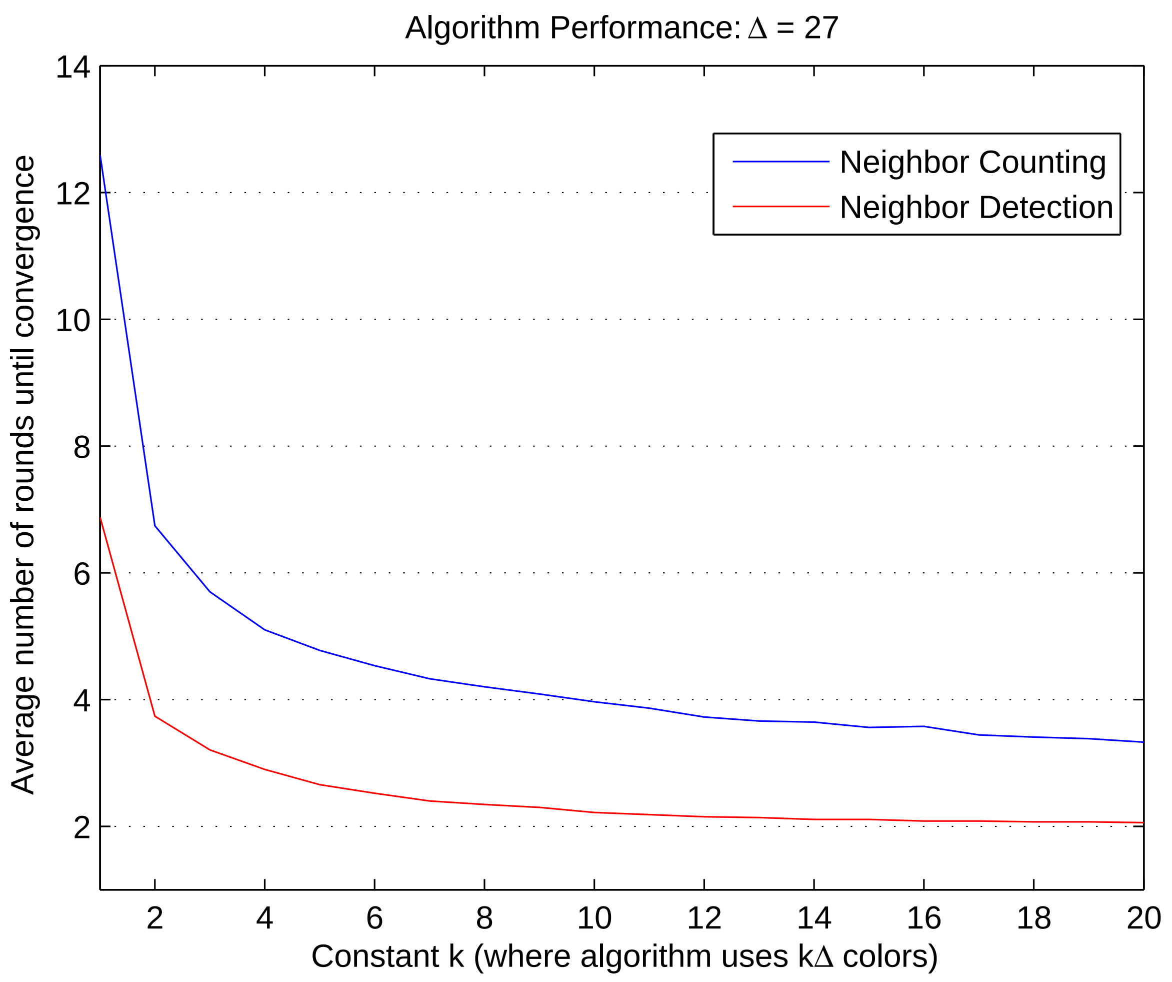
Algorithm Performance: Δ = 27
Average number of rounds until convergence
Constant k (where algorithm uses kΔ colors)
Neighbor Counting
Neighbor Detection
2
4
6
8
10
12
14
16
18
20

FIGURE 2