Distributed Sleep Scheduling in Wireless Sensor Networks via Fractional Domatic Partitioning

André Schumacher and Harri Haanpää

Helsinki University of Technology, Department of Information and Computer Science,
P.O. Box 5400, FI-02015 TKK, Finland
Andre.Schumacher@tkk.fi, Harri.Haanpaa@tkk.fi

Abstract. We consider setting up *sleep scheduling* in sensor networks. We formulate the problem as an instance of the *fractional domatic partition problem* and obtain a distributed approximation algorithm by applying linear programming approximation techniques. Our algorithm is an application of the Garg-Könemann (GK) scheme that requires solving an instance of the minimum weight dominating set (MWDS) problem as a subroutine. Our two main contributions are a distributed implementation of the GK scheme for the sleep-scheduling problem and a novel asynchronous distributed algorithm for approximating MWDS based on a primal-dual analysis of Chvátal's set-cover algorithm. We evaluate our algorithm with `ns2` simulations.

1 Introduction

In *sleep scheduling*, sensor-network nodes switch between active and inactive states to save energy, thus extending network lifetime. A variety of protocols have been proposed for having a sensor network self-organize by choosing subsets of nodes to be active and serve as a *backbone* for routing or providing coverage; see e.g. [1,2,3,4]. Many protocols are heuristic and do not provide performance guarantees.

The sleep-scheduling problem can be modeled using a pairwise redundancy relationship between sensor nodes. In the resulting *redundancy graph* adjacent nodes represent sensors that can measure the same data. When backbone connectivity is not a concern, e.g., because data generation and collection phases are separated, the network can be considered operational as long as at any time each inactive node has an active neighbor in the redundancy graph. Although our coverage model is very simple, we consider it useful when node density is rather large, so that nodes nearby typically measure similar data. In graph-theoretic terms, the problem reduces to finding dominating sets in the redundancy graph and computing an assignment of dominating sets to time slots that achieves maximum length while satisfying node-battery constraints. This notion of sleep scheduling assumes a global clock to determine at any time the set of active nodes. It is usually sufficient, however, that nodes are loosely synchronized.

Floréen et al. [5] and Suomela [6] use the same redundancy model. Cărbunar et al. [7] consider a geometric setting where nodes have a fixed sensing radius.

R. Guerraoui and F. Petit (Eds.): SSS 2009, LNCS 5873, pp. 640–654, 2009.

They construct a graph of nodes which each individually could become inactive without sacrificing coverage. By introducing edges between nodes that share an edge in the *Voronoi diagram* and searching for large independent sets in this graph, battery capacity of some nodes can be preserved while retaining network-wide coverage.

Assuming uniform battery capacities and a node can only participate in one dominating set during the operation of the network, the sleep-scheduling problem is also known as the *domatic partition problem*. In the version we consider, dominating sets can be active for an arbitrarily long period while satisfying battery constraints. Removing integrality constraints enables us to apply approximation techniques for Linear Programming (LP) and allows for a longer lifetime in some networks, such as the five-cycle with unit capacities.

In Section 2, we formulate the sleep-scheduling problem as an LP packing problem and apply the Garg-Könemann [8] (GK) technique to obtain a distributed approximation algorithm for general redundancy graphs. General redundancy graphs are interesting for sensor networks, as other models, such as *unit-disk graphs*, do not capture non-uniform sensing capabilities or obstacles in the terrain. We also present a novel asynchronous distributed algorithm for approximating the *minimum weight dominating set* (MWDS) problem, which we will then use within the GK scheme. We develop our algorithm in Section 3 and first describe a centralized implementation. In Section 4 we present an efficient distributed implementation which does not require network-wide clock-synchronization. Further, in Section 5 we provide `ns2` [9] simulation data, which indicate a low number of messages required in practice. Section 6 presents our conclusions.

2 Domatic Partition

The domatic partition problem is a well-known problem in graph theory. The maximum number of disjoint dominating sets of a graph is called the *domatic number*. Feige et al. [10] show that the domatic number can be approximated in polynomial time within a factor of $O(\log n)$, where n is the number of nodes, but that it is hard to approximate it within a $(1-\epsilon)\ln n$ factor for any $\epsilon > 0$. Moscibroda and Wattenhofer [11] extend the results of Feige et al. and obtain a distributed, randomized algorithm for the same problem.

In this section, we give a formal description of the sleep-scheduling problem that allows arbitrary activation periods and formulate it as an LP. We then describe the application of the Garg-Könemann algorithm and a distributed implementation that is suitable for wireless sensor networks. For simplicity, we assume that all nodes have unit battery capacity. We note, however, that the extension to arbitrary capacities is possible.

2.1 Problem Formulation

We assume a given connected *transmission graph* $G(V, E)$ that models the sensor network with unique node identifiers. The edges in E represent the links between

the radio nodes, which we assume to be undirected. Denote by $N(v)$ the neighbors of v in G and define $N^+(v)$ to be the *extended neighborhood* $N(v) \cup \{v\}$ of v. Define $\delta = \min_{v \in V} N(v)$ and $\delta^+ = \min_{v \in V} N^+(v)$ to be the minimum degree and minimum extended degree, respectively. Similarly, define Δ and Δ^+ to be size of the largest neighborhoods. By $N_k^+(v)$ we denote the k-hop extended neighborhood of v, i.e., all nodes at a hop-distance of at most k from v and define $N_0^+(v) = \{v\}$, so that $N_1^+(v) = N^+(v)$.

For simplicity of exposition, we consider the redundancy graph and G to be identical, so that the problem involves finding dominating sets in G. This assumption could be removed, when nodes know their neighbors in the redundancy graph and can communicate with them over only a few hops in G. However, note that we do not require any specific structural properties on either graph.

We introduce variables x_D that correspond to the total activation time of dominating set D. The domatic partition problem can be formulated as the following LP with a possibly exponential number of variables.

FRAC_DOMPART_PRIMAL

$$\begin{aligned} \max \quad & \sum_D x_D \\ \text{s.t.} \quad & \sum_{D: v \in D} x_D \leq 1 \quad \forall v \in V \qquad (1) \\ & x_D \geq 0 \quad \forall D \end{aligned}$$

The objective $\sum_D x_D$ is the length of the sleep schedule and (1) is the capacity constraint for node v. From (1) and since there is a node that can be dominated by at most δ^+ different dominating sets it follows that δ^+ is an upper bound on the total lifetime of any feasible solution. For the domatic number problem x_D must be integral. As we assume that nodes can participate in several dominating sets, we do not require integrality of x_D. This problem has only been rarely addressed in the literature. It was shown in [6] that the hardness of approximation result of [10] for the domatic number problem also holds in this case. Floréen et al. [5] propose a local algorithm that achieves a constant approximation factor in so-called *marked* graphs, which are bounded-degree graphs that contain specially distributed marked nodes for breaking symmetry. Since we consider general graphs we can only aim at a logarithmic approximation factor.

We propose a distributed version of the Garg-Könemann scheme for approximating LP packing problems, which requires a solution to the following dual problem. The dual is formulated by introducing dual variables y_v for the capacity constraints of node v.

FRAC_DOMPART_DUAL

$$\begin{aligned} \min \quad & \sum_v y_v \\ \text{s.t.} \quad & \sum_{v \in D} y_v \geq 1 \quad \forall D \qquad (2) \\ & y_v \geq 0 \quad \forall v \in V \end{aligned}$$

Validating constraint (2) for given y_v corresponds to solving an instance of the minimum weight dominating set (MWDS) problem with y_v as constant node

weights. Our algorithm for approximating MWDS does not require network-wide synchronization or geometric restrictions on the dependency graph. Suomela [6] applies the GK scheme in a centralized setting to so-called *local graphs*, where $V \subseteq \mathbb{R}^d$, all edges have length at most 1 and node density is bounded by a constant. It was shown in [6] that the MWDS problem in these graphs can be solved efficiently. Berman et al. [12] propose to use the greedy set-cover approximation algorithm by Chvátal [13] within the GK scheme. Although their approach is similar to ours, their algorithm is centralized. See also [4] for a survey of algorithms for variations of lifetime maximization problems within the context of *sensor network coverage*, which can be also seen as heuristics for problems similar to domatic partition.

2.2 Garg-Könemann Scheme

For simplicity of exposition, we first describe the GK scheme as applied to problem FRAC_DOMPART_PRIMAL in a centralized setting and then elaborate on a distributed version suitable for implementation in sensor networks. The GK scheme takes as input an LP packing problem and a small positive constant ϵ. After termination the primal objective value is guaranteed to be at least $(1-\epsilon)^2$ times the optimum (for details see [8]). The algorithm proceeds in iterations, as described in Algorithm 1.

---

initially :
  $\beta \leftarrow (1+\epsilon)((1+\epsilon)L)^{-1/\epsilon}$
  for all D: $x_D(0) \leftarrow 0$
  for all $v \in V$: $y_v(0) \leftarrow \beta$

in iteration $k \geq 1$
  use oracle to find MWDS D^* using $y_v(k-1)$ as node weights
  if $(\sum_v y_v(k-1) \geq 1)$
    for all D: $x_D(k) \leftarrow \frac{x_D(k-1)}{\log_{1+\epsilon} \frac{1+\epsilon}{\beta}}$
    return
  else
    $x_{D^*}(k) \leftarrow x_{D^*}(k-1) + 1$
    for all $v \in V$
      if $v \in D^*$ then $y_v(k) \leftarrow (1+\epsilon) y_v(k-1)$
      else $y_v(k) \leftarrow y_v(k-1)$

Algorithm 1. GK scheme for fractional domatic partition

Denote by $y_v(k-1)$ the value of the dual variable y_v at the beginning and by $y_v(k)$ its value at the end of iteration k and define $x_D(k)$ similarly. In iteration k one selects the MWDS D^* depending on the current node weights and increases its activation time by one. If the total weight of all nodes in the networks is at least one, the primal variables are scaled down by a value depending on the size of the instance, and the algorithm terminates. Otherwise, the dual variables

for the nodes in D^* are multiplied by $(1+\epsilon)$. For the value L in the scaling factor it is sufficient to choose $L = |V|$, the maximum size of any dominating set. Note that the dominating sets found in different iterations do not need to be disjoint.

Instead of solving the MWDS subproblem in each iteration exactly, we use an approximation oracle with approximation factor $\phi > 1$. The resulting sleep schedule is guaranteed to have a length of at least $(1-\epsilon)^2/\phi$ times the optimal length, where ϵ can be chosen arbitrarily small. For details on the application of the GK scheme in combination with an approximation oracle see the paper by Tsaggouris and Zaroliagis [14].

In Section 3 we propose a distributed MWDS algorithm with an approximation factor of $\phi = O(\ln \Delta^+)$, so that the combined algorithm is asymptotically optimal for the sleep-scheduling problem. By choosing a different MWDS approximation algorithm it is likely that better approximation guarantees can be achieved for certain graph classes.

2.3 Distributed GK Implementation

We now describe a distributed implementation of Algorithm 1 and how we combine it with the MWDS subroutine of Section 4. We assume the existence of a single initiator node which knows the number of nodes $|V|$.

First we construct a spanning tree of the transmission graph in style of the *Shout* protocol [15]. The spanning tree is used to send and receive control messages within the network. While constructing the spanning tree, the node weights are initialized to β, as in Algorithm 1.

In the first iteration, the initiator node broadcasts an initiate message in the network. This is a signal for the nodes to solve the subproblem within the inner loop of the GK algorithm, in our case the MWDS problem. The nodes then solve the subproblem, and a convergecast follows, whereby the initiator obtains the sum of the weights $y_v(0)$ that is needed for testing the termination condition.

Until the termination condition is met, in subsequent iterations the nodes update their weight y_v according to the solution of the subproblem in the previous iteration. In our case, the nodes found to be in the dominating set in the previous iteration set $y_v \leftarrow (1+\epsilon)y_v$ before solving the MWDS problem in the current iteration. When the termination condition is satisfied, the initiator broadcasts a final message to inform the other nodes of the termination.

In our implementation, nodes need to remember the iterations in which they were in the dominating set. The sleep schedule results from this information. As an implementation note, during the broadcast and convergecast in each iteration, we let the nodes collect some data they need for the MWDS computation. Namely, at the start of the MWDS algorithm the nodes have to know not only their own weight but also the weights of their neighbors, as well as the number of neighbors each neighbor has. Instead of having separate phases for collecting this data, this is convenient to embed in the GK scheme.

3 Minimum Weight Dominating Set Approximation

This section describes an approximation algorithm for MWDS inspired by parallel algorithms based on linear programming duality proposed by Rajagopalan and Vazirani [16] for weighted set-cover. Although we use it within the GK scheme, we consider the more general MWDS setting. Dominating sets can be used among other purposes for network coverage, routing, and sleep scheduling. For an overview of the relevant literature see [17].

3.1 Problem Formulation

We first formulate an LP for the problem. Introduce variable z_v for each v corresponding to v being selected for the dominating set, whereby we initially do not require integrality of z_v. Denote the weight of node v by w_v.

FRAC_DOMSET_PRIMAL

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v \, z_v \\ \text{s.t.} \quad & \sum_{u \in N^+(v)} z_u \geq 1 \quad \forall v \in V \\ & z_v \geq 0 \quad \forall v \in V \end{aligned}$$

Algorithm 2 is Chvátal's algorithm applied to MWDS that obtains an integral solution to the previous LP. It repeatedly adds to the dominating set the node with the lowest ratio of weight to *span*, the number of uncovered nodes that the node would cover, until all nodes are covered. The algorithm gives a dominating set with weight at most $\phi = H_{\Delta^+}$ times the optimum, where $H_i = \sum_{j=1}^{i} j^{-1}$ is the i-th *harmonic number*. This follows from the results in [13], as Δ^+ is the size of the largest set in the corresponding set-cover instance.

initially :
  $C \leftarrow \emptyset$
  for all $v \in V$: $z_v \leftarrow 0$

while $C \neq V$
  $v' \leftarrow \arg\min_v \frac{w_v}{|N^+(v) \setminus C|}$
  $z_{v'} \leftarrow 1$
  $C \leftarrow C \cup N^+(v')$

Algorithm 2. Greedy algorithm MWDS based on [13]

When one assumes unit node weights, one can easily obtain approximation algorithms that achieve a constant approximation factor in unit-disk graphs [18], and even polynomial-time approximation schemes (PTAS) are possible [19]. In general graphs, however, the inapproximability results for the set-cover problem [20] imply that Chvátal's algorithm is essentially the best-possible polynomial time approximation algorithm under standard complexity assumptions.

Distributed algorithms based on Chvátal's algorithm have been proposed for both unit and arbitrary weights. Most of them, however, assume a synchronous message passing model. Jia et al. [21] remark that the straightforward distributed implementation of the greedy algorithm in the synchronous model has linear time complexity. They propose randomized algorithms with polylogarithmic time complexity and approximation guarantees similar to Chvátal's algorithm, but their implementation requires careful clock synchronization in the network. Alternatively, synchronization techniques proposed by Awerbuch [22] can be applied, which further complicate the algorithm and require message overhead. Our algorithm is deterministic, requires no synchronization, is simple to implement, and shares the approximation guarantee of Chvátal's algorithm.

Wang et al. [23] propose a distributed asynchronous algorithm for connected MWDS based on a hybrid approach between the independent set approach [18] and Chvátal's algorithm applied locally in each neighborhood. However, for general graphs the approximation guarantee in [23] can be worse than for Chvátal's algorithm and may further depend on the weights of adjacent nodes.

3.2 Centralized Implementation

We first describe our algorithm for approximating MWDS in a centralized setting. Introduce dual variables α_v for the coverage constraint of node v in problem FRAC_DOMSET_PRIMAL. We formulate the dual as follows.

FRAC_DOMSET_DUAL

$$\begin{aligned} \max \quad & \sum_{v \in V} \alpha_v \\ \text{s.t.} \quad & \sum_{u \in N^+(v)} \alpha_u \leq w_v \quad \forall v \in V \\ & \alpha_v \geq 0 \quad \forall v \in V \end{aligned}$$

Algorithm 2 can be translated into an algorithm that maintains a pair of primal and dual solutions. The primal solution is initially infeasible and becomes feasible at termination. The dual solution is initially feasible but may become infeasible. However, as one is able to bound the maximum dual constraint infeasibility, a dual feasible solution is obtained by the technique of dual fitting [24].

The algorithm is best explained in its continuous version. Denote by $z(t)$ and $\alpha(t)$ the value of a pair of primal and dual solutions at time t respectively (not necessarily feasible). At start, $z(0) = \alpha(0) = 0$. Start increasing all $\alpha_v(t)$ at unit rate until the first dual constraint holds with equality, say the constraint for z_v. This happens at time $t_1 = w_v / |N^+(v)|$, so the node first chosen has the least ratio of weight to span. Fix $z_v(t) = 1$ for $t \geq t_1$ and for all $v' \in N^+(v)$ let $\alpha_{v'}(t) = 0$ for $t > t_1$. Keep raising the other dual variables and proceed as before, breaking ties arbitrarily. As $\alpha_{v'}(t) = 0$ for all t after v' was covered, they no longer contribute to the dual constraints. The order in which these get tight is exactly the same in which Algorithm 2 adds nodes to the dominating set, and each node is chosen at a time that equals its weight divided by the number of its uncovered neighbors. Assume that k nodes got tight at time points $t_1, \ldots, t_k$.

One can show the following pair of primal and dual solutions is feasible and at most a factor of H_{Δ^+} apart, therefore establishing the approximation guarantee based on weak duality.

$$z_v = z_v(t_k)\ \forall v \in V, \quad \alpha_v = \frac{1}{H_{\Delta^+}} \max_{t_1,\ldots,t_k} \alpha_v(t_i)\ \forall v \in V$$

4 Distributed MWDS Approximation

In this section we obtain a distributed approximation algorithm for the MWDS problem from the centralized dual-increase algorithm described above. Although the order in which nodes enter the dominating set can be different, the resulting dominating set is guaranteed to be the same. We assume that each node is aware of the weight and degree of all its neighbors and also knows its neighbors in a spanning tree rooted at the initiator by executing the steps of Section 2.3.

The previously described algorithm is only feasible in a strictly synchronized setting, since nodes need to increase their α_v variables uniformly. We now describe a voting scheme that does not require synchronization. After termination each node knows all dominators in its one-hop neighborhood. We first explain the basic ideas of the algorithm in 4.1 and then describe it in more detail in 4.2.

4.1 Algorithm Outline

Throughout the algorithm, each node is in one of three *cover states*: *uncovered*, *covered*, or *dominator*. Denote by U the set of uncovered nodes, where initially $U = V$. Each node v maintains its own *price*

$$p_v := \begin{cases} \frac{w_v}{|N^+(v) \cap U|} & \text{if } N^+(v) \cap U \neq \emptyset, \\ \infty & \text{otherwise.} \end{cases}$$

In the continuous time version, node v would become a dominator at time p_v if the set of its uncovered neighbors stayed unchanged until then. To estimate p_v node v must know the state of its neighbors.

The straightforward method of repeatedly having each node compute its price and letting the node with the minimum p_v in the network become a dominator would be inefficient. Instead, it suffices to consider two-hop local neighborhoods only. The crucial observation is that as the algorithm proceeds, p_v can only increase, as the number of uncovered neighbors can only decrease. Thus, if node v has the minimum p_v in $N_2^+(v)$, it is guaranteed to become a dominator at time p_v, since $N^+(v) \cap U$ stays unchanged until then. Then the idea of the distributed algorithm is clear: whenever node v has the minimum p_v in $N_2^+(v)$, add it to the set of dominators and let its neighbors know they are dominated.

During the algorithm each node v monitors whether it has the minimum p_v in $N_2^+(v)$. If so, v declares itself a dominator and informs its neighbors, who then mark themselves as covered. A node becoming covered may affect the prices of

its neighbors, as the prices depend on the number of uncovered neighbors. The algorithm terminates when all nodes are covered.

Each uncovered node v monitors the weights of the nodes in $N^+(v)$ and votes for the neighbor with the lowest price. If some node receives votes from all of its uncovered neighbors, and there is at least one of those, it has the lowest price in $N_2^+(v)$ and may therefore declare itself dominator.

To reduce the number of messages, when u votes for v, it also informs v of a limit; the vote is valid as long as the price of v does not exceed the limit. When u votes, it votes for the neighbor with the lowest price and sets the limit to that of the neighbor with the second-lowest price. If v raises its price above the limit, it will notify u so that u can decide again which node to vote for.

As a technical point, nodes only inform the nodes that are currently voting for them about price updates. If a node receives a vote with a limit that is lower than the current price of the recipient (e.g., if the voter has old information about the price of the recipient), then the recipient will reply by informing the voter of its current price.

4.2 Voting Scheme

We now describe the distributed implementation given in Algorithm 3. Each node v keeps a tuple $\mathrm{NL}_u = (\mathrm{id}, \mathrm{weight}, \mathrm{degree}, \mathrm{span}, \mathrm{limit}, \mathrm{notify})$ for each $u \in N^+(v)$, which together form the *neighbor list* NL, where $\mathrm{id} = u$, *degree* is the degree of u, *span* is the number of uncovered nodes in $N^+(u)$, *limit* is the highest price limit received in any vote from u for v, and *notify* is a boolean variable which indicates whether u needs to be notified of a change in the price of v. The list is kept in increasing order of price, where ties are broken using node identifiers. Additionally, v maintains a set $U(v) \subseteq N^+(v)$ of uncovered neighbors, a set $D(v) \subseteq N^+(v)$ of neighbors that have become dominators, and a set $S(v) \subseteq N^+(v)$ of *supporters* of v, i.e., neighbors that are voting for v.

Initially, $\mathrm{NL}_u = (u, w_u, \delta_u, \delta_u + 1, 0, \mathrm{false})$ for all $u \in N^+(v)$, where δ_u is the degree of u, and v calculates its price $p_v = \frac{w_v}{\delta_v + 1}$. Node v also initializes $D(v)$, $S(v)$ and $U(v)$ accordingly. After receiving an initialization message, node v votes for the node at the head of the list NL. We denote the kth entry of the list by $\mathrm{NL}(k)$, where $1 \leq k \leq |N^+(v)|$. So v sends the message VOTE(limit) to the node with id $\mathrm{NL}(1)[\mathrm{id}]$, say u, where $\mathrm{limit} = p_{\mathrm{NL}(2)[\mathrm{id}]}$. Note that $|N^+(v)| > 1$ because G is connected. When u receives the vote, it first checks whether the vote is valid, i.e., it checks whether $\mathrm{limit} \geq p_u = \frac{w_u}{\delta_u + 1}$. If it is valid, u records v entering its set of supporters $S(u)$ and stores the limit value in its local neighbor list. If v and u are the same node, node v performs exactly the same changes in its own neighbor list without transmitting the message.

Whenever v receives a valid vote or if one of its neighbor was covered, it checks whether there is at least one uncovered node in $N^+(v)$ which also votes for v. If so, i.e., if $S(v) = U(v)$ and $|S(v)| > 0$, then v declares itself dominator and informs all its neighbors with a DOMINATOR($N(v)$) message. It includes the ids of its one-hop neighbors to let each recipient $u \in N(v)$ update its own price based on the number of nodes in $N^+(v) \cap N^+(u)$ that were covered by v.

```
initially :
  D(v) ← ∅; U(v) ← N+(v); S(v) ← ∅; NL ← ∅
  for all u ∈ N+(v)
    NL ← NL ∪ (id: u, weight: w_u, degree: δ_u, span: δ_u + 1, limit: 0, notify: false)
  schedule price_update_timer() after T1 seconds
  cast_vote() after T2 seconds

if v receives VOTE(limit) from u
  if (NL_v[weight]/NL_v[span] < limit) // vote is valid
    S(v) ← S(v) ∪ {u}
    if (NL_u[limit] < limit) NL_u[limit] ← limit
    if (check_all_covered_and_voted()) declare_myself_dominator()
  else send PRICE(NL_v[span], v ∈ U(v) ? UNCOVERED : COVERED) to u

if v receives PRICE(new_span, new_state) from u // (either overheard or unicast)
  old_first ← NL(1)
  if (u ∈ U(v) and new_state == COVERED) // u informs of becoming dominated
    U(v) ← U(v) \ {u}
    NL_v[span] ← NL_v[span] − 1
    for all w ∈ S(v) do
      if (NL_w[limit] < NL_v[weight]/NL_v[span])
        NL_w[notify] ← true
        S(v) ← S(v) \ {w}
  NL_u[span] ← new_span
  if (v ∉ D(v) and check_all_covered_and_voted()) declare_myself_dominator()
  else check_and_terminate()
  if (v ∈ U(v))
    if ( old_first != NL(1)[id] or (old_first == u and message was unicast))
      cast_vote()
    if ( old_first == NL(1)[id] and old_first == v)
      NL_v[limit] ← NL(2)[weight]/NL(2)[span]
      if (check_all_covered_and_voted()) declare_myself_dominator()

if v receives DOMINATOR(N(u)) from u
  D(v) ← D(v) ∪ {u}
  NL_u[span] ← 0
  if (|U(v) \ N+(u)| < NL_v[span])
    NL_v[span] ← |U(v) \ N+(u)|
    for w ∈ S(v) with NL_w[limit] < NL_v[weight]/NL_v[span]
      NL_w[notify] ← true
      S(v) ← S(v) \ {w}
  if (v ∈ U(v))
    for all w ∈ N(v) \ (N+(u) ∪ D(v))
      send PRICE(NL_v[span], COVERED) to w
      NL_w[notify] ← false
  U(v) ← U(v) \ N+(u)
  if (v ∉ D(v) and check_all_covered_and_voted()) declare_myself_dominator()
  else check_and_terminate()
```

Algorithm 3. Distributed MWDS as executed by node v

```
void function cast_vote() at v
    limit ← NL(2)[weight]/NL(2)[span]
    if (NL(1)[id] ≠ v)
        send VOTE(limit) to NL(1)[id]
    else
        NL_v[limit] ← limit
        S(v) ← S(v) ∪ {v}
        if (check_all_covered_and_voted() and v ∉ D(v))
            declare_myself_dominator()

void function declare_myself_dominator() at v
    D(v) ← D(v) ∪ {v}
    U(v) ← ∅
    for all u ∈ N(v)
        send DOMINATOR(N(v)) to u
    NL_v[span] ← 0
    stop price_update_timer()
    check_and_terminate()

bool function check_all_covered_and_voted() at v
    if (S(v) = U(v) and |S(v)| > 0) return true
    else return false

void price_update_timer() at v
    for all u ∈ U(v) with NL_u[notify] == true
        NL_u[notify] ← false
        if (NL_u[limit] < NL_v[weight]/NL_v[span])
            send PRICE(NL_v[span], v ∈ U(v) ? UNCOVERED : COVERED) to u
    if (v ∉ D(v)) schedule price_update_timer() after T_1 seconds

void check_and_terminate() at v // test for local termination, perform convergecast
    if (U(v) = ∅ and all child nodes in spanning tree have reported weight of their subtree
    for current GK iteration)
            send terminate message to parent, include sum of weights of local tree branch
```

Algorithm 3. (*Continued*)

If v receives from u an invalid VOTE(limit) (with limit $< p_v$), then v replies with PRICE(span, state), informing of its current span and cover state instead of recording the limit for u. The set $S(v)$ remains unchanged in this case. When receiving the reply, u updates the span and state for v in its local memory. This update may initiate a price update to be transmitted by u if v indicated it is covered but $v \in U(u)$ prior to receiving the price update from v.

When the price of a node v changes because the number of uncovered nodes in $N(v)$ decreases, v goes through its set of supporters and sets the notify flag for those nodes $u \neq v$ that are required to leave $S(v)$ because v's price just exceeded the limit $\mathrm{NL}_u[\mathrm{limit}] < p_v$. To these neighbors v later sends a price update PRICE(span, state).

Upon receiving a price update, each uncovered node v checks whether the lowest-price entry NL(1) has changed. Let u be the neighbor with the former lowest entry and let $u \neq v$. If it has changed, i.e., if $u \neq \text{NL}(1)[\text{id}]$, then v sends a vote to the new best entry as described above. If it stayed the same and if the price update message originated from u, then v sends a new vote to u with a –now larger– limit value than previously and thus reenters $S(u)$. If $u = v$, then v records the new limit value for NL_v.

If a node v receives a DOMINATOR($N(u)$) from node u and if v was previously uncovered, it sends a message PRICE(span, state), where state = covered, to all neighbors in $N(v) \setminus N(u)$ that are not marked as dominators in $D(v)$, independently of their limit value.

Communication Complexity. Assuming fixed-length fields for node identifiers, weights, and number of neighbors, the communication complexity of the distributed algorithm is $O(|V|\Delta^2)$. The price of a node can change at most Δ^+ times. Each price change can trigger at most Δ price updates, each of which may require sending one vote. So the total number of vote and price update messages is $O(|V|\Delta^2)$, each of constant size. At most $|V|$ nodes may become dominators. Each dominator sends at most Δ dominator messages to inform its neighbors, and each dominator message contains information on at most Δ neighbors.

4.3 Practical Considerations

One advantage of wireless networks is their broadcast nature. As a further improvement, we let nodes overhear price updates sent between neighbors. Furthermore, the length of network-interface queues is typically limited. To prevent packet drops due to buffer overflows, instead of sending price updates immediately the neighbors are marked for notification, and marked nodes are notified periodically. We study the effect of the price update interval by experiments.

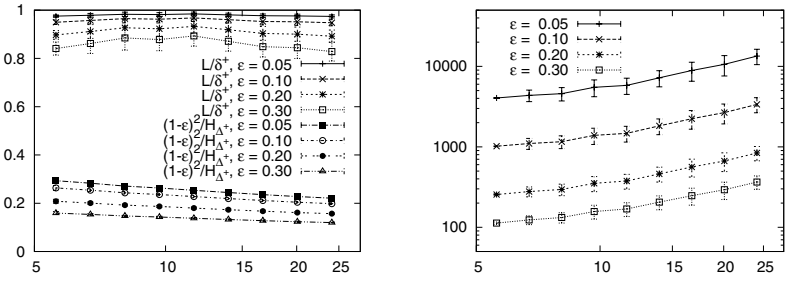
5 Experimental Evaluation

We evaluate our algorithm in two parts. After generating a number of test instances, we first use Matlab to compare the actual performance of the algorithm with the theoretical guarantee and to compute the number of GK iterations required. After that, we simulate our distributed algorithm with `ns2` to verify the correctness of our algorithm and to determine the number of messages required.

5.1 Performance Evaluation of the Centralized GK Scheme

We generated a set of 20 disk graphs by scattering 150 nodes onto square areas of various sizes uniformly at random. Connectivity was determined by the `ns2` default transmission range. We varied the average node density by changing the size of the area and discarded disconnected graphs.

Figure 1(a) shows the performance for different ϵ compared to the approximation bound using the upper bound on network lifetime δ^+. One observes that



(a) Lifetime/δ^+ for different ϵ (b) Number of iterations for different ϵ

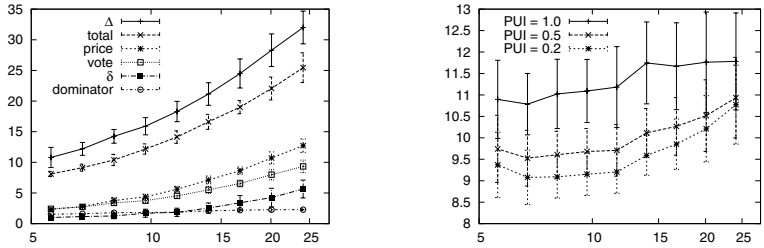
Fig. 1. The x-axis shows the expected node degree on a logarithmic scale disregarding terrain boundary effects; a) also shows the bound for the approximation factor

the total lifetime is close to its upper bound and the algorithm performs better than what one might expect from the approximation guarantee. Figure 1(b) shows the required number of iterations for the same set of instances. In all plots errorbars show the standard deviation over a set of 20 instances.

5.2 Network Simulations

We implemented the combined distributed algorithm of Sections 2.3 and 4 as a routing agent in `ns2` and consider the number of control messages and simulated termination time for the same network instances used in the Matlab experiments. Additionally, we evaluate the effect on the simulated running time achieved by choosing different values for the length of the price update interval (PUI).

Figure 2(a) shows the number of messages per node per iteration required for a fixed value of PUI and $\epsilon = 0.2$. The total number of messages per node



(a) Messages per node per iteration according to type for PUI=1.0

(b) Seconds per iteration for different PUI length

Fig. 2. Number of messages and second per iteration versus expected node degree; data on x-axis is shown on a logarithmic scale disregarding terrain boundary effects. Message Counts also include retransmissions due to collisions.

generally lies between the maximum and the average degree in the graph. Figure 2(a) also shows the number of messages split according to type. The price update messages have the largest contribution to the number of messages, followed by votes and dominator messages. Figure 2(b) shows the average duration of a single iteration of the algorithm for different lengths of PUI. One observes that the actual value of PUI generally has a minor effect on the average duration of a single iteration of the GK scheme, particularly when the network is dense.

6 Conclusions

We present a distributed approximation algorithm for the sleep-scheduling problem based on the Garg-Könemann scheme and linear programming duality. A key component of the algorithm is our efficient distributed implementation of Chvátal's greedy set-covering algorithm. The set-covering problem is a central combinatorial problem and we believe our implementation to be useful also in other problem settings; moreover, the LP duality technique used to obtain locality may be useful also for other problems. Our algorithm is based on a mathematical framework that provides a guarantee on the solution quality. Moreover, our simulation results suggest that the algorithm also performs well in practice.

Acknowledgements. The authors thank Pekka Orponen for fruitful discussions on the topic. A. Schumacher has been supported by the Helsinki Graduate School of Computer Science and Engineering and by the Nokia Foundation.

References

1. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. Wirel. Netw. 8(5), 481–494 (2002)
2. Basagni, S., Mastrogiovanni, M., Petrioli, C.: A performance comparison of protocols for clustering and backbone formation in large scale ad hoc networks. In: Proc. IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems, pp. 70–79 (2004)
3. Wang, L., Xiao, Y.: A survey of energy-efficient scheduling mechanisms in sensor networks. Mobile Networks and Applications 11(5), 723–740 (2006)
4. Dietrich, I., Dressler, F.: On the lifetime of wireless sensor networks. ACM Trans. Sen. Netw. 5(1), 1–39 (2009)
5. Floréen, P., Kaski, P., Musto, T., Suomela, J.: Local approximation algorithms for scheduling problems in sensor networks. In: Kutyłowski, M., Cichoń, J., Kubiak, P. (eds.) ALGOSENSORS 2007. LNCS, vol. 4837, pp. 99–113. Springer, Heidelberg (2008)
6. Suomela, J.: Locality helps sleep scheduling. In: Working Notes of the Workshop on World-Sensor-Web: Mobile Device-Centric Sensory Networks and Applications (2006), `http://www.sensorplanet.org/wsw2006/8_Suomela_WSW2006_final.pdf`
7. Cărbunar, B., Grama, A., Vitek, J., Cărbunar, O.: Redundancy and coverage detection in sensor networks. ACM Trans. Sen. Netw. 2(1), 94–128 (2006)
8. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM J. Comput. 37(2), 630–652 (2007)

9. McCanne, S., Floyd, S., Fall, K., Varadhan, K.: The network simulator `ns2`, The VINT project (1995), `http://www.isi.edu/nsnam/ns/`
10. Feige, U., Halldórsson, M.M., Kortsarz, G.: Approximating the domatic number. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC 2000), pp. 134–143. ACM, New York (2000)
11. Moscibroda, T., Wattenhofer, R.: Maximizing the lifetime of dominating sets. In: Proceedings of The 19th IEEE International Parallel and Distributed Processing Symposium (2005)
12. Berman, P., Calinescu, G., Shah, C., Zelikovsky, A.: Power efficient monitoring management in sensor networks. In: Wireless Communications and Networking Conference, WCNC 2004, pp. 2329–2334. IEEE, Los Alamitos (2004)
13. Chvátal, V.: A Greedy Heuristic for the Set-Covering Problem. Mathematics of Operations Research 4(3), 233–235 (1979)
14. Tsaggouris, G., Zaroliagis, C.: QoS-aware multicommodity flows and transportation planning. In: Jacob, R., Müller-Hannemann, M. (eds.) ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
15. Santoro, N.: Design and Analysis of Distributed Algorithms. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, Hoboken (2006)
16. Rajagopalan, S., Vazirani, V.V.: Primal-dual RNC approximation algorithms for set cover and covering integer programs. SIAM J. Comput. 28(2), 525–540 (1999)
17. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Connected Dominating Set in Sensor Networks and MANETs, pp. 329–369. Kluwer Academic Publishers, Dordrecht (2005)
18. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple heuristics for unit disk graphs. Networks 25, 59–68 (1995)
19. Nieberg, T., Hurink, J., Kern, W.: Approximation schemes for wireless networks. ACM Trans. Algorithms 4(4), 1–17 (2008)
20. Feige, U.: A threshold of ln n for approximating set cover. J. ACM 45(4), 634–652 (1998)
21. Jia, L., Rajaraman, R., Suel, T.: An efficient distributed algorithm for constructing small dominating sets. Distrib. Comput. 15(4), 193–205 (2002)
22. Awerbuch, B.: Complexity of network synchronization. J. ACM 32(4), 804–823 (1985)
23. Wang, Y., Wang, W., Li, X.Y.: Distributed low-cost backbone formation for wireless ad hoc networks. In: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2005), pp. 2–13. ACM, New York (2005)
24. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM 50(6), 795–824 (2003)