Efficient algorithms to solve a class of resource allocation problems in large wireless networks

Luo, Jun.; Girard, Andre.; Rosenberg, Catherine.

2009

Luo, J., Girard, A., & Rosenberg, C. (2009). Efficient algorithms to solve a class of resource allocation problems in large wireless networks. 7th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, 2009: Seoul Korea.

https://hdl.handle.net/10356/83921

Efficient Algorithms to Solve a Class of Resource Allocation Problems in Large Wireless Networks

Jun Luo
School of Computer Engineering
Nanyang Technological University
Singapore 639798
Email: junluo@ntu.edu.sg

André Girard
Groupe d'Études et de Recherche
en Analyse des Décisions
Montréal, Canada H3T 2A7
Email: andre.girard@gerad.ca

Catherine Rosenberg
Department of Electrical and
Computer Engineering
University of Waterloo
Waterloo, Canada N2L 3G1
Email: cath@ecemail.uwaterloo.ca

***Abstract*—We focus on efficient algorithms for resource allocation problems in large wireless networks. We first investigate the link scheduling problem and identify the properties that make it possible to compute solutions efficiently. We then show that the node on-off scheduling problem shares these features and is amenable to the same type of solution method. Numerical results confirm the efficiency of our technique for large scale problems. We also extend the technique to the case where the objective function is nonlinear showing that our technique blends smoothly with a sequential linear programming approach. Numerical results for a cross-layer design with a nonlinear fairness utility show that it is possible to compute optimal solutions for large wireless networks in reasonable CPU time.**

I. Introduction

Resources such as bandwidth or power are generally scarce in multihop wireless networks, so that the problem of their optimal allocation has received a lot of attention over the last few years. This problem includes, among others, link scheduling, both for time sharing and spatial reuse, power control, channel allocation, gateway or relay placement, and node on-off schedule for energy saving. Many approaches have been used to study these problems and their variants. In most cases, they yield hard combinatorial optimization problems. Some analytic methods can be used to provide bounds on various measures of network performance but this generally requires some strong simplifying assumptions (e.g., [1]). Instead, exact numerical solutions are often needed both to understand the trade-offs that are available and to actually design the networks. Our literature survey seems to indicate that current techniques are limited in the size of the problems that can be studied, e.g., to about 20 nodes for a cross-layer design with a nonlinear objective [2] , which is not nearly large enough to design realistic networks.

Recently, we have developed an efficient solution technique for jointly optimizing scheduling, routing, power control and rate adaptation in a fixed multihop wireless network, with a linear objective function [3]. This we call the SRPR problem with linear objective. It can deal with relatively large network sizes, can handle arbitrary network parameters such as topology, interference model, routing, etc, and can deliver both optimal and near-optimal solutions.

The goal of this paper is to show how this solution technique can address more problems than simply those of [3]. First, we review in Section II the features of the *link scheduling problem* (LSP), a simplified version of SRPR, on which the solution method is based. Next, we show in Section III how another type of problem, namely node on-off scheduling for energy saving, has features similar to those of LSP and can be solved by similar techniques. Numerical results for this problem demonstrate the efficiency of our technique for large scale problems. In Section IV, we extend the solution method to problems with a nonlinear objective and present in Section IV-E numerical results for the SPPR with a proportional fairness objective, showing that the solution technique can handle reasonably large networks in moderate computation time. We conclude our paper in Section V.

II. Review of The Solution Technique

We have studied how to maximize the minimum flow throughput of a fixed multihop wireless network operating under a joint scheduling, routing, power control, and rate adaptation. This we call the SRPR problem with linear objective [3]. The interested readers can find the complete model in the reference and for now, we concentrate on the link scheduling sub-problem which is the difficult part and is the basis on which many extensions can be built. We use this problem as an example to illustrate the type of problem structure that is amenable to our numerical technique and to explain how an efficient solution can be constructed.

A. Compact Formulation of Hard Constraints

We model the multihop network at the PHY and MAC layers as a set $\mathcal{N}$ of *nodes* and a set $\mathcal{L}$ of *directed link*s, with $|\mathcal{N}| = N$ and $|\mathcal{L}| = L$. Each node $i \in \mathcal{N}$ has a location (x_i, y_i). We denote by $\mathcal{L}_i$ the set of links incident (inbound or outbound) to a node i. A link $l \in \mathcal{L}$ is identified by its source-destination pair $o(l)$ and $d(l)$.

At any given instant, the scheduler designates a set of links to transmit simultaneously to allow spatial reuse. These links must be chosen in such a way that 1) they meet certain conditions related to the radio interface characteristics and 2) they do not interfere with each other beyond a certain limit. We call such a set of links an *Independent Set* (ISet).

The typical radio interface characteristics can be expressed as follows. Two links $l = (i, j)$ and $l' = (i', j')$ do interfere with each other if $(i = i') \vee (i = j') \vee (j = i') \vee (j = j')$. Therefore, a set $s \subseteq \mathcal{L}$ is an ISet only if:

$$(i \neq i') \wedge (i \neq j') \wedge (j \neq i') \wedge (j \neq j') \quad \forall l, l' \in s. \tag{1}$$

There are many types of interference constraints. One example is the SINR-based constraint, which is based on the assumption that the interference at the receiving end of a given link is the cumulative interference from all the links that are active at the same time. In this case, a set $s \subseteq \mathcal{L}$ is an ISet iff it meets condition (1) and the condition:

$$\gamma_l = \frac{G_l P_l}{N_0 + \sum_{l' \in s: l' \neq l} G_{l'l} P_{l'}} \geq \beta_l \quad \forall\, l \in s. \tag{2}$$

Here, γ_l is the *signal to interference plus noise ratio* (SINR) of link l, β_l is the SINR threshold for a link rate c_l, P_l is the transmit power of the source of link l, N_0 is the average thermal noise power, and G_l (resp. $G_{l'l}$) is the channel gain of link l (resp. from $o(l')$ to $d(l)$).[1]

Optimization problems for scheduling such networks can take a number of forms with different objectives and constraints but in all cases, there must be a set of binary decision variables q_l to indicate whether link l is active at a given instant, making them Integer Programming (IP) problems. These variables must meet a set of hard linear constraints:

$$\sum_{l \in \mathcal{L}_i} q_l \;\leq\; 1 \qquad \forall\, i \in \mathcal{N} \tag{3}$$

$$M\,(1 - q_l) + P_l G_l \;\geq\; \beta_l \left[N_0 + \sum_{l' \in \mathcal{L}: l' \neq l} P_{l'} G_{l'l} q_l \right] \quad \forall\, l \in \mathcal{L} \tag{4}$$

where M is some large constant to express the fact that the constraint needs to be satisfied only for the links that are active. Eqs. (3) and (4) correspond to (1) and (2), respectively. The structure of these constraints makes it difficult to solve these problems directly for large networks using generic solvers. This paper deals with the application and generalization of a technique, the *extensive formulation*, to solve them efficiently for large instances.

B. Extensive Formulation

A standard technique for solving some classes of IPs or MIPs is based on the so-called *extensive formulation* [4]. This is useful when there are two sets of constraints: *easy* and *hard*, also called *complicating*, constraints. The easy constraints are such that optimization can be made efficiently, e.g., network flows, spanning tree, etc, while the hard constraints cannot be handled so easily and lead to hard problems, e.g., coupling constraints in a network flow problem or, for any problems involving link scheduling, constraints (3–4).

The extensive formulation of the problem is built by taking all the variables involved in the hard constraints, for instance the $\mathbf{q}$ variables, and listing all possible values of these variables compatible with the hard constraints. Here we call a set of these variables a *configuration*. An ISet is such a configuration. The hard constraints are removed from the problem and the remaining optimization is then to choose the appropriate configurations and optimize whatever other variables are left subject to the easy constraints.

In some cases, the extensive formulation can be solved more easily than the original problem. We have found this is the case for the link scheduling problem, as explained below.

C. Link Scheduling Problem (LSP)

In addition to the PHY and MAC layer models described earlier, we have $\mathcal{F}$ as the set of *flow*s and $|\mathcal{F}| = F$. We also have, for each link l, a traffic load x_l^f,[2] which is the average traffic of flow f on link l. The link scheduling problem (LSP) that we consider is to find a schedule of minimum length that is able to carry these loads [5], [6].

To construct the extensive formulation, we need to enumerate all the ISets compatible with the constraints (3–4). We denote by $\mathcal{I}$ the set of all ISets and the structure of $\mathcal{I}$ is represented by the link-set incidence matrix Q

$$q_{l,s} = \begin{cases} 1 & \text{if } l \in s \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with $l \in \mathcal{L}$ and $s \in \mathcal{I}$. Note that each column $\mathbf{q}_s$ of Q is a binary vector that represents an ISet s. Based on this, the vector of decision variables $\boldsymbol{\alpha} = [\alpha_s]_{s \in \mathcal{I}}$ is defined where α_s is the fraction of the time that ISet s is going to be used. The LSP problem has the following form

$$\max_{\boldsymbol{\alpha} \succeq \mathbf{0}} \quad -\sum_{s \in \mathcal{I}} \alpha_s \tag{6}$$

$$(\nu_l) \qquad c_l \sum_{s \in \mathcal{I}} q_{l,s} \alpha_s \;\geq\; \sum_{f \in \mathcal{F}} x_l^f \qquad \forall\, l \in \mathcal{L} \tag{7}$$

where we have put the Lagrangian multipliers corresponding to each constraint in parenthesis. In this formulation, we have $\nu_l \geq 0$. Eq. (7) expresses the fact that the average traffic on a link cannot be larger than the link rate c_l times the amount of time the link is used. Note that this problem is a crucial component of many network design problems, including the cross-layer design problem we will present in Section IV-D.

The **first important feature** of this formulation is that all the integer variables have been removed so that the extensive formulation is a standard linear program. This immediately suggests trying to solve it with a standard LP solver. This in turn requires the enumeration of all ISets. We now discuss some of the features of the LSP that make this enumeration possible for relatively large cases.

[1]The benefit of associating certain properties (such as the rate and transmit power) with a link is that a *physical* link can be decomposed into several *logical* entities, with each represented by a given set of properties. Consequently, the link scheduling problem is extended to a more general resource allocation problem. For example, let both c_l and P_l take values from a finite set $\mathcal{C}$ and $\mathcal{P}$, respectively. The activation of the (logical) link l implies also the allocation of rate c_l and power P_l to this link. Moreover, associating properties such as the channel gain allows us to extend link scheduling also to channel allocation.

[2]It can be readily computed given the knowledge of the routing paths of individual flows.

D. Complete Enumeration

The extensive formulation has a very large number of α_s variables, one per ISet, which generally grows exponentially with the problem size. For sufficiently small networks, they can be explicitly enumerated and the resulting LP can be solved using a standard simplex algorithm since all the columns of Q in constraint (7) are available. Such a direct approach faces two difficulties. First, enumerating all the columns can be a tremendous task. Secondly, solving an LP with many variables can also be hard even though good commercial LP/NLP solvers such as CPLEX [7] and MINOS [8] can deal with problems with hundreds of thousand variables. This approach does have an advantage, though: once Q is known, other components of the problem, such as the cost coefficients of a linear objective or the flows, can be changed freely without much complication.

For this approach, an efficient enumeration to generate all columns of Q is necessary. It is based on the observation that removing a link from an ISet gives another ISet, since having one less link cannot prevent a set of links that could previously transmit from doing so. This we call the *divisible* property:

Definition 1: We say that a binary vector $\mathbf{q}'$ is a subvector of another binary vector $\mathbf{q}$ if $\mathbf{q}'$ is obtained from $\mathbf{q}$ by setting one or more of $\mathbf{q}$'s components to zero. We denote this relation $\mathbf{q}' \sqsubseteq \mathbf{q}$. We then say that Q is *divisible* if for any $\mathbf{q} \in Q$, every $\mathbf{q}'$ such that $\mathbf{q}' \sqsubseteq \mathbf{q}$ is also in Q.

If Q is divisible, the enumeration can be reasonably fast using a depth-first search algorithm. Define the *size* of vector $\mathbf{q}$ as the number of non-zero elements of $\mathbf{q}$. The root of the search tree starts at vectors of size 0. Each node of the tree corresponds to a binary vector $\mathbf{q}$. Let S_n be the set of nodes of size n. For each $\mathbf{q}' \in S_n$, we construct a child node $\mathbf{q}$ of size $n+1$ if 1) $\mathbf{q}' \sqsubseteq \mathbf{q}$ and 2) $\mathbf{q}$ satisfies the conditions that define Q. In other words, the fact that the matrix is divisible allows us to construct ISets of size n based only on the ISets that were created at level $n-1$; children of a node $\mathbf{q}' \notin S_n$ are all pruned from the search tree.

In addition, the solution of the resulting LP/NLP can be made faster if we can reduce the size of Q. This can be done due to the following proposition that summarizes the observations made in [9], [10]:

Proposition 1: The range space of Q is the span of all the maximal column vectors, where $\mathbf{q} \in Q$ is defined as being maximal if $\mathbf{q} \not\sqsubseteq \mathbf{q}', \forall\, \mathbf{q}' \in Q$.

The depth-first search can then be pruned to leave only maximal vectors which can be used as input to the LP solver.

This divisibility property of the constraint matrix is the **second important feature** of the LSP that makes a direct solution of the LP feasible. Nevertheless, we have found that such a direct approach is still limited in the size of the problems that it can solve, and we have developed another solution technique based on column generation which can solve much larger problems by taking advantage of this divisibility property.

E. Column Generation

For large problems, the list of ISets cannot be built explicitly but may be generated as needed by the optimization. The method is called *column generation* and the generation of a configuration that can improve the current solution is called the *pricing* [4]. Column generation is particularly useful when the pricing operation is a simple problem such as a shortest path or minimum-cost flow. Nevertheless, we want to show that column generation can still be very efficient even though the pricing problem remains hard as it is the case for LSP.

Column generation is basically the revised simplex algorithm which performs the standard simplex operations using only the current basis matrix. The only difference is that a sub-problem, called the *pricing* problem, is solved at each iteration to find the off-basis column with a sufficiently large *reduced price*[3]. If the solution of the pricing sub-problem has a positive value, the corresponding column is pivoted into the basis; otherwise, the current solution is optimal. For the LSP, the computation time is dominated by the pricing operation.

1) Optimal Pricing: If we use the standard revised simplex algorithm, we must solve the pricing problem to optimality at each iteration. This guarantees that the final solution is optimal whenever there is no solution to the pricing problem with a positive reduced price. The pricing of the α_s variables requires evaluating the corresponding reduced price r_s.

$$r_s = -1 + \sum_{l \in \mathcal{L}} c_l \nu_l q_{l,s} \tag{8}$$

where the constant -1 comes from the cost coefficient in the objective (6) and ν_l is the Lagrangian multiplier of constraint (7). At each iteration, we have to find a maximum-price ISet, i.e., a binary vector $\mathbf{q}_s$ that maximizes r_s such that constraints (3–4) are met. This is an NP-hard problem.

2) Positive Pricing: Solving the pricing sub-problem to optimality at each iteration is a time-consuming operation that can be avoided most of the time. Instead of finding the column with the largest reduced price, we can use any heuristic procedure to choose **any** column, e.g., any ISet, with a positive reduced price. Pivoting this column into the basis may still bring **some** increase of the objective function. A greedy heuristic is an obvious choice for positive pricing, but at some point, it might fail to find a column with a positive reduced price. In this case, we have two options.

The first one is to stop at that point. This we call *approximate positive pricing*. The problem with this approach is that we cannot guarantee that the final solution is optimal since the heuristic may have missed some column with a positive reduced price.

The second option, which we call *exact* positive pricing, is to revert to a more accurate, and hence more complicated pricing algorithm that guarantees to find a column with a positive reduced price if it exists. Consequently, if this pricing algorithm does not find any column with a positive reduced

[3]A term involved in the simplex algorithm [11]. It represents the marginal improvement that a column can bring to the objective.

price, we know that we have an optimal solution. We describe in Section II-E3 the exact positive pricing that we have developed.

Remarks: In theory, using only optimal pricing usually needs fewer iterations to converge than the other methods since it always chooses the column that has the largest potential to increase the objective at the current solution. This is one reason why it has been widely used to solve problems related to LSP [12], [13], [2], [14]. However, the total CPU time taken by the optimal pricing is actually much longer when the pricing sub-problem is NP-complete/hard. It has been shown recently by [15] that approximate positive pricing based on a greedy heuristic converges very fast to a suboptimal solution. Unfortunately, it is not clear how close a suboptimal solution is to the optimum [14] since the greedy algorithm might miss some columns with a positive price.

3) Our Positive Pricing Algorithm: Our implementation of the positive pricing involves two algorithms. The first algorithm is a *greedy pricing*. The algorithm simply orders the links in decreasing weights. The link with the largest weight is first chosen, then the link with the largest possible weight that is still independent of the chosen one is selected, and so on until an ISet is constructed. There are many ways to define those link weights. The most straightforward one is to use the $c_l\nu_l$ defined in Eq. (8) as the weights, but we use a more sophisticated definition taking into account also some measure of the interference; it usually works better than the straightforward one. We will explain this in more detail when we come to a particular example in Section IV-D2.

When this fails to find a suitable column, the procedure reverts to our second algorithm, the efficient enumeration of ISets described in Section II-D, until it finds an ISet with a positive reduced price or fails to do so after enumerating all the possible ISets. If we want to guarantee the optimality of the final solution of the column generation algorithm, a full enumeration is needed to solve the pricing sub-problem exactly **but** only in the last iteration and on a limited subset of links (as discussed below).

Although the pricing by enumeration used in the second algorithm is still a complex task, it can be made quite efficient due to the following reasons:

- The enumeration can be fast if Q has the divisible property.
- Apart from the latest iteration, the enumeration only runs until a column with positive price is identified.
- More importantly, the enumeration is performed only on the set of links whose price is positive since there is no advantage in adding a link with zero price when maximizing the total price of an ISet. Hence the dual degeneracy may reduce the size of the problem significantly.

Our numerical results show that the suboptimal solution obtained with only the greedy pricing can be almost optimal in most cases. When the greedy pricing fails to produce a column with positive reduced price, we invoke the enumeration pricing if we are after the exact optimal solution. This is the **third important feature** of the LSP: Even though the pricing problem is hard, it is possible to use a greedy algorithm either to produce a good approximate solution or to combine it with an efficient enumeration on a reduced set of links to find a provably optimal solution.

F. Summary

The solution techniques work well for the LSP because the problem has the following features.

1) It is possible to give an extensive formulation.
2) The resulting problem is a standard LP.
3) A full enumeration of the columns for reasonably large cases is doable thanks to the divisibe property of Q.
4) Column generation works well even though the pricing sub-problem is hard for large problems.
5) Using a greedy algorithm as an approximate pricing leads to a final solution very close to the optimum.

What we want to show in the next section is that there exist other network design problems that share these features, so that the techniques proposed for LSP can in principle be used to solve them efficiently as well. To this end, we will formulate one such problem, show that it has the right structure, exhibit the pricing, determine an efficient set of weights for the positive pricing algorithm and develop the numerical tools.

III. Node On-Off Scheduling

Densely deployed wireless sensor networks often have more nodes than needed to cover their deployment area. Therefore it is possible to activate only a subset of nodes for a certain time period in order to increase the total *coverage time* [16], [17], [18] by saving the energy of the remaining nodes. This is done by partitioning the node set $\mathcal{N}$ into several *cover sets*[4] $\mathcal{M}_k \in \mathcal{N}$, and turn on each one only for a time t_k. We can optimize the coverage time of the network by maximizing the sum of all t_ks. Our technique provides a way of solving these problems exactly as opposed to the approximate solutions proposed in [17], [18] for example.

A. Network Model

The energy consumption model is the following. We first assume that all nodes spend an identical power P^t when they are on, but the formulation that we will present can handle a power varying with both node and cover set. We also assume that each node i is given a total amount of energy E_i initially, and we denote by $\mathbf{E} = [E_i]^T$ the vector of energy allocation of the nodes. We continue using some of the modeling parameters defined in Sec. II-A and II-C.

Obviously, each cover set has to meet certain *coverage conditions*. An example of such conditions is when there is a number of locations $\{\ell_i\}$ that have to be covered by the sensors at all times. In this case, a cover set is a set of sensors such that for each location ℓ_i, there is at least one sensor j such that the distance $d(\ell_i, j) \leq R_j^s$ where R_j^s is the so-called *sensing range* of j, i.e., the maximum distance at which a sensor can detect whatever it is supposed to detect. The extensive form can be

[4]The term *cover set* should not be confused with the concept of a cover set for the classical set covering problem

obtained by enumerating all subsets that meet this condition. This produces a set of N-dimensional $\{0,1\}$ *characteristic vectors* $\mathbf{q}_k$ each representing a cover set $\mathcal{M}_k$, i.e., $q_{i,k}=1$ if $i \in \mathcal{M}_k$ and $q_{i,k}=0$ otherwise. The collection of vectors $\mathbf{q}_s$ forms the Q matrix of the problem as described for LSP.

We also define the decision variable t_k as the time that cover set k is used and $\mathbf{t}=[t_k]$ as the schedule vector. We first consider a simpler case where connectivity is not a issue. Then we extend the problem formulation to take connectivity into account.

B. On-Off Schedule for Coverage Only

This problem assumes that nodes spend most of their energy to perform their tasks, e.g., monitoring, so that the energy spent for communications can be neglected. The problem can be written as

$$\max_{\mathbf{t}\succeq\mathbf{0}} \quad \sum_{\mathcal{M}_k\in\mathcal{N}} t_k \tag{9}$$

$$(\boldsymbol{\omega}) \qquad P^t \sum_{\mathcal{M}_k\in\mathcal{N}} \mathbf{q}_k t_k \ \preceq\ \mathbf{E} \tag{10}$$

where $\boldsymbol{\omega}$ is the Lagrangian multipliers (vector). We have $\boldsymbol{\omega}\preceq\mathbf{0}$. Note that this is a standard LP just as for LSP.

C. On-Off Schedule for Coverage and Connectivity

In addition to simply covering an area, a wireless sensor network has to transmit the collected information to certain destinations, e.g., the gateways. Therefore, each cover set $\mathcal{M}_k$ must meet certain *connectivity conditions*. A simple example is to require a cover set to be connected given a *transmission range* R_i^t for a node i. More complicated examples may involve a flow rate vector $\boldsymbol{\lambda}=[\lambda^f]_{f\in\mathcal{F}}$ for a certain flow set $\mathcal{F}$ which has to be feasible for a given cover set $\mathcal{M}_k$.

No matter what connectivity conditions are involved, a node will consume a communication power P^c to maintain connectivity. The following is a possible formulation of P^c for node i in cover set $\mathcal{M}_k$.

$$P_{i,k}^c=\sum_{j\neq i}\left[\frac{P_{i,j}x_{i,j}^{k,f}}{c_{i,j}}+\frac{P_{\rm rx}x_{j,i}^{k,f}}{c_{j,i}}\right]$$

where $x_l^{k,f}$ is the load produced by flow f on link l during t_k, $c_{i,j}$ is the rate of link (i,j), $P_{i,j}$ is the transmit power of link (i,j) (which enables node i to have, for example, a transmission range of R_i^t), and $P_{\rm rx}$ is a fixed receiving power identical for all nodes. It is straightforward to see that, as the ratio between a load and a link rate represents the time fraction during which a link is active, the two terms in the expression indicate the total transmit and receiving power consumptions, respectively, during t_k.

The constraint (10) now becomes

$$(\omega_i) \qquad \sum_{\mathcal{M}_k\in\mathcal{N}} P_{i,k}q_{i,k}t_k \quad\le\quad E_i \qquad \forall\ i\in\mathcal{N} \tag{11}$$

where $P_{i,k}=P_{i,k}^t+P_{i,k}^c$. Note that, $P_{i,k}^t$ extends the constant P^t by considering it as a function of both node and cover set. We have $\omega_i\le 0, \forall\ i\in\mathcal{N}$.

D. Pricing and Weight Choice

First, the reduced price of a column corresponding to a t_k variable is given by

$$r_k=1+\sum_{i\in\mathcal{N}}\omega_i P_{i,k}q_{i,k}$$

where the constant 1 comes from the cost coefficient in the objective (9) and ω_i is the Lagrangian multipliers of constraints (10) or (11). An optimal pricing requires the solution of a minimum cover with node weight $-\omega_i P_{i,k}$ for a cover set $\mathcal{M}_k$, whereas a positive pricing only needs to identify some cover set $\mathcal{M}_k$ whose r_k is positive. The column generation terminates if $r_k\le 0$ for all off-basis columns.

Second, the pricing algorithms, both greedy and enumeration-based, can take advantage of specific geometrical property of individual sensor coverage regions. For example, if they are convex, e.g., circles, an efficient data structure based on a planar graph can be used [17].

Third, although the matrix Q with columns $\mathbf{q}_k$ does not have the divisible property described in Section II-D, its bit-wise complement matrix $\overline{Q}$ does and we can implicitly enumerate $\mathcal{M}_k$ by actually enumerating its complement set.

Finally, it is very unlikely that all the N constraints (11) are met with equality unless nodes are regularly distributed and they share the same power consumption profile. Therefore, the dual degeneracy can also be used. In this case, a node i with $\omega_i=0$ can always be chosen to construct a cover set $\mathcal{M}_k$ since it does not decrease the reduced price r_k. Such a node can always be included in Q and thus can always be removed from $\overline{Q}$. Because we are enumerating $\overline{Q}$ instead of Q, this means that the enumeration will be done on a smaller set and will thus be made faster.

E. Numerical Results

In this section, we use several examples to demonstrate the efficiency of our algorithm. The numerical results presented here and in Section IV-E are produced by tools we developed. The tools are programmed in C++ and call the APIs of CPLEX [7] to solve an LP or IP in the case that an optimal pricing is required.

We have a 160×160 m^2 area to be covered by a wireless sensor network. In this area, we have 441 targets organized in a grid with a separation of 8 m between targets. In each scenario, we randomly deploy N sensors with $N=100,200,400$ or 800. The sensing range R_i^s for each sensor is a Gaussian random variable with identical mean and the same standard deviation of 5 m. For each sensor i, $R_i^t=1.5R_i^s$. A cover set must have the following properties:

1) All the targets can be sensed and
2) It forms a connected sub-network.

For the energy model, we assume that any node in an active cover set consumes a power proportional to the square of the sensing range, i.e., $P_{i,k}\propto(R_i^s)^2$. Without loss of generality, we take $\mathbf{E}=\mathbf{1}$.

In order to show the advantage of our positive pricing technique, we first compare the computation times of two algorithms involving an optimal pricing and an **exact** positive pricing respectively. We start by computing the maximum coverage time of a 200-node network using on-off scheduling. The computation is done for different mean sensing ranges from 20 m to 78 m with a step size of 2 m. The results in terms of computation times are shown in Fig. 1. It is obvious

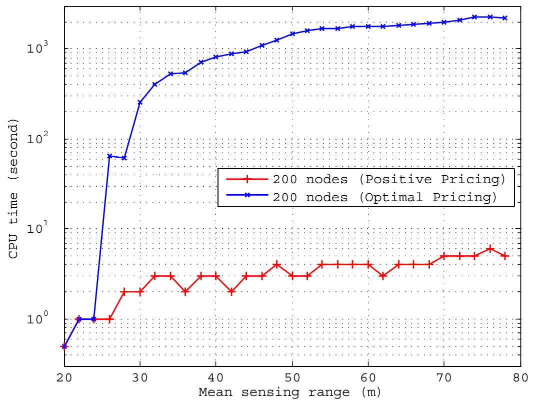


Fig. 1. CPU time comparison between optimal pricing and positive pricing.

that the optimal pricing is far slower than the positive pricing. This is not a surprise because the optimal pricing needs to solve an NP-hard *connected sensor coverage problem* [19] in each column generation iteration.

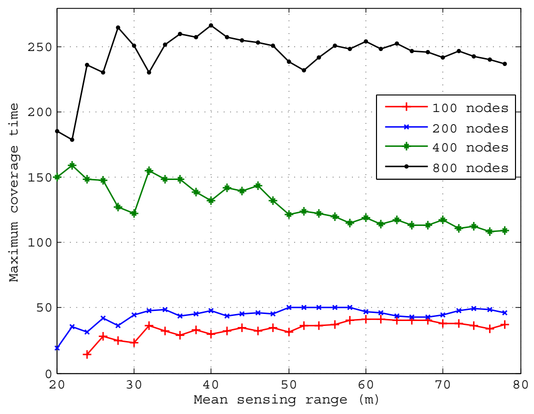
Using our tool that is based on the positive pricing described in Section III-D, we compute the maximum coverage times for networks of 100, 200, 400, and 800 nodes. The results for both coverage times and computation times are reported in Fig. 2. It is interesting to see that the coverage time is not a monotonic function of the mean sensing range under our model. On one hand, increasing the sensing range may result in more cover sets and less nodes in each cover set; this can potentially improve the coverage time. On the other hand, the power consumption is increased with the sensing range, this contributes negatively to the coverage time. Therefore, one has to be careful when tuning the power to vary the sensing (or transmission) range. Our tool provides a convenient way to study the tradeoffs involved in the design issues for very large networks. The computation time, shown in Fig. 2(b), grows significantly with the network size. This is an inevitable consequence of the NP-hard nature of the problem, albeit the efficiency of our algorithm.

IV. Extension to Nonlinear Objectives

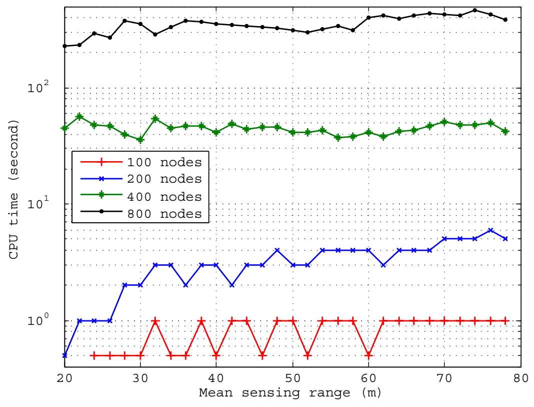
Network optimization problems may sometimes involve a nonlinear (but concave in the case of maximization) objective $U(\mathbf{y})$. In this section, we first revisit the column generation method of Section II-E to see how it can be extended to deal with a nonlinear objective of this kind, then we describe a specific problem that is amenable to this solution technique.

A. Column Generation for Nonlinear Objectives

One possible approach to nonlinear objectives is the nonlinear column generation presented in [2]. It combines Lagrangian decomposition with column generation and requires solving two nonlinear optimizations in each iteration to obtain upper and lower bounds. The pricing sub-problem has to be solved to optimality and, as we explained in Section II-E1, the exact pricing used in [2] does not scale well to large problems. While it is true that the positive pricing algorithm scales well, combining it with the Lagrangian decomposition is not an efficient approach, as it does not produce an upper bound on the value of the objective.



(a)



(b)

Fig. 2. Coverage Time (a) and computation time (b) as functions of sensing range and network size.

Our approach is based on the *sequential linear programming* (SLP), also known as the *Frank-Wolfe* method [20]. This is particularly useful for problems with a nonlinear objective $U(\mathbf{y})$ and linear constraints $A\mathbf{y} = \mathbf{b}$. At each iteration, it only needs to solve a linear program to compute a feasible direction and then to calculate the maximum in this direction using a line search algorithm. The main steps of the Frank-Wolfe method are as follows:

1) Find an initial feasible solution $\mathbf{y}_0$.
2) At iteration i, let $\mathbf{y}_i$ be the current solution. A linearized version of the problem with the objective function $\langle \nabla U(\mathbf{y}_i), (\mathbf{y} - \mathbf{y}_i) \rangle$ and the original constraints is solved. This produces a vector $\mathbf{y}_i^*$ that is a vertex of the domain and a direction $\mathbf{d}_i = \mathbf{y}_i^* - \mathbf{y}_i$.
3) Find, for $\tau_i \geq 0$, the step size in the direction $\mathbf{d}_i$,

by solving the one-dimensional nonlinear optimization $\max_{\tau_i} U(\mathbf{y}_i + \tau_i \mathbf{d}_i)$.

We now explain how the Frank-Wolfe method can be used together with our column generation technique efficiently.

B. Line Search

Given a direction from the current solution, we must find the maximum of the function in that direction. This is a one-dimensional nonlinear optimization problem. In our implementation, the line search is done with the *golden section method* [20] but any one-dimensional minimization technique could be used. The important point is that it is stopped before optimality when there is a sufficient increase of the objective, based on the two following rules:

1) *Armijo rule*:
$U(\mathbf{y}_i + \tau_i \mathbf{d}_i) - U(\mathbf{y}_i) \geq c_1 \tau_i \langle \nabla U(\mathbf{y}_i), \mathbf{d}_i \rangle$.
2) *Curvature condition*:
$|\langle \nabla U(\mathbf{y}_i + \tau_i \mathbf{d}_i), \mathbf{d}_i \rangle| \leq c_2 |\langle \nabla U(\mathbf{y}_i), \mathbf{d}_i \rangle|$

with $0 < c_1 < c_2 < 1$.

C. Direction Calculation

This is a standard LP where the objective function is now $\langle \nabla U(\mathbf{y}), \mathbf{d} \rangle$ subject to the constraints $A\mathbf{y} = \mathbf{b}$. This can be solved by any one of the techniques explained in Sections II-D and II-E.

1) Steepest Direction: In principle, the Frank-Wolfe method requires that the direction-finding problem be solved at optimality at each iteration. The LP can be solved with either the optimal or the exact positive pricing technique since they both produce an optimal solution. Calculating an optimal solution finds the vertex that produces the largest increase in the neighborhood of the current solution and is similar to a steepest descent. In general, this may need fewer iterations but may be quite slow, especially if the LP has a hard pricing sub-problem.

2) Feasible Direction: We can speed up the direction calculation if we just compute any feasible direction where the function increases, i.e., any direction with a positive projection on the gradient. This can be done by our positive pricing algorithm.

A feasible direction can be computed with the positive pricing algorithm where we do not necessarily run the procedure to termination. The idea is that the computation-intensive enumeration step is not invoked as long as we can make a sufficiently large progress with only the greedy pricing. This is equivalent to the two conditions

$$\frac{\langle \nabla U(\mathbf{y}_i), \mathbf{d}_i \rangle}{\|\nabla U\| \|\mathbf{d}\|} \geq \delta > 0 \tag{12}$$

$$\frac{\|\mathbf{y}_i - \mathbf{y}_{i-1}\|}{\|\mathbf{y}_i\|} \geq \theta > 0 \tag{13}$$

for some given parameters δ and θ. Eq. (12) states that the current direction is sufficiently close to the gradient and Eq. (13) that the solution vector has changed by a sufficiently large amount. If neither condition holds, the enumeration-based column generator is used. This insures the eventual convergence to optimality since no feasible direction with an increasing value of the objective will be left out.

A formulation of the first-order optimality condition [20] is that at an optimal point, there is no feasible direction with a positive projection on the gradient. In our case, the algorithm terminates if the projection of the direction on the gradient is small enough,

$$\frac{\langle \nabla U(\mathbf{y}_i), \mathbf{d}_i \rangle}{\|\nabla U\| \|\mathbf{d}\|} \leq \delta' \tag{14}$$

for $0 < \delta' \ll 1$. The method falls in the class of *feasible direction* techniques. At each iteration, a small move in the direction that has been computed remains feasible and the direction is such that the function is not decreasing in that direction. Like all feasible direction methods, this one is also susceptible to jamming where the algorithm does not make any progress even though the current point may be far from the optimal solution. We have not observed this in our numerical work, as we have computed the norm of the first-order optimality conditions to guarantee that the final solution was in fact close to the optimum.

D. Nonlinear SRPR

The problem presented in this section is an extension of the LSP problem: it jointly optimizes scheduling, routing, and flow rate allocation. It differs from the SRPR problem [21] [3] in that it has a nonlinear objective function that computes a solution which is proportionally fair to all the flows.

1) Network Model and Problem Formulation: In addition to the PHY and MAC layer models described in Section II-A and II-C, we need to model the network layer. We define a flow $f \in \mathcal{F}$ as a source-destination pair (f_s, f_d) for $f_s, f_d \in \mathcal{N}$. Flow f has a rate λ^f at which the source is generating traffic. Let $\boldsymbol{\lambda} = [\lambda^1, \cdots, \lambda^F]$ be the flow rate vector. For convenience, we define the node-flow incidence vector $\mathbf{d}^f = [d_i^f]^T_{i \in \mathcal{N} \setminus \{f_d\}}$ for each node i and flow f as

$$d_i^f = \begin{cases} 1 & \text{if } i = f_s \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

In general, there is not always a direct link between f_s and f_d so that the traffic has to be routed through some intermediate nodes. For this, we also define the standard node-arc incidence matrix $A^f = [a_{i,l}^f]_{i \in \mathcal{N} \setminus \{f_d\}, l \in \mathcal{L}}$, where for each node i and link $l = (o(l), d(l))$, we have

$$a_{i,l}^f = \begin{cases} +1 & \text{if } i = o(l) \\ -1 & \text{if } i = d(l) \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

The dependence on f is useful when one wants to prevent certain flows from using some links. This in turn leads to a set of decision variables $x_{i,j}^f$ which are the link loads defined in Section II-C. We denote by $\mathbf{x}^f$ a vector of link loads for f and by $\mathbf{x}$ the concatenation of all $\mathbf{x}^f$s.

The proportional fairness problem is shown in the following. The maximization is explicitly taken with respect to the flow rate allocation vector $\boldsymbol{\lambda}$, link load allocation vector $\mathbf{x}$, and link scheduling vector $\boldsymbol{\alpha}$.

$$\max_{\lambda,\mathbf{x},\boldsymbol{\alpha}\succeq\mathbf{0}} \quad \sum_f \log \lambda^f \tag{17}$$

$$(\mu_i^f) \qquad A^f\mathbf{x}^f \;\geq\; \lambda\mathbf{d}^f \qquad \forall\; f \in \mathcal{F} \tag{18}$$

$$(\nu_l) \qquad c_l \sum_{s\in\mathcal{I}} q_{l,s}\alpha_s \;\geq\; \sum_{f\in\mathcal{F}} x_l^f \qquad \forall\; l \in \mathcal{L} \tag{19}$$

$$(\zeta) \qquad \sum_{s\in\mathcal{I}} \alpha_s \;\leq\; 1 \tag{20}$$

where we have put the Lagrangian multipliers corresponding to each constraint in parenthesis. Constraint (18) is the multicommodity flow conservation equation, the rest is just the link scheduling problem defined in Section II-C. In this formulation, we have $\mu_i^f \geq 0$, $\nu_l \geq 0$ and $\zeta \leq 0$.

2) Pricing and Weight Selection: The reduced price of a potential column α_s is

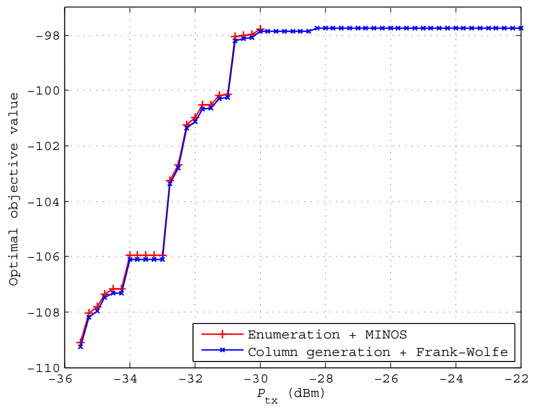
$$r_s = \zeta + \sum_{l\in\mathcal{L}} c_l\nu_l q_{l,s} \tag{21}$$

where ν_l and ζ are the Lagrangian multipliers of constraints (19) and (20). Therefore, the positive pricing needs to identify an ISet s whose r_s is positive, and the column generation terminates if $r_s \leq 0, \forall s \in \mathcal{I}$.

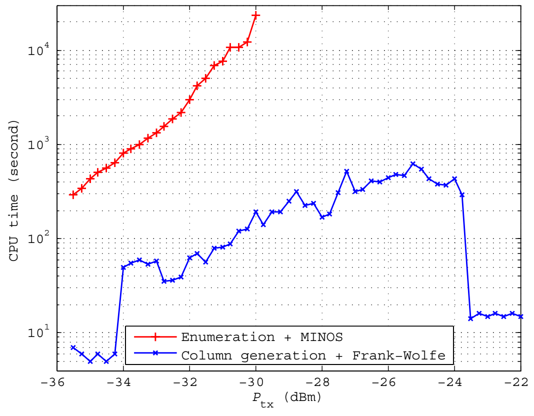
Other than a slight difference in the pricing formula, the pricing algorithm is the same as that presented in Section II-E3. First, the design of the greedy pricing should exploit the structure of the interference constraints, rather than simply relying on the link price $c_l\nu_l$. For example, the interference constraints are sometimes defined by a *conflict graph* [12] in which a vertex represents a link and an edge joins two vertices if the corresponding links conflict with each other. In this case, the link weight used in the greedy heuristic should involve, in addition to $c_l\nu_l$, the vertex degree in the conflict graph. Secondly, the Q matrix is divisible under commonly used interference constraints; this makes the enumeration easier whenever it is needed. Finally, since it is rare that a load vector $\mathbf{x}$ saturates all links whose load is positive, dual degeneracy almost always happens due to the complementary slackness. As we explained in Section II-E3, the positive pricing can be very efficient in this case.

E. Numerical Results

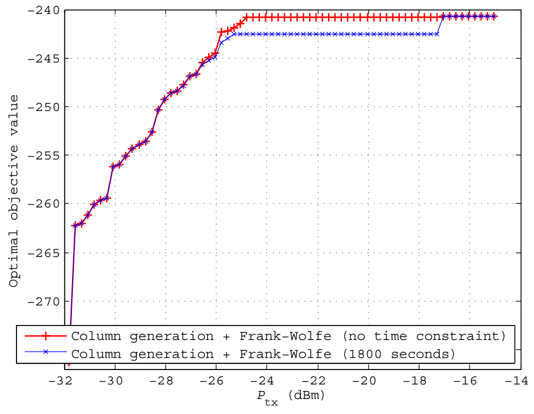
In this section, we report two sets of results to show how our algorithm can handle relatively large networks in reasonable computation time. The first and second cases involve two networks with 30 and 60 nodes randomly deployed in areas of 36×36 m^2 and of 56×56 m^2, respectively. For each network, we require every node to send a flow to the node in the center, which represents the *gateway*. This is a typical scenario for wireless mesh or sensor networks. For radio propagation, the channel gain G between two points separated by distance d is assumed to be given by $F_l(d/d_0)^{-\eta}$, where d_0 is the close-in reference distance, F_l is the shadowing and fading gain and η is the path loss exponent. We assume $d_0 = 0.1$ m,



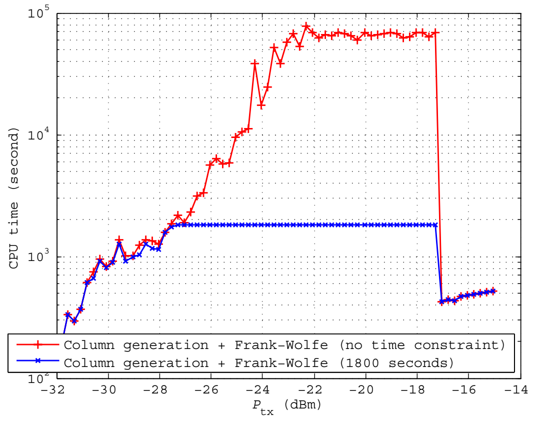
(a) Rand30 objective values.



(b) Rand30 computation time.



(c) Rand60 objective values.



(d) Rand60 computation time.

Fig. 3. Comparison between column generation and enumeration (a),(b) and between column generations with different time constraints (c),(d).

$F_l = 1, \forall\ l \in \mathcal{L}$ and $\eta = 3$. The link rate c_l is normalized to 1 for all links and the corresponding SINR threshold is set to $\beta = 6.4$dB. We also assume that all links have the same transmit power P_{tx} and our computations are done for different values of this power.

In the first case for the 30-node network, we compare the efficiency of two methods by computing the objective values as a function of the transmit power. The first method is based on a complete enumeration of all the ISets as described in Section II-D which is then solved using the commercial nonlinear solver MINOS [8]. The second method is our solution technique based on column generation. Fig. 3 (a) shows the optimal value of the objective function as a function of the power and Fig. 3 (b) the CPU time required to obtain these solutions. Our algorithm is significantly faster than the commercial solver. Note that the commercial solver in unable to produce solutions for large values of the power (i.e., $P_{\text{tx}} > -30$dBm).

In the second case, for the 60-node network, we have used two variants of the column generation algorithm. In the first one, we let the computation run until an optimal solution is found while in the second case, we put a 30 minutes limit on the total CPU time. As we can see in Fig. 3 (c) and (d), the difference in the final value of the objective function is always very small and the two solutions are often identical. This shows that we can get nearly optimal solutions in reasonable time for large networks. Note that a solution technique based on enumeration does not work on such a large network.

V. Conclusion

We revisit a technique based on column generation in this paper. This technique has been shown to be efficient in dealing with a cross-layer design problem with linear objectives [3]. By analyzing the problem structure that makes this technique efficient, we are able to extend it to solve other problems. We first apply this technique to the node on-off scheduling problem, in which case we can handle problems with hundreds of nodes easily. We also extend this technique to address cross-layer design problems with nonlinear objectives. The numerical results show that our algorithm can solve problems that a commercial solver is unable to tackle.

We believe that the same approach can be applied to solve other resource allocation problems. For example, we have been able to formulate a type of relay or gateway placement problem using the extension formulation and have also derived the pricing formula for it. Due to the page limitation, we leave these results for future work.

References

[1] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Trans. on Information Theory*, vol. 46, no. 2, pp. 388–404, 2000.

[2] M. Johansson and L. Xiao, "Cross-Layer Optimization of Wireless Networks Using Nonlinear Column Generation," *IEEE Trans. on Wireless Communications*, vol. 5, no. 2, pp. 435–445, 2006.

[3] J. Luo, C. Rosenberg, and A. Girard, "Engineering Wireless Mesh Networks: Joint Scheduling, Routing, Power Control and Rate Adaptation," Submitted to IEEE/ACM Transactions on Networking (a short version presented in PIMRC'08), April 2009.

[4] D. Villeneuve, J. Desrosiers, M. Lübbecke, and F. Soumis, "On Compact Formulations for Integer Programs Solved by Column Generation," *Annals of Operations Research*, vol. 139, no. 1, pp. 375–388, 2005.

[5] G. Brar, D. Blough, and P. Santi, "Computationally Efficient Scheduling with the Physical Interference Model for Throughput Improvement in Wireless Mesh Networks," in *Proc of the 12th ACM MobiCom*, 2006.

[6] T. Moscibroda, Y. Oswald, and R. Wattenhofer, "How Optimal are Wireless Scheduling Protocols?" in *Proc. of the 26th IEEE INFOCOM*, 2007.

[7] "ILOG CPLEX 11.0." [Online]. Available: http://www.ilog.com/products/cplex/

[8] "MINOS 5.5." [Online]. Available: http://www.sbsi-sol-optimize.com/asp/sol_product_minos.htm

[9] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu, "Impact of Interference on Multi-hop Wireless Network Performance," in *Proc. of the 9th ACM MobiCom*, 2003.

[10] A. Karnik, A. Iyer, and C. Rosenberg, "Throughput-Optimal Configuration of Wireless Networks," *IEEE/ACM Trans. on Networking*, vol. 16, no. 5, pp. 1161–1174, 2008.

[11] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. New York: Wiley, 1988.

[12] P. Björklund, P. Värbrand, and D. Yuan, "Resource Optimization of Spatial TDMA in Ad Hoc Radio Networks: A Column Generation Approach," in *Proc. of the 22th IEEE INFOCOM*, 2003.

[13] J. Zhang, H. Wu, Q. Zhang, and B. Li, "Joint Routing and Scheduling in Multi-radio Multi-channel Multi-hop Wireless Networks," in *Proc. of the 24th IEEE INFOCOM*, 2005.

[14] A. Capone and G. Carello, "Scheduling Optimization in Wireless Mesh Networks with Power Control and Rate Adaptation," in *Proc. of the 3rd IEEE SECON*, 2006.

[15] M. Cao, X. Wang, S.-J. Kim, and M. Madihian, "Multi-Hop Wireless Backhaul Networks: A Cross-Layer Design Paradigm," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 4, pp. 738–748, 2007.

[16] S. Slijepcevic and M. Potkonjak, "Power Efficient Organization of Wireless Sensor Networks," in *Proc. of IEEE ICC*, 2001.

[17] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky, "Power Efficient Monitoring Management in Sensor Networks," in *Proc. of IEEE WCNC*, 2004.

[18] M. Cardei, M. Thai, Y. Li, and W. Wu, "Energy-Efficient Target Coverage in Wireless Sensor Networks," in *Proc. of the 24th IEEE INFOCOM*, 2005.

[19] H. Gupta, S. Das, and Q. Gu, "Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution," in *Proc. of the 4th ACM MobiHoc*, 2003.

[20] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, Massachusetts: Athena Scientific, 1999.

[21] J. Luo, A. Iyer, and C. Rosenberg, "Throughput-Lifetime Tradeoffs in Multihop Wireless Networks under a Realistic Interference Model," in *Proc. of the 45th Allerton Conference*, 2007.