Distributed Game-Theoretic Vertex Coloring[⋆]

Ioannis Chatzigiannakis[1,2], Christos Koninis[1,2],
Panagiota N. Panagopoulou[2], and Paul G. Spirakis[1,2]

[1] Computer Engineering and Informatics Department, Patras University
[2] Research Academic Computer Technology Institute, Greece
{ichatz,koninis,panagopp,spirakis}@cti.gr

Abstract. We exploit the game-theoretic ideas presented in [12] to study the vertex coloring problem in a distributed setting. The vertices of the graph are seen as players in a suitably defined strategic game, where each player has to choose some color, and the payoff of a vertex is the total number of players that have chosen the same color as its own. We extend here the results of [12] by showing that, if any subset of non-neighboring vertices perform a selfish step (i.e., change their colors in order to increase their payoffs) in parallel, then a (Nash equilibrium) proper coloring, using a number of colors within several known upper bounds on the chromatic number, can still be reached in polynomial time. We also present an implementation of the distributed algorithm in wireless networks of tiny devices and evaluate the performance in simulated and experimental environments. The performance analysis indicates that it is the first practically implementable distributed algorithm.

1 Introduction

One of the central optimization problems in Computer Science is the problem of *vertex coloring* of graphs, i.e., the problem of assigning a color to each vertex of the graph so that no pair of adjacent vertices gets the same color (i.e., so that the coloring is *proper*) and so that the total number of distinct colors used is minimized. In this work, we deal with the problem of vertex coloring in a distributed setting, with a focus on distributed implementations applicable to Wireless Sensor Networks (WSNs). Finding a good (with respect to the total number of colors used) coloring of the nodes of a WSN has many practical applications: First, colors may be seen as frequencies, so that a proper coloring of the nodes corresponds to a solution to the frequency assignment problem; Furthermore, a coloring of a WSN actually partitions its nodes into subsets (each corresponding to a color), such that no communication link exists between any pair of nodes in the same subset, and such a partition might be useful when designing sleep/awake protocols in order to save energy or providing secure group communication.

[⋆] This work has been partially supported by the ICT Programme of the European Union under contract numbers ICT-2008-215270 (FRONTS) and ICT-2008-224460 (WISEBED).

The challenge of designing vertex coloring algorithms applicable in WSNs lies in both the intrinsic difficulty of the original problem of vertex coloring and the particularities of WSNs. More specifically, the global optimum of vertex coloring is NP-hard [7], and the best polynomial time approximation algorithm achieves an approximation ratio of $O(n(\log\log n)^2/(\log n)^3)$ [6] (n being the number of vertices). In the distributed setting of wireless sensor networks the problem of vertex coloring has been studied before; the randomized algorithm presented in [10] needs, with high probability, $O(\Delta \log n)$ time and uses $O(\Delta)$ colors, where n and Δ are the number of nodes in the network and the maximum degree, respectively. This algorithm requires knowledge of a linear bound on n and Δ. This result was improved in [14], where the coloring problem is solved in $O(\Delta + \log\Delta \log n)$ time, given an estimate of n and Δ, and $O(\Delta + \log^2 n)$ without knowledge of Δ, while it needs $\Delta + 1$ colors.

In this work, our objective is to find a (suboptimal, inevitably) proper coloring, that at least guarantees some bounded maximum total number of colors used. Towards this direction, we try to exploit the ideas presented in [12]: the vertices of the graph are seen as players in a suitably defined strategic game, where each player has to choose some color, and the payoff of a player is the total number of players that have chosen the same color as his own (unless some neighbor has also chosen the same color, in which case the payoff is 0). The vertices are allowed to perform, sequentially, selfish steps, i.e., change their colors in order to increase their payoffs, and in [12] it was shown that this selfish improvement sequence converges, in polynomial time, into a pure Nash equilibrium of the game, which is actually a proper coloring of the vertices of the graph that uses a total number of colors satisfying all known upper bounds on the chromatic number of the graph (that is, the minimum number of colors needed to color the graph).

Our contribution. Our objective here is to exploit the game-theoretic ideas presented in [12] so as to deploy an efficient, in terms of both time complexity and number of colors used, distributed algorithm for vertex coloring. The algorithm of [12] is in fact a local search method (the vertices perform local changes, by moving to color classes of higher cardinality, until no further local moves are possible); however, it relies on several assumptions that obstruct its straightforward implementation in a distributed setting. In particular, it requires that global information about the cardinalities of all color classes is at any time available to each vertex, and that only a single vertex at each time is allowed to perform a local move. Here, we propose an implementation of the algorithm that deals with these issues, thus enabling its application to distributed settings, while preserving both the efficiency (in terms of time complexity) and the quality (in terms of number of colors used) of the coloring produced.

The solution we propose relies on an extension of the results of [12] that we give here. Namely, we raise the requirement that only one vertex at a time is allowed to perform a selfish step in order to guarantee polynomial time convergence into a Nash equilibrium coloring. In particular, we show that if any subset of non-neighboring vertices perform a selfish step in parallel, then a Nash equilibrium coloring satisfying the bounds given in [12] can still be reached in polynomial time.

This potentiality of parallelization of selfish steps allows us to derive a distributed implementation of the game-theoretic vertex coloring algorithm, which computes, in polynomial time, a proper coloring of the vertices of a graph using at most

$$k \leq \min \left\{ \Delta_2(G) + 1\,,\ \frac{n + \omega(G)}{2}\,,\ n - \alpha(G) + 1\,,\ \frac{1 + \sqrt{1 + 8m}}{2}\,, \right.$$
$$\left. \frac{\chi(G) + 1}{2} + \sqrt{m - \frac{(\chi(G) - 1)(\chi(G) + 1)}{4}} \right\}$$

distinct colors, where n is the number of vertices, m is the number of edges, $\Delta_2(G)$ is the maximum degree of a vertex which is connected to a vertex of equal or higher degree, and $\omega(G)$, $\alpha(G)$ and $\chi(G)$ are the clique number, independence number and chromatic number, respectively, of the graph under consideration. To the best of our knowledge, this is the first distributed, polynomial-time implementation of a vertex coloring algorithm that achieves all the above bounds. Our algorithm requires $O(n)$ memory to maintain the local lists with the cardinalities of all color classes. Also, the protocol does not require any initial knowledge on the network (e.g., network size, average node degree etc.) but gathers all necessary information dynamically.

The paper is organized as follows. The next section introduces the definitions and notations needed to solve the problem of vertex coloring. We then present the game-theoretic approach for vertex coloring based on local search. In Section 3 we first prove that a "parallelization" of the local search approach is still possible. Based on this fact, we then present a distributed implementation of the algorithm that converges fast into a pure Nash equilibrium, and thus into a proper coloring. We also describe a self-stabilizing version of the distributed algorithm that recovers from transient faults regardless of its initial state. Section 4 presents an implementation of the distributed algorithm in wireless networks of tiny devices and evaluates the performance in simulated and experimental environments. The performance analysis indicates that our distributed algorithm is the first practically implementable for real networks.

2 Background

Definitions and notation. For a finite set A we denote by $|A|$ the cardinality of A. For a positive integer $n \in \mathbb{N}$ let $[n] = \{1, \ldots, n\}$. Denote $G = (V, E)$ a simple, undirected graph with vertex (node) set V and set of edges E. For a vertex $v \in V$ denote $N(v) = \{u \in V : \{u, v\} \in E\}$ the set of its neighbors, and let $\deg(v) = |N(v)|$ denote its degree. Let $\Delta(G) = \max_{v \in V} \deg(v)$ be the maximum degree of G. Let

$$\Delta_2(G) = \max_{u \in V} \max_{v \in N(u) : d(v) \leq d(u)} \deg(v)$$

be the maximum degree that a vertex v can have, subject to the condition that v is adjacent to at least one vertex of degree no less than $\deg(v)$. Clearly,

$\Delta_2(G) \leq \Delta(G)$. Let $\mathrm{diam}(G)$ be the diameter of G, i.e., the maximum length of a shortest path between any pair of vertices of G. Let $\omega(G)$ and $\alpha(G)$ denote the clique number and independence number of G, i.e. the number of vertices in a maximum clique and a maximum independent set of G.

The Vertex Coloring problem. One of the central optimization problems in Computer Science is the problem of *vertex coloring* of graphs: given a graph $G = (V, E)$ of n vertices, assign a color to each vertex of G so that no pair of adjacent vertices gets the same color (i.e., so that the coloring is *proper*) and so that the total number of distinct colors used is minimized. The *chromatic number* of G, denoted by $\chi(G)$, is the global optimum of vertex coloring, i.e., the minimum number of colors needed to properly color the vertices of G. The vertex coloring problem is known to be NP-hard [7], and the chromatic number cannot be approximated to within $\Omega(n^{1-\epsilon})$ for any constant $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{co\text{-}RP}$ [5].

The game-theoretic approach. Panagopoulou and Spirakis [12] proposed an efficient vertex coloring algorithm that is based on *local search*: Starting with an arbitrary proper vertex coloring (e.g. the trivial proper coloring where each vertex is assigned a unique color), each vertex (one at a time) is allowed to move to another color class of higher cardinality, until no further local moves are possible. This local search method is illustrated in [12] via a game-theoretic analysis, because of the natural correspondence of the local optima of the proposed method to the pure Nash equilibria of a suitably defined *strategic game*.

In particular, given a finite, simple, undirected graph $G = (V, E)$ with $|V| = n$ vertices, the *graph coloring game* $\Gamma(G)$ is defined as the game in strategic form where the set of players is the set of vertices V, and the action set of each vertex is a set of n colors $[n] = \{1, \ldots, n\}$ (for simplicity, we represent each color by an integer). A *configuration* or *pure strategy profile* $\mathbf{c} = (c_v)_{v \in V} \in [n]^n$ is a combination of actions, one for each vertex. That is, c_v is the color chosen by vertex v. For a configuration $\mathbf{c} \in [c]^n$ and a color $x \in [n]$, we denote by $n_x(\mathbf{c})$ the number of vertices that are colored x in $\mathbf{c}$, i.e. $n_x(\mathbf{c}) = |\{v \in V : c_v = x\}|$. The *payoff* that vertex $v \in V$ receives in the configuration $\mathbf{c} \in [n]^n$ is

$$\lambda_v(\mathbf{c}) = \begin{cases} 0 & \text{if } \exists u \in N(v) : c_u = c_v \\ n_{c_v}(\mathbf{c}) & \text{else} \end{cases} .$$

In other words, given a proper coloring, the payoff of a vertex equals the cardinality of the color class it belongs to.

A *pure Nash equilibrium* [11] is a configuration $\mathbf{c} \in [n]^n$ such that no vertex can increase its payoff by unilaterally deviating. Let $(x, \mathbf{c}_{-v})$ denote the configuration resulting from $\mathbf{c}$ if vertex v chooses color x while all the remaining vertices preserve their colors. Then, a configuration $\mathbf{c} \in [n]^n$ of the graph coloring game $\Gamma(G)$ is a pure Nash equilibrium if, for all vertices $v \in V$, $\lambda_v(x, \mathbf{c}_{-v}) \leq \lambda_v(\mathbf{c}) \quad \forall x \in [n]$. We also say that $\mathbf{c}$ is a *Nash equilibrium coloring*.

A vertex $v \in V$ is *unsatisfied* in the configuration $\mathbf{c} \in [n]^n$ if there exists a color $x \neq c_v$ such that $\lambda_v(x, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$; else we say that v is *satisfied*. Clearly, a configuration $\mathbf{c}$ is a Nash equilibrium coloring if and only if all vertices are satisfied in $\mathbf{c}$. For an unsatisfied vertex $v \in V$ in the configuration $\mathbf{c}$, we say that v performs a *selfish step* if v unilaterally deviates to some color $x \neq c_v$ such that $\lambda_v(x, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$.

The analysis of the graph coloring game given in [12] illustrates that $\Gamma(G)$ has always pure Nash equilibria, and that each pure equilibrium is a proper coloring of G. Furthermore, there exists a pure equilibrium that corresponds to an optimum coloring. It is also shown that any pure Nash equilibrium of the game is a proper coloring of G that uses a number of colors, k, bounded above by all the general known to us upper bounds on the chromatic number of G. In particular, k is proved to be bounded above by $\Delta_2(G)+1$, $\frac{n+\omega(G)}{2}$, $n-\alpha(G)+1$, and $\frac{1+\sqrt{1+8m}}{2}$. In [4], it was further shown that $k \leq \frac{\chi(G)+1}{2}+\sqrt{m-\frac{(\chi(G)-1)(\chi(G)+1)}{4}}$. Therefore, the number of colors used by any pure Nash equilibrium of $\Gamma(G)$ is at most

$$k \leq \min \left\{ \Delta_2(G)+1\,,\ \frac{n+\omega(G)}{2}\,,\ n-\alpha(G)+1\,,\ \frac{1+\sqrt{1+8m}}{2}\,, \right. \\ \left. \frac{\chi(G)+1}{2} + \sqrt{m - \frac{(\chi(G)-1)(\chi(G)+1)}{4}} \right\} . \tag{1}$$

Most interestingly, it is proven that any sequence of selfish steps, when started with a proper (e.g., the trivial) coloring, always reaches a pure Nash equilibrium in $O(n \cdot \alpha(G))$ selfish steps. The proof follows from the existence of a *potential function* [9], which is a function defined over the set of pure strategy profiles of a game, and has the property that the difference of the function's value after a player deviates equals the corresponding difference of that player's payoff. In particular, [12] showed that the graph coloring game $\Gamma(G)$ possesses a potential function defined as

$$\Phi(\mathbf{c}) = \frac{1}{2} \sum_{x \in [n]} (n_x(\mathbf{c}))^2 \quad \forall \mathbf{c} \in [n]^n \ ,$$

with the property that, for any proper coloring $\mathbf{c} \in [n]^n$, for any vertex $v \in V$, and for any color $x \in [n]$, it holds that

$$\lambda_v(x, \mathbf{c}_{-v}) - \lambda_v(\mathbf{c}) = \Phi(x, \mathbf{c}_{-v}) - \Phi(\mathbf{c}) \ .$$

Therefore, if any vertex v performs a selfish step then the value of Φ is increased as much as the payoff of v is increased. Since the payoff of v is increased by at least 1 and the value of Φ is bounded above by $\frac{n \cdot \alpha(G)}{2}$, it follows that after at most $\frac{n \cdot \alpha(G)}{2}$ selfish steps there will be no vertex that can improve its payoff (because Φ will have reached a local maximum, which is no more than the global maximum, which is no more than $(n \cdot \alpha(G))/2$), so a pure Nash equilibrium will have been reached. This implies the following simple centralized algorithm $\mathcal{A}$ that computes, in polynomial time, a pure Nash equilibrium of $\Gamma(G)$ – and thus a proper coloring of G, satisfying Inequality 1:

```
Algorithm $\mathcal{A}$
Input: Graph $G$ with vertex set $V = \{v_1, \ldots, v_n\}$
Output: A pure Nash equilibrium $\mathbf{c} = (c_{v_1}, \ldots, c_{v_n}) \in [n]^n$ of $\Gamma(G)$
 Initialization: for $i = 1$ to $n$ do $c_{v_i} = i$
 repeat
    find an unsatisfied vertex $v \in V$ and a color $x \in [n]$
              such that $\lambda_v(x, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$
    set $c_v = x$
 until all vertices are satisfied
```

3 Distributed Vertex Coloring as a Game

The algorithm $\mathcal{A}$ presented in the previous section is actually a local search algorithm: starting with an arbitrary proper vertex coloring (such as the trivial proper coloring), we allow each vertex (one at a time) to move to another color class of higher cardinality, until no further local moves are possible. Our aim is to apply this local search method in a distributed environment, so that we end up with a distributed algorithm that computes a proper coloring of the vertices of a graph using a total number of colors satisfying Inequality 1.

However, there are two main challenges arising in implementing $\mathcal{A}$ in a distributed environment: First, $\mathcal{A}$ requires *mutual exclusion*: only one node at a time can perform a selfish step in order to guarantee convergence into a pure Nash equilibrium coloring. Second, $\mathcal{A}$ requires *global knowledge*: each node needs to be aware of the cardinalities of each color class in order to decide whether to perform a selfish step or not.

In this section, we deal with both the above challenges. First, we prove that a "parallelization" of algorithm $\mathcal{A}$ is possible, in the sense that even if we allow multiple non-neighboring unsatisfied vertices to perform simultaneous selfish steps, then (a slight modification of) the algorithm still converges into a pure Nash equilibrium, and thus into a proper coloring satisfying the bounds of Inequality 1. Then, we propose a method to deal with the global information problem (possibly at the expense of parallelization) so that, after performing a selfish step, a node needs not to inform all the other nodes about the change, but only a small subset of them. Finally, we discuss the ability of our distributed algorithm to self-stabilize.

In the following, we assume that each node has, and is aware of, a unique identification number (id).

3.1 Simultaneous Execution of Selfish Steps

We study whether polynomial time convergence into a Nash equilibrium coloring can still be guaranteed if vertices are allowed to perform selfish steps simultaneously. We still assume however that all vertices have access to global information about the cardinality of each color class. Note that, in order to avoid non-proper colorings, we should not allow any pair of neighboring vertices change their colors simultaneously. Towards this direction, we allow an unsatisfied vertex to perform a selfish step only if it has the maximum id among the unsatisfied vertices in the neighborhood.

More specifically, given a configuration $\mathbf{c} \in [n]^n$, we say that the vertices perform a *joint selfish step* if:

1. Each vertex that is unsatisfied in $\mathbf{c}$ sends a message to inform its neighbors.
2. If an unsatisfied vertex v does not receive any message from a vertex of higher id, then it performs a selfish step, i.e., it changes its color to c'_v such that $\lambda_v(c'_v, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$.

Clearly, in a joint selfish step, possibly more than one vertices will change their color simultaneously. We will examine whether a sequence of joint selfish steps converges into a Nash equilibrium coloring.

Given a configuration $\mathbf{c} \in [n]^n$, we define vector $\mathbf{\Lambda}(\mathbf{c})$ as the n-vector whose ith entry, $\Lambda_i(\mathbf{c})$, is the cardinality of the ith largest color class. We say that $\mathbf{\Lambda}(\mathbf{c})$ is lexicographically greater than $\mathbf{\Lambda}(\mathbf{c}')$ if $\Lambda_i(\mathbf{c}) > \Lambda_i(\mathbf{c}')$ for some $i \in [n]$ and, if $i > 1$, then $\Lambda_j(\mathbf{c}) = \Lambda_j(\mathbf{c}')$ for all $j \in \{1, \ldots, i\}$. If $\Lambda_i(\mathbf{c}) = \Lambda_i(\mathbf{c}')$ for all $i \in [n]$, then $\mathbf{\Lambda}(\mathbf{c})$ is lexicographically equal to $\mathbf{\Lambda}(\mathbf{c}')$.

Lemma 1. *Let $\mathbf{c} \in [n]^n$ be any configuration and let $\mathbf{c}' \in [n]^n$ be a configuration that results from $\mathbf{c}$ after a joint selfish step. Then, $\mathbf{\Lambda}(\mathbf{c})$ can not be lexicographically greater than $\mathbf{\Lambda}(\mathbf{c}')$.*

Proof. Let $Y \subseteq [n]$ be the subset of colors that correspond to the color classes of maximum cardinality in $\mathbf{c}$. If the cardinality of such a maximum color class increases, then $\Lambda_1(\mathbf{c}) < \Lambda_1(\mathbf{c}')$ and we are done.

Assume now that the cardinality of each maximum color class in $\mathbf{c}$ does not increase. Observe that a vertex colored x in $\mathbf{c}$ for some $x \in Y$ gets maximum payoff, so it could improve it only by moving to a color class of equal (and thus maximum, as well) cardinality. But since the cardinality of each maximum color class in $\mathbf{c}$ does not increase, it must be the case that all these cardinalities remain the same. Therefore $\Lambda_i(\mathbf{c}) = \Lambda_i(\mathbf{c}')$ for all $i \in [k]$. This further implies that no vertex colored $x' \notin Y$ in $\mathbf{c}$ chooses color $x \in Y$ in $\mathbf{c}'$.

Similar arguments apply for the cardinalities of the second to maximum color classes in $\mathbf{c}$ and so on, to conclude that either (i) $\mathbf{\Lambda}(\mathbf{c}')$ is lexicographically greater than $\mathbf{\Lambda}(\mathbf{c}')$ or (ii) $\mathbf{\Lambda}(\mathbf{c}')$ is lexicographically equal to $\mathbf{\Lambda}(\mathbf{c}')$ and each vertex that performs a selfish step moves to a color class of equal cardinality in $\mathbf{c}$. □

Lemma 2. *Let $\mathbf{c} \in [n]^n$ be any configuration and let $\mathbf{c}' \in [n]^n$ be a configuration that results from $\mathbf{c}$ after a joint selfish step. If $\mathbf{\Lambda}(\mathbf{c})$ is lexicographically greater than $\mathbf{\Lambda}(\mathbf{c}')$, then $\Phi(\mathbf{c}) > \Phi(\mathbf{c}')$.*

Proof. Clearly, for any configuration $\mathbf{c}''$, $\Phi(c'') = \sum_{i=1}^{n} \Lambda_i^2(\mathbf{c}'')$. The proof follows from the fact that $\mathbf{\Lambda}(\mathbf{c}')$ has at least the same number of zero entries as $\mathbf{\Lambda}(\mathbf{c}')$, since no vertex has an incentive to choose a color that is not used by any other vertex. □

In order to avoid loops of the joint selfish steps sequence, we have to deal with the case where $\mathbf{\Lambda}(\mathbf{c}')$ is lexicographically equal to $\mathbf{\Lambda}(\mathbf{c})$, and hence $\Phi(\mathbf{c}') = \Phi(\mathbf{c})$. To do so, we propose the *probabilistic joint selfish step*, given a configuration $\mathbf{c} \in [n]^n$:

1. The vertices perform a joint selfish step, resulting in configuration $\mathbf{c}'$.
2. Each vertex which performed a selfish step checks if $\Phi(\mathbf{c}) = \Phi(\mathbf{c}')$. If yes, it changes back to its previous color with probability $1/2$ and repeats this step.

We show that the probabilistic joint selfish steps sequence converges into a pure Nash equilibrium in polynomial time, with high probability:

Theorem 1. *With high probability, the probabilistic joint selfish step sequence converges in polynomial time into a Nash equilibrium coloring.*

Proof. Assume $\Phi(\mathbf{c}') = \Phi(\mathbf{c})$ after a joint selfish step. Then Lemma 1 and Lemma 2 imply that at least $k \geq 2$ vertices with different colors but with the same payoff in $\mathbf{c}$ permuted their colors. If some, but not all, of these vertices return to their original colors, then some payoffs will increase, yielding to a lexicographically greater configuration $\mathbf{c}''$, and thus $\Phi(\mathbf{c}'') > \Phi(\mathbf{c})$. The probability that this does not happen is $2 \cdot 2^{-k}$ (equal to the probability that all change their colors or all preserve their colors). So the probability that Step 2 of the probabilistic joint selfish steps sequence is repeated t times is 2^{-t-k+1}. $\square$

3.2 Distributing Global Information

The (probabilistic) joint selfish steps sequence converges to a Nash equilibrium coloring provided that all vertices know the cardinalities of all color classes. In order to achieve this in a distributed environment, we let each vertex maintain a local list with the cardinalities of all color classes. Initially, each vertex is assumed to have a unique color (its id) and all these cardinalities equal to 1. An arbitrary vertex, which we call *the Judge* initiates the coloring procedure. It performs a selfish step, if possible, and sends the original list of cardinalities to its neighbors, who become its children. Then these nodes perform a joint selfish step, if possible, and also pass the original list to their neighbors. If a vertex has no parent and receives the list from some vertex, it sets itself as the child of that vertex.

This way, a tree (whose root is the Judge) is constructed, and when the leaves are reached, all unsatisfied vertices will have actually performed a joint selfish step, based on *the same* global information about the cardinalities of color classes. This is assured by the fact that no matter if a vertex performs a selfish step or not, it sends to its children the list of cardinalities that was originally sent by the Judge.

Now, the information about the color changes has to be sent back to the Judge: starting by the leaves, each child sends the information about its new color to its parent and so on, until the root of the tree gets informed about the new cardinalities of all color classes. Then the root sends the updated list to its children and the above procedure is repeated until the Judge gets the same list in two successive steps. In order to avoid loops, if the root observes that the list has changed but the value of the potential function Φ did not change, a message is forwarded to ask the vertices to change back to their original color with probability 1/2.

The above procedure, combined with the results of the previous subsection, guarantees that after at most $O(n \cdot a(G))$ phases (each phase starting with the Judge forwarding the list of cardinalities and ending with the Judge receiving the information about all color changes that have occurred), a proper coloring of $G = (V, E)$ will be produced, using a number of colors satisfying the bounds of Inequality 1. To the best of our knowledge, this is the first distributed implementation of a coloring algorithm that achieves all these bounds. We also note that the complexity bound of $O(n \cdot a(G))$ is rather strict, in the sense that it assumes that no more than $o(1)$ vertices perform a selfish step in the same phase. In Section 4 we provide experimental evidence that, in practice, the number of vertices performing a joint selfish step is large, and therefore the coloring procedure converges faster than the $O(n \cdot a(G))$ bound suggests.

3.3 Self-stabilization

We now present a self-stabilizing version of the distributed vertex coloring algorithm; that is, starting from an arbitrary state (coloring of vertices), it guarantees to converge to a legitimate state (proper coloring) in finite number of steps and to remain in a legitimate set of states thereafter. Essentially the self-stabilizing version of our distributed algorithm can recover from transient faults.

In the sequel we consider that the system can start in any configuration. That is, the color of each vertex can be corrupted. Note that we do not make any assumption on the bound of the corrupted nodes. In the worst case all the nodes in the system may start in a corrupted configuration. However, we assume that the unique identification numbers of the vertices are stored in a read-only memory segment that cannot be affected by transient faults.

A central modification to the algorithm is to force vertices to periodically broadcast their color and id to their neighbors. By doing so they can detect (a) whether the coloring is not proper and (b) if a vertex is unsatisfied. This simple mechanism allows the vertices to detect if a transient fault has occurred or if the algorithm was initiated from an arbitrary state. The periodic broadcasting also guarantees that after all vertices have been satisfied the algorithm will not terminate; it will continue to check whether a selfish step can be taken indefinitely. Therefore, when a transient failure occurs, the procedure will be re-initiated.

When two or more vertices detect that their color is not proper (i.e., the same color is used by another vertex) they locally resolve the conflict by changing their color. Each vertex chooses as a new color the maximum color used in the neighborhood plus its own id (recall that for simplicity colors are represented as integers). This will increase the number of colors used in the neighborhood but will result in a proper coloring in $O(1)$ steps. After selecting the new colors, the vertices notify the Judge about the conflict resolution by using the tree structure. In order to guarantee that the tree structure will be functional despite the transient failures, we replace the tree construction algorithm used in the previous section with a self-stabilizing tree structure (e.g., see [1]).

When the Judge receives notification of a conflict resolution, it waits for a $O(\text{diam}(G))$ period (where $\text{diam}(G)$ is a predefined upper bound of the diameter of G) so that any other conflict is propagated through the tree structure. Then the Judge broadcasts a request message to all vertices to report their color as if they have changed their color during the previous phase. This allows the Judge to recount the color classes and reconstruct the necessary global information. Finally when this reconstruction phase is complete, the new list of colors is broadcast to the vertices to locally check if they wish to conduct a selfish step or not.

The above discussion clearly implies the following:

Lemma 3. *The self-stabilizing version of the distributed vertex coloring algorithm assures that:*

1. *Starting from an arbitrary configuration, eventually all vertices are properly colored.*
2. *Starting from an arbitrary configuration, the Judge is informed about the color of each vertex.*
3. *Starting from an arbitrary configuration, eventually all vertices are informed about the cardinality of each color class.*
4. *Starting from an arbitrary configuration, the algorithm returns to a proper configuration (*convergence*).*
5. *Starting from a proper configuration, the algorithm preserves the proper configuration (*closure*).*

4 Algorithm Engineering

Most of the times in Computer Science, researchers tend to design an algorithm in an abstract way. This happens because an algorithm should be able to be used in many different situations and it is up to the developer to decide the way it should be turned into code for a real system. Almost every time the developer finds many limitations in the ways she can operate within the given hardware and software specifications. These problems are further augmented when implementing algorithms for wireless sensor networks due to the extremely limited resources and also due to the heterogeneous nature (both in terms of hardware and software). Algorithm development for such networks is complex as it unites the challenges of distributed applications and embedded programming.

As a starting point we implemented a centralized version that follows closely the initial design of [12]. We call this algorithm JColoring (noted as JC in the figures). The algorithm was implemented in a real environment that actually exchanged network messages containing protocol payload. The implementation of this centralized version required two sub-protocols that enabled (i) only a single vertex to perform a selfish step each time (mutual exclusion) and (ii) all vertices to know the cardinalities of all color classes (global knowledge).

In this centralized version, the Judge coordinates the graph coloring game by communicating with each vertex sequentially. The Judge sends a message to the next vertex in the sequence containing the list of color classes along with their cardinalities. Upon receiving this message, the vertex has to decide whether to take a selfish step or not and informs the Judge about its new color. Communication between vertices is implemented using the well-established ad-hoc routing protocol TORA [13]. The particular protocol was selected because of its simplicity and ability to operate adequately in wireless settings.

As expected, the evaluation of this version indicated the very poor scalability of the system due to the single point of coordination. It also revealed another major practical problem: in the hardware platform we used to test the algorithm, the available payload of each packet was about 120 bytes. Therefore in a single message we were able to store about 30 colors. So when the number of vertices grew beyond this number, the messages containing the list with the color cardinalities were fragmented in two (or more) packets thus drastically decreasing the available bandwidth and leading to longer execution times.

After achieving a satisfactory state of the centralized version, the next step was to implement our distributed algorithm as described in Section 3. We call this algorithm PJColoring (noted as PJC in the figures). With the data structures and the message fragmentation mechanism in place we proceeded by implementing (i) the simultaneous execution of selfish steps (thus leveraging the mutual exclusion limitation) and (ii) the distributed global information provisioning. Essentially the tree construction algorithm used for efficiently broadcasting the distributed global information replaced the TORA routing algorithm.

The evaluation of our distributed algorithm revealed yet another interesting Algorithm Engineering issue. The convergecast operation for collecting the information about the color changes resulted in a high number of message exchanges. To resolve this issue we implemented an aggregation mechanism. Each parent, after collecting the new colors of its children, it summarizes the information and propagates a single message to its parent. This aggregation is repeated by each parent until the root of the tree is reached. In many cases this technique led to a reduction of up to 50% of message exchanges. In the figures we note this improved version of the algorithm as PJC (with aggregation).

Successful Algorithm Engineering for tiny artifacts requires the validation and evaluation of algorithms in experiments on real networked embedded devices. We ported our distributed algorithm into the real hardware platform as a final step of validation and evaluation of its performance. This immediately revealed a serious problem related to wireless communication, that of medium congestion. The implementation of the simultaneous execution produced a large number of concurrent exchanges and thus message collisions that were unresolved by the MAC. To reduce the collisions we imposed a short random delay before each transmission; yet, some messages were still not delivered properly. This was inevitable due to the non-deterministic behavior of the wireless medium. To completely overcome these problems we implemented the self-stabilizing version of the algorithm leading to a system that was fully operational in real wireless sensor network deployments.

4.1 Implementation Details

We decided to implement our algorithms using Wiselib [2]: a code library, that allows implementations to be OS-independent. It is implemented based on C++ and templates, but without virtual inheritance and exceptions. Algorithm implementations can be recompiled for several platforms and firmwares, without the need to change the code. Wiselib can interface with systems implemented using C (Contiki), C++ (iSense), and nesC (TinyOS).

Furthermore, an important feature of Wiselib is the already implemented algorithms and data structures. Since different kind of hardware uses different ways to store data (due to memory alignment, inability to support dynamic memory, etc.), it is important to use these safe types as much as possible since they have been tested before on most hardware platforms. As of mid 2010, the Wiselib includes about 40 Open Source implementations of standard algorithms, and is scheduled to grow to 150-200 algorithms by the end of 2011. We use Wiselib data structure to implement the maps for counting the cardinalities of the colors and the implementation of TORA algorithm.

Finally, Wiselib also runs on the simulator Shawn [8], hereby easing the transition from simulation to actual devices. This feature allows us to validate the faithfulness of our implementation and also get results concerning the quality of our algorithms without time consuming deployment procedures and harsh debugging environments. Shawn allows repeatability of simulations in an easy way by using only a single configuration file. It provides many options such as packet loss, radius of communication, ways of communicating and even mobility in an abstract way, without needing to provide specific code for every hange.

4.2 Performance Evaluation: Simulations

We start by presenting the results of the validation and performance evaluation based on simulated executions of our implementation in the Shawn environment. Using different types of networks and experimenting with the different parameters, we were able to identify that parameters that influence the correctness and performance of our protocol. The first type of network topologies (called “Fixed Diameter”), deploys $n = [25, 250]$ vertices in a way such that the network diameter remains constant as n increases: the resulting networks have $\text{diam}(G) = 6$ and the average vertex degree (which we call “density”) is $2 \leq \text{avgDeg}(G) \leq 40$. The second type of network topologies (called “Fixed Density”), deploys $n = [10, 230]$ in a way such that the average vertex degree remains constant as n increases: the resulting networks have $3 \leq \text{diam}(G) \leq 15$ and $\text{avgDeg}(G) = 12$.

A very important performance metric is the number of colors used. Figure 1 depicts the number of colors used for each set of topologies. As expected, as the density of the network increases (i.e., the average vertex degree), the chromatic number increases and hence the number of colors used by our algorithm.

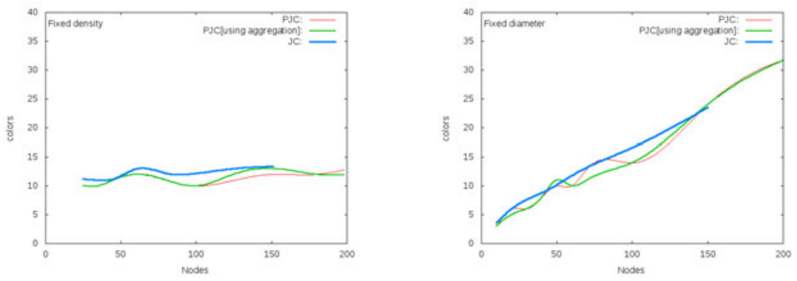

Fig. 1. Colors used for both types of network topologies

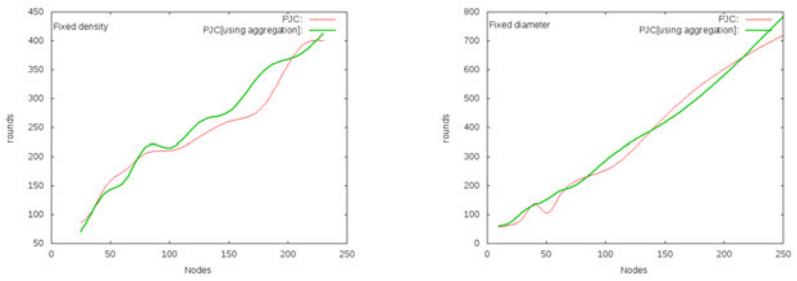

Fig. 2. Time complexity for both types of network topologies

When the density remains more or less constant, the chromatic number remains constant, as also does the number of colors used by the algorithm.

The next algorithm property that we evaluate is the time complexity. Figure 2 depicts the number of rounds required to reach a Nash equilibrium. It is evident that as the number of vertices increases, so does the number of rounds required. However, the "Fixed Diameter" topologies seem to be harder to address. This is caused due to the fact that only few vertices can change color in each step of the algorithm in dense environments. As a result the algorithm takes more steps until it reaches an equilibrium. These topologies require about twice the number of rounds than the "Fixed Density" topologies. In these figures we have not included the results for the JColoring algorithm as they cannot be fit properly: in all cases the resulting rounds are about 20 times more than the PJColoring algorithm and thus even for $n = 50$ it becomes totally impractical.

In the sequel we examine the communication complexity of the algorithms. Figure 3 shows the number of message exchanges for each set of topologies considered. The behavior of the algorithms are similar to that observed in the previous figure: (a) as the number of vertices increases, so does the number of messages exchanged, (b) the "Fixed Diameter" topologies seem to be harder to address and (c) the JColoring exchanges much greater messages than the PJColoring algorithms. In these figures it is clear that for $n > 50$ the JColoring algorithm becomes impractical, while PJColoring performs well for all the topologies examined.

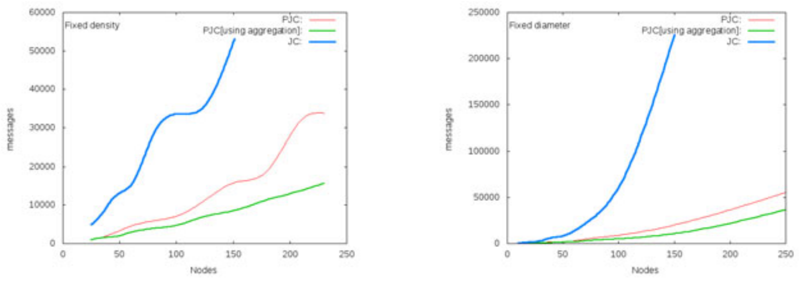

Fig. 3. Communication complexity for both types of network topologies

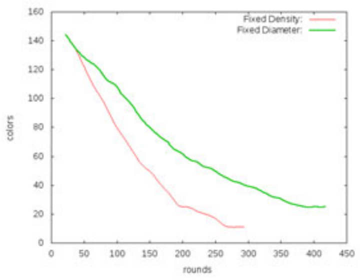

Fig. 4. Colors used as the simulation evolves for PJColoring (with aggregation) for different type of topologies of fixed number of vertices ($n = 150$)

We conduct a final set of simulated executions to evaluate the ability of PJColoring (with aggregation) to perform parallel selfish steps. Figure 4 depicts the number of colors used as the network evolves over time. We present the results for both types of topologies when $n = 150$. We observe that in both cases the algorithm reaches the equilibrium fast, although in the denser topology the progress is slower. It is evident that the algorithm succeeds in parallelizing the process of selfish coloring.

4.3 Performance Evaluation: Experiments

Although simulations enable both verification and performance evaluation of protocols, they do not take into account read-world effects that may change their behavior. Also, in physical deployment protocols must operate within the given hardware and software specifications (i.e., memory and message payload constrains). Thus we continue the evaluation of the PJColoring (with aggregation) algorithm in our experimental testbed that consists of iSense sensor nodes with a 32 Bit RISC Controller running at 16 MHz, with 96 KB RAM and 128 KB Flash and an IEEE 802.15.4 compliant wireless radio interface. All nodes come with a permanent energy source allowing for an arbitrary duty cycle. Nodes are equipped with a multitude of sensors measuring light, temperature, humidity, acceleration, magnetic-field levels and barometric pressure. For more details on the installation and technical aspects of the experimental testbed see [3]. In the sequel we consider that each node represents a single vertex.

The first set of experiments was conducted in networks where all the nodes were within communication range of each other, thus forming a clique. In Tab. 1 we can see the breakdown of the total execution time of the protocol. The selection of this topology is twofold. Firstly, to determine the effect of network density in the execution of our protocol (i.e., collisions, etc.). We observed that up to networks sizes of eight nodes the results were always correct (i.e., messages were delivered properly), while in larger networks message loses always occurred forcing the non-stabilizing version of the protocol to produce not consistent results. Secondly, to determine the real execution time of a protocol phase. In these topologies the protocol terminates immediately after the initial assignment of colors, i.e., after checking that every available color is taken by a neighbor. For all topologies considered the execution time of a single phase is about $130ms$.

Table 1. Real execution times of PJColoring (with aggregation) for clique topology

Nodes	Tree Construction	PJColoring	Total
5	745 ms	132 ms	877 ms
6	739 ms	133 ms	872 ms
7	754 ms	141 ms	895 ms
8	752 ms	137 ms	889 ms

The second set of experiments was conducted in networks where the nodes where positioned in a line. Tab. 2 lists the total execution time of the protocol. This topology produces networks of increasing diameter thus requiring more time to construct the broadcast tree and complete the convergecast operation. Based on the results, the initialization of the two processes requires about $750ms$, and then for each further hop an additional time of about $300ms$ was required. The initialization time of the two processes is also confirmed by the previous set of experiments that used single-hop networks (see Tab. 1). In these topologies, we observe that the time required by the selfish steps is also increasing with the network size. It seems that for each additional hop an additional time of $600ms$ is required, or, based on the previous set of experiments, about 4 protocol phases. By carefully examining the particular topology, it is evident that the addition of a new node forces all other nodes to take additional selfish steps until a Nash equilibrium is reached.

The performance evaluation we conducted both in simulated and experimental environments indicates interesting aspects of the vertex coloring problem. It also indicates that our solution successfully raises the limitation of [12] where only one vertex at a time is allowed to perform a selfish step in order to guarantee polynomial time convergence into a Nash equilibrium coloring. Our distributed algorithm allows non-neighboring vertices to perform a selfish step in parallel, thus drastically reducing time complexity. The Algorithm Engineering process

Table 2. Real execution times of PJColoring (with aggregation) for line topology

Nodes	Tree Construction	PJColoring	Total
2	736 ms	138 ms	1389 ms
3	1032 ms	728 ms	1760 ms
4	1298 ms	1324 ms	2622 ms
5	1756 ms	2011 ms	3767 ms

led to a practical algorithm that can be executed in real networks and, interestingly, further reduces communication complexity while preserving the time complexity and the number of colors used.

References

1. Afek, Y., Kutten, S., Yung, M.: Memory-efficient self stabilizing protocols for general networks. In: Toueg, S., Kirousis, L.M., Spirakis, P.G. (eds.) WDAG 1991. LNCS, vol. 579, pp. 15–28. Springer, Heidelberg (1992)
2. Baumgartner, T., Chatzigiannakis, I., Fekete, S., Koninis, C., Kröller, A., Pyrgelis, A.: Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks. In: Silva, J.S., Krishnamachari, B., Boavida, F. (eds.) EWSN 2010. LNCS, vol. 5970, pp. 162–177. Springer, Heidelberg (2010)
3. Chatzigiannakis, I., Koninis, C., Mylonas, G., Colesanti, U., Vitaletti, A.: A peer-to-peer framework for globally-available sensor networks and its application in building management. In: DCOSS/IWSNE 2009 (2009)
4. Escoffier, B., Gourves, L., Monnot, J.: Strategic Coloring of a Graph. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 155–166. Springer, Heidelberg (2010)
5. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. Journal of Computer and System Sciences 57(2), 187–199 (1998)
6. Halldórsson, M.: A still better performance guarantee for approximate graph coloring. Information Processing Letters 45, 19–23 (1993)
7. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
8. Kröller, A., Pfisterer, D., Buschmann, C., Fekete, S.P., Fischer, S.S.: A new approach to simulating wireless sensor networks. In: DASD 2005, pp. 117–124 (2005)
9. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
10. Moscibroda, T., Wattenhofer, R.: Coloring Unstructured Radio Networks. In: SPAA 2005, pp. 39–48 (2005)
11. Nash, J.F.: Non-cooperative games. Annals of Mathematics 54(2), 286–295 (1951)
12. Panagopoulou, P.N., Spirakis, P.G.: A game theoretic approach for efficient graph coloring. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 183–195. Springer, Heidelberg (2008)
13. Park, V.D., Corson, M.S.: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In: INFOCOM 1997, pp. 1405–1413 (1997)
14. Schneider, J., Wattenhofer, R.: Coloring unstructured wireless multi-hop networks. In: PODC 2009, pp. 210–219 (2009)