2011-07-22

Modeling Wireless Networks for Rate Control

David C. Ripplinger
Brigham Young University - Provo

Modeling Wireless Networks for Rate Control

David C. Ripplinger

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2011

ABSTRACT

Modeling Wireless Networks for Rate Control

David C. Ripplinger
Department of Computer Science, BYU
Master of Science

Congestion control algorithms for wireless networks are often designed based on a model of the wireless network and its corresponding network utility maximization (NUM) problem. The NUM problem is important to researchers and industry because the wireless medium is a scarce resource, and currently operating protocols such as 802.11 often result in extremely unfair allocation of data rates. The NUM approach offers a systematic framework to build rate control protocols that guarantee fair, optimal rates. However, classical models used with the NUM approach do not incorporate partial carrier sensing and interference, which can lead to significantly suboptimal performance when actually deployed.

We quantify the potential performance loss of the classical controllers by developing a new model for wireless networks, called the first-principles model, that accounts for partial carrier sensing and interference. The first-principles model reduces to the classical models precisely when these partial effects are ignored. Because the classical models can only describe a subset of the topologies described by the first-principles model, the score for the first-principles model gives an upper bound on the performance of the others. This gives us a systematic tool to determine when the classical controllers perform well and when they do not. We construct several representative topologies and report numerical results on the scores obtained by each controller and the first-principles optimal score.

Keywords: wireless networks, rate control, modeling, partial carrier sensing, partial interference

ACKNOWLEDGMENTS

Thanks go to Christopher Grant in the BYU Math Department for his help with developing the theorems and definitions for uniform random sets.

Table of Contents

List of Figures **vi**

List of Tables **viii**

1 Introduction **1**

2 The Maximal Clique Model **5**

2.1 Constraints . . . 5

2.2 Objective . . . 7

2.3 Solution and Controller Design . . . 9

3 The Partial Interference Model **13**

3.1 Constraints . . . 13

3.2 Objective . . . 15

3.3 Solution and Controller Design . . . 15

3.4 Comparison to the Maximal Clique Model: Numerical Results . . . 17

3.4.1 Performance Metric . . . 17

3.4.2 Results . . . 18

4 The First-Principles Model **23**

4.1 A Simple Example . . . 25

4.2 Union of Uniform Random Sets . . . 26

4.3 Constraints . . . 28

4.4 Objective . . . 31

5 Reduction of the First-Principles Model to the Classical Models 33

5.1 Reduction to the Maximal Clique Model 33

5.2 Reduction to the Partial Interference Model 35

6 Solution to the First-Principles Optimization Problem 37

6.1 Tightening the Upper Bound 39

6.2 Implementation 42

7 Performance of the Classical Controllers: Numerical Results 45

7.1 Two-Link Topologies 46

7.2 Three-Link Topologies 47

7.3 Partial Interference Topologies 51

7.4 Partially Dependent Interferers 52

7.5 Chain Topology 54

7.6 Mesh Topology 56

7.7 Summary of Results 57

8 Conclusion 59

A Branch and Bound Code 63

References 85

List of Figures

2.1 An example wireless network and its contention graph 6

3.1 Topologies used to compare classical controllers 19

3.2 Performance of classical controllers with I interferers on one link 19

3.3 Performance of classical controllers with N contenders with one interferer . . 20

3.4 Performance of classical controllers with I interferers and N contenders . . . 21

4.1 Example of the first-principles model, random overlap 25

6.1 Example decision tree for the branch and bound method 38

7.1 Two-link topologies 46

7.2 Optimality CDFs of the classical controllers for the two-link topologies . . . 48

7.3 Infeasibility CDFs of the classical controllers for the two-link topologies . . . 48

7.4 Three-link topologies varied over interference 48

7.5 Optimality CDFs for the three-link topologies varied over interference 49

7.6 Three-link topologies varied over contention 50

7.7 Optimality CDFs for the three-link topologies varied over contention 50

7.8 Optimality CDFs for the partial interference topologies 51

7.9 Maximal clique controller performance for partial interference topologies . . 52

7.10 Partial interference controller performance for partial interference topologies 53

7.11 Topologies with partially dependent interferers 53

7.12 Optimality of classical models for partially dependent interferers 54

7.13 Chain topology 54

7.14 Mesh topology 56

List of Tables

2.1 Notation used in the maximal clique model 5

3.1 Notation used in the partial interference model 13

4.1 Notation used in the first-principles model 24

7.1 Parameters in the mesh topology 57

Chapter 1

Introduction

Wireless networks are often used as a low-cost alternative to wired infrastructure, while also accommodating mobile users. The most prevalent medium access control (MAC) protocol used in wireless networks is defined in the IEEE 802.11 standard. However, research has shown that when the wireless network is extended to multiple hops, the 802.11 MAC is plagued with serious fairness and efficiency problems, sometimes completely starving one data flow in favor of another [1, 2, 3]. This is in part due to the fact that sharing resources in a wireless network is a fundamentally different problem than in a wired network, and requires some theoretical understanding to solve.

As a result of these problems, rate allocation in wireless networks to achieve maximum network utilization and fairness has become a popular area of research. Seminal research in this area includes [4, 5] for wired networks, and has been readily extended to wireless networks in [6, 7, 8, 9, 10, 11, 12]. See [13] for a survey on this and other active research topics for wireless networks.

Some research seeks to improve the capacity of a mesh network by introducing multiple radios operating on different frequencies, or by manipulating the transmission power of the radios in order to reduce contention [14, 15]. We view this work as complementary to ours. We limit the scope of our research area to designing transmission rate controllers for either communication flows or links, where algorithms manipulating routes, radio frequencies, antenna direction, or transmission power are already employed, and such algorithms are quasi-static with respect to the rate controller.

In this thesis, we will answer the following question: *When do certain classical controllers behave poorly due to partial carrier sensing and interference effects?* Congestion control algorithms are often designed based on a model of the wireless network and its corresponding network utility maximization (NUM) problem. In the network utility maximization (NUM) approach, an objective function for the network is defined, typically a sum of utility functions for each link's or flow's sending rate, where the form of the utility function defines a particular notion of fairness. Next, a set of constraints is used to model the unique characteristics of the wireless network, such as carrier sensing or interference constraints. The solution to this optimization problem will then yield a set of rates that maximize network utility for the links or flows. These rates can then be used as input to a rate controller that sits on top of (or inside) the MAC protocol, limiting the packet transmission rate for each flow or link. When the optimization problem is convex, it can often be translated into a distributed rate control algorithm, making it practical to deploy in a wireless network.

Our focus is on the constraints used to model the wireless network, as this is the critical piece in the NUM approach. If the model is inaccurate, then the optimization problem may not yield an accurate or optimal solution. We limit our study to stationary, multi-hop wireless networks that use carrier sense multiple access (CSMA), such as the 802.11 MAC. CSMA protocols work by each node listening to see if the medium is occupied before sending data, as opposed to working out a precise schedule with other nodes of which times to send. This broadly characterizes the most widely-used wireless networks in the field, often referred to as mesh networks.

We present a first-principles model of wireless networks for the rate control or NUM problem. By first-principles, we mean that the most basic assumptions are made of how multi-hop wireless networks with CSMA operate. In this model, perceived times that the medium is occupied are represented as a random set. Our model may also be classified as a measurement based model, as it takes as inputs the probabilities of links carrier sensing or interfering with each other. Such an approach is more realistic than physical layer modeling, such as the various signal fading models with SINR thresholds, and has been used to model wireless networks for various purposes [16, 17, 18, 19, 20, 21, 11]. A combination of measurement based modeling and physical layer modeling is used in [18] to determine probabilities of carrier sensing and interference between pairs of nodes. Kashyap, et al [20] extends this idea to also model probabilities of carrier sensing and interference of groups of nodes in order to determine the capacity of a wireless link. We note specifically that their approach is very similar to ours, modeling the states (transmitting, deferring, idle) of a node as random sets. However, they also model specific aspects of the 802.11 MAC as opposed to generic CSMA, and their model predicts an uncontrolled environment, where there is no rate controller other than the 802.11 MAC. In the future, we hope to combine the qualities of ours and Kashyap's models to achieve further accuracy for rate control problems.

We show that, under limiting conditions, our model reduces to previously proposed models in the literature. Specifically, with the assumption of binary, symmetric sensing, our model reduces to the common maximal clique model used by seminal research in this area [6, 10]. Likewise, with the assumption of no carrier sensing between interfering links, our model reduces to the partial interference model developed by Niculescu [21] and later employed in [12].

Because the first-principles model induces a non-convex NUM problem, it is too complex to deploy an efficient, distributed rate controller that solves it. However, we can solve it offline using a branch and bound method on several constructed network topologies that are representative of scenarios that often occur in real wireless networks. The objective function of our first-principles NUM problem can serve as a scoring mechanism for various rate inputs that any controller may compute, and the optimal solution will be an upper bound on the performance of the classical NUM-based rate controllers. We implement a branch and bound algorithm to solve the problem, and we compare its solution (the upper bound) on several representative topologies with the scores achieved by the classical NUM-based controllers. In this way, we determine under what kinds of topologies partial carrier sensing and interference adversely affect the performance of the classical controllers.

Some interesting scenarios were found in which the classical controllers performed as low as 33% of the optimal. The maximal clique controller was found to perform poorly in many small scenarios, especially those in which there was much partial (and little binary) carrier sensing or interference. Much fewer cases were found when the partial interference controller performed poorly. These were primarily limited to cases of significant partial carrier sensing (with little or no interference in the network), and of very heavy interference by several links that perfectly carrier sensed each other. We also computed the performance of the classical controllers on a chain topology and a mesh topology, which were larger and more typical of real networks than the toy topologies we explored. In both cases, the maximal clique controller performed above 86% and the partial interference controller above 97% of the optimal. We believe that these results are a strong indication that the partial interference model is good enough in most networks for rate control. Furthermore, we now have a method of detecting poor performance in a network, along with motifs that are easy to isolate as the culprits.

Chapter 2

The Maximal Clique Model

The maximal clique model is the most widely used model for rate optimization in wireless networks [6, 10]. The notation used in this model is given in Table 2.1.

2.1 Constraints

Figure 2.1 shows how a contention graph is inferred from a wireless network. This graph has a vertex representing each active link, and each edge signifies that two links *contend*, or cannot send at the same time. The maximal cliques are then identified. A clique is a subgraph such that each pair of its vertices share an edge. A maximal clique is a clique such that no other vertex in the parent graph could be included to form another clique. For each maximal clique j, its links' sending rates s must sum to at most some clique capacity, which for our purposes will always be 1:

$$\sum_{i \in L(j)} s_i \leq 1, \quad \forall j \in C. \tag{2.1}$$

s_i	Sending rate of link i.
L	Set of links.
C	Set of maximal cliques.
$L(j)$	Set of links in maximal clique j.
$C(i)$	Set of maximal cliques containing link i.

Table 2.1: Notation used in the maximal clique model.

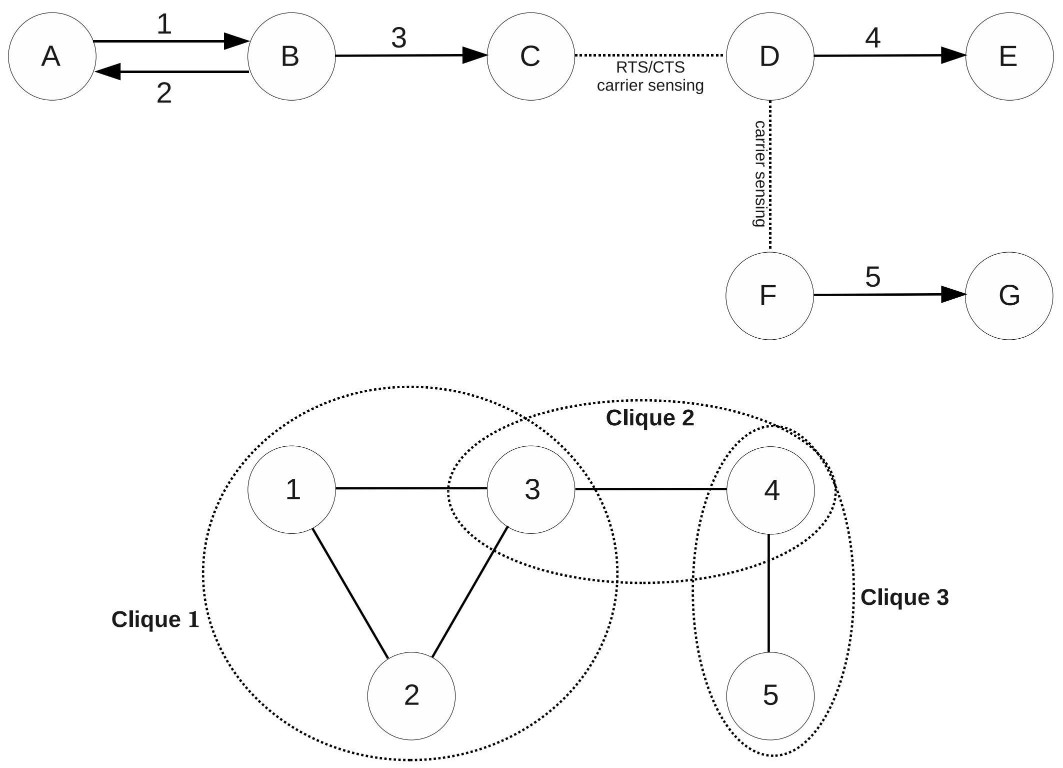


Figure 2.1: An example wireless network and its corresponding contention graph with the maximal cliques circled. Links 1, 2, and 3 contend because they share node B. Links 4 and 5 contend because sending nodes D and F are within carrier sensing range of each other. Assuming RTS/CTS is enabled, links 3 and 4 contend because sending node D is within carrier sensing range of receiving node C, which sends out CTS signals. This model infers that links in a maximal clique have sending rates summing to at most some clique capacity.

The constraints specific to Figure 2.1 are

$$s_1 + s_2 + s_3 \leq 1, \tag{2.2}$$

$$s_3 + s_4 \leq 1, \tag{2.3}$$

$$s_4 + s_5 \leq 1. \tag{2.4}$$

Allowing non-unity values for clique capacities is a technique to account for less than ideal usage of the medium and indirect scheduling conflicts. The latter is demonstrated in [6] by showing a ring contention graph of 5 links, where (2.1) with clique capacities of 1 is inaccurate, predicting that each $s_i = 1/2$. This allocation causes a scheduling conflict. If links 1 and 3 send during the first half of a block of time, and links 2 and 4 send during the second half, link 5 cannot send at all, since it contends with link 1 during the first half and link 4 during the second half. Measuring clique capacities less than one would allow the model to constrain the sending rates only to those such that a feasible schedule exists.

One of the advantages of building the contention graph is that contention can be interpreted mathematically as a simple sum of rates being constrained, no matter the reason for contention. The most common reasons for contention are that two links share the same sending node or their sending nodes are within carrier sensing range of each other. Note, however, that any reason for two links contending can be appropriately modeled by the contention graph. For example, in Figure 2.1, links 3 and 4 contend because RTS/CTS (request-to-send/clear-to-send) is enabled, so that sending node D is within carrier sensing range of receiving node C. Node C will send out CTS signals to B, but D will overhear the CTS signals and defer the medium.

2.2 Objective

The objective function in the optimization problem must yield high throughput while also providing fairness between flows. It is well known that these two goals are conflicting for wireless networks — to provide fairness, some flows may need to sacrifice throughput. A systematic way of achieving a balance between the two is by constructing the objective function as a sum of rate utility functions, which are concave and strictly increasing. The type of fairness is determined by the utility function's degree of concavity. Several notions of fairness and their corresponding utility functions are well established in the literature [22, 23].

The two most common notions of fairness are max-min fairness and proportional fairness. Max-min fairness can be understood as the rates achieved by increasing all the sending rates from zero simultaneously until a constraint is active. Then continue to increase the sending rates for those links that still have not been affected by some active constraint until the next constraint becomes active, repeating the process until all links have been maximized. These rates are achieved in the NUM problem by choosing the utility function

$$U(s_i) = s_i^{-\alpha}/\alpha, \qquad \alpha \to \infty, \tag{2.5}$$

which can be approximated by choosing a large α. Proportional fairness seeks to achieve equal rates among links, but it allows for one link to lower its rate somewhat if it allows for multiple other links to then increase their rates a significant amount. It is achieved in the NUM problem by choosing the utility function

$$U(s_i) = \ln s_i. \tag{2.6}$$

For this and subsequent models, we will only consider proportional fairness for two reasons. First, max-min fairness often results in significant lost overall bandwidth in the network. Second, while there exists a distributed solution to the maximal clique NUM problem for any notion of fairness, the logarithms in proportional fairness allow decoupling of variables in subsequent models, so that a distributed solution exists for the partial interference NUM problem, and a tighter upper bound on the first-principles NUM problem can be calculated.

For our purposes then, the objective function for the maximal clique model is

$$f(s) = \sum_{i \in L} \ln s_i. \tag{2.7}$$

2.3 Solution and Controller Design

Here we show how the maximal clique NUM problem can be solved distributively and the high-level considerations in deploying a controller to implement it. The development is adapted from [10]. It is implicit in this and all problems in this thesis that rates are constrained to be non-negative.

The problem is

$$\begin{array}{ll} \text{maximize} & \sum_{i \in L} \ln s_i \\ \text{subject to} & \sum_{i \in L(j)} s_i \leq 1, \quad \forall j \in C. \end{array} \tag{2.8}$$

To solve it, we write the Lagrangian, which is constructed by adding to the objective function a linear combination of the constraint functions:

$$L(s, \lambda) = \sum_{i \in L} \ln s_i + \sum_{j \in C} \lambda_j \left(1 - \sum_{i \in L(j)} s_i \right). \tag{2.9}$$

Each λ_j can be thought of as a price for maximal clique j, or the per-unit cost of breaking constraint j. They are also called the Lagrange multipliers. Note that maximizing the Lagrangian over s can be thought of as a relaxation of the original problem. Instead of breaking the constraints being infinitely bad, one only incurs a finite cost. Before moving on, we first rearrange the terms in (2.9) to get

$$L(s, \lambda) = \sum_{i \in L} (\ln s_i - \Lambda_i s_i) + \sum_{j \in C} \lambda_j, \tag{2.10}$$

where $\Lambda_i = \sum_{j \in C(i)} \lambda_j$ is the sum of prices for each clique that link i belongs to.

Next we construct the dual function, which is simply the Lagrangian maximized over s:

$$D(\lambda) = \max_{s} \sum_{i \in L} (\ln s_i - \Lambda_i s_i) + \sum_{j \in C} \lambda_j. \tag{2.11}$$

The max can be brought inside the sum to get

$$D(\lambda) = \sum_{i \in L} \max_{s_i} (\ln s_i - \Lambda_i s_i) + \sum_{j \in C} \lambda_j. \tag{2.12}$$

The function

$$\rho(s_i, \lambda) = \ln s_i - \Lambda_i s_i \tag{2.13}$$

being maximized over s_i is concave and not monotonic, so that a unique maximizer $\bar{s}_i(\lambda)$ is guaranteed. We can solve for it analytically by taking the derivative and setting it equal to zero:

$$\bar{s}_i(\lambda) = \Lambda_i^{-1}. \tag{2.14}$$

Substituting this into (2.12) results in

$$D(\lambda) = \sum_{i \in L} \left(\ln \Lambda_i^{-1} - 1 \right) + \sum_{j \in C} \lambda_j. \tag{2.15}$$

The dual function, for any non-negative λ, gives an upper bound on the original objective function in (2.8). In order to tighten this upper bound, we solve the dual problem, which is to minimize the dual function over λ:

$$\begin{array}{lll} \text{minimize} & \sum_{i \in L} \left(\ln \Lambda_i^{-1} - 1 \right) + \sum_{j \in C} \lambda_j & \\ \text{subject to} & \lambda_j \geq 0, & \forall j \in C. \end{array} \tag{2.16}$$

When the optimal score for both the primal problem and the dual problem are equal, it is said that they have strong duality. For convex problems, strong duality holds when Slater's condition is met, that is, there exists a feasible point in the relative interior of the domain. Slater's condition holds for all problems in this thesis. We thus have strong duality for (2.8) and (2.16), and the corresponding optimal sending rates are $s^* = \bar{s}(\lambda^*)$, where λ^* are the optimal prices.

The dual problem is solved using the gradient descent method. From Danskin's Theorem [24], we know that

$$\frac{\partial D}{\partial \lambda_j} = 1 - \sum_{i \in L(j)} \bar{s}_i. \tag{2.17}$$

Using a step size γ in the negative direction of the gradient gives the algorithm

$$\lambda_j(k+1) = \max\left\{0,\quad \lambda_j(k) - \gamma\left(1 - \sum_{i \in L(j)} \bar{s}_i(\lambda(k))\right)\right\}. \tag{2.18}$$

Each link in a maximal clique can share with each other their current values of $\bar{s}_i$, which is only local information, and then compute the next λ_j. The convergence of the algorithm is well established in the literature, even when it is asynchronous [5].

There are some difficulties with actually implementing this controller. It is well known that discovering all maximal cliques in a graph is NP-hard, so it would be necessary to approximate the set of maximal cliques. Also, if non-unity clique capacities are used in the model in order to make it more accurate, these capacities would have to be remeasured each time the optimization problem would need to be solved. Finally, the gradient descent method is frequently subject to slow convergence depending on the particular problem and choice of step size, in comparison to Newton-based methods. An implementation was deployed on a real network in [10], which had convergence rates on the order of tens of seconds.

Chapter 3

The Partial Interference Model

The partial interference model supplements the maximal clique model with constraints on the receiving rates [12]. The model is based on an empirical study of carrier sensing and interference in a wireless mesh network [21]. The notation used in this model is given in Table 3.1.

3.1 Constraints

In the partial interference model, an interfering node may corrupt a fraction of the packets received at a remote node. Partial interference is not represented in the contention graph, but is instead represented in a directional, weighted interference map or matrix, and is incorporated as a constraint on receiving rates. To model partial interference accurately, we separate contention constraints from interference constraints. Contention is represented as an undirected edge between two vertices (links), and interference is modeled as a direc-

s_i	Sending rate of link i.
r_i	Receiving rate of link i.
d_i	Delivery ratio of link i.
a_{ij}	Receiving interference probability of link j interfering with link i.
L	Set of links.
L_i	Set of all links in L except i.
C	Set of maximal cliques.
$C(i)$	Set of maximal cliques containing link i.
$L(j)$	Set of links in maximal clique j.

Table 3.1: Notation used in the partial interference model.

tional, weighted edge from the interfering link to the receiving link that is affected by the interference.

The constraint on each receiving rate is

$$r_i = d_i s_i \prod_{j \in L_i} (1 - a_{ij} s_j), \quad \forall i \in L, \tag{3.1}$$

where the delivery ratio d_i implies inherent loss over the link. In Figure 2.1, assume RTS/CTS is disabled so that node D can corrupt some packets received at C. The contention graph of Figure 2.1 would then be modified by replacing the edge between links 3 and 4 with a directional edge with weight a_{34}. Then (2.3) is dropped and in its stead we have

$$r_3 = d_3 s_3 (1 - a_{34} s_4). \tag{3.2}$$

The partial interference model is less conservative than the maximal clique model because more links are assumed to transmit concurrently. For example, consider links 3 and 4 in Figure 2.1, and suppose link 4 corrupts 40% of packets received at link 3. If both links transmit at the clique capacity 1, then the sum of their effective receiving rates becomes $1 + (1 - 0.4) = 1.6$. In the maximal clique model, these links cannot transmit at the same time because they are in the same clique, resulting in a total effective receiving rate of 1. The partial interference model thus allows for significantly higher utilization of the network in certain scenarios.

It is important to recognize that even complete interference cannot be accurately modeled as contention. That is, a link j will not become a contender to a remote link i even if the interference factor $a_{ij} = 1$. Consider again the relationship between links 3 and 4 in Figure 2.1. Suppose $a_{34} = 1$. If interference is modeled as contention, then both links will transmit at a rate of 0.5. However, the total effective receiving rate will be $r_2 + r_3 = (0.5)(0.5) + 0.5 = 0.75$. With the partial interference model, it is easy to see that link 3 should send at the full rate regardless of link 4's rate. If link 4 continues to send at a rate of 0.5, then the total effective receiving rate will be $(1)(0.5) + 0.5 = 1$. Thus the partial interference model will result in higher utility.

3.2 Objective

This leads to modifying the objective function. Note that we now care about receiving rates instead of sending rates. Therefore, the objective function for the partial interference model is

$$f(r) = \sum_{i \in L} \ln r_i. \tag{3.3}$$

Here we also see why choosing proportional fairness is helpful mathematically. If r_i in the objective function is replaced by (3.1), each term in the product converts to a sum of logarithms, so that each term in the summation is dependent on only one variable in s.

3.3 Solution and Controller Design

The partial interference NUM problem can be solved distributively almost identically to the method for the maximal clique NUM problem. The development is adapted from [12].

Noting that delivery ratios d only add a constant to the objective function (3.3) when written as a function of s, we will simply assume from this point on that $d_i = 1$ for all i. Rearranging the log terms on each s_i in the objective, the primal problem is

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in L} \left(\ln s_i + \sum_{j \in L_i} \ln(1 - a_{ji} s_i) \right) \\ \text{subject to} \quad & \sum_{i \in L(j)} s_i \leq 1, \qquad\qquad \forall j \in C. \end{aligned} \tag{3.4}$$

Just as with the maximal clique NUM problem, we solve (3.4) by constructing the Lagrangian:

$$L(s, \lambda) = \sum_{i \in L} \left(\ln s_i + \sum_{j \in L_i} \ln(1 - a_{ji} s_i) \right) + \sum_{j \in C} \lambda_j \left(1 - \sum_{i \in L(j)} s_i \right). \tag{3.5}$$

We again rearrange the terms to get

$$L(s, \lambda) = \sum_{i \in L} \left(\ln s_i + \sum_{j \in L_i} \ln(1 - a_{ji} s_i) - \Lambda_i s_i \right) + \sum_{j \in C} \lambda_j, \tag{3.6}$$

where $\Lambda_i = \sum_{j \in C(i)} \lambda_j$ as before.

We next construct the dual function:

$$\begin{aligned} D(\lambda) &= \max_s \sum_{i \in L} \rho(s_i, \lambda) + \sum_{j \in C} \lambda_j &(3.7)\\ &= \sum_{i \in L} \max_{s_i} \rho(s_i, \lambda) + \sum_{j \in C} \lambda_j, &(3.8)\end{aligned}$$

where

$$\rho(s_i, \lambda) = \ln s_i - \Lambda_i s_i + \sum_{j \in L_i} \ln(1 - a_{ji} s_i). \tag{3.9}$$

Note that this development only differs from the maximal clique model's development by changing $\rho(s_i, \lambda)$ from (2.13) to (3.9), so that

$$\bar{s}_i(\lambda) = \arg\max_{s_i} \rho(s_i, \lambda) \tag{3.10}$$

produces a different answer by incorporating a cost for each time link i interferes with another link. It is difficult to solve (3.10) analytically, but it is easy to do numerically, for example with Newton's method.

The dual problem is

$$\begin{array}{lll} \text{minimize} & \sum_{i \in L} \rho(\bar{s}_i(\lambda), \lambda) + \sum_{j \in C} \lambda_j & \\ \text{subject to} & \lambda_j \geq 0, & \forall j \in C. \end{array} \tag{3.11}$$

Again, we use (2.18) to solve the dual problem, and the optimal rates are given by $s^* = \bar{s}(\lambda^*)$, where λ^* are the optimal prices.

3.4 Comparison to the Maximal Clique Model: Numerical Results

We seek to determine in what situations the partial interference controller outperforms the maximal clique controller, and by how much. We use MATLAB to numerically compute solutions to the rate optimization problem for several different wireless networks. We use network topologies that represent basic situations — these can be thought of as building blocks out of which larger topologies can be formed.

We introduce three strains of the maximal clique controller that we compare with the partial interference controller. The interference-as-contention (IC) controller replaces any interference mappings with contention, no matter how small the interference factor a. The interference-ignored (II) controller simply ignores any interference mappings and models only contention. The adaptive contention (AC) controller follows the IC controller or the II controller, depending on which controller has higher performance. Thus the AC controller gives the maximal clique model the benefit of the doubt — it ignores interference when this provides good performance and models it as contention otherwise.

3.4.1 Performance Metric

To compare these different controllers, we define a performance metric that is based on the objective function of the partial interference model, using receiving rates. To make the performance metric intuitive, we use the geometric mean of the receiving rates, as this is a monotonic transformation of the partial interference objective function:

$$P(r) = e^{f(r)/|L|} \tag{3.12}$$

The comparison should be made between the performance observed with receiving rates r^* derived from the partial interference controller and the receiving rates r' actually obtained by the other controller from its sending rates s', according to the partial interference constraint on receiving rates. In other words, one of the maximal clique controllers comes up with some sending rates s' that it considers optimal. We take that s' and substitute it into (3.1) to get r'. Then we substitute both r' and the partial interference controller's solution r^* into (3.12) to get their respective scores P.

For ease of interpretation, we consider the ratio R of performances P, that is,

$$R = P(r^*)/P(r'). \tag{3.13}$$

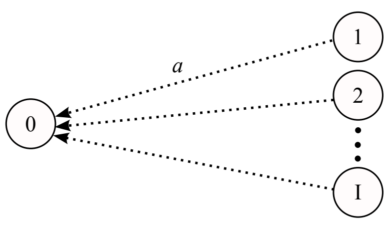
Thus, the comparison will simply read that the partial interference controller outperforms the maximal clique controller R times.

3.4.2 Results

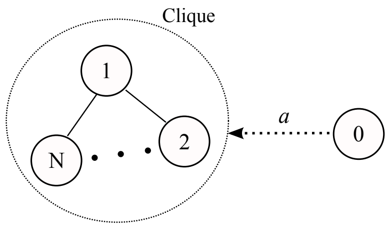
We consider three generic network topologies, shown in Figure 3.1 (the two topologies shown and a hybrid), and plot R for each topology and for each strain of the maximal clique controller being compared with the partial interference controller. Each topology is represented in the figures as a combined contention graph and interference map, where solid lines denote contention and dashed arrows denote the link at the arrow's tail interfering with the link at the arrow's head.

In all cases, the IC controller never does as well as the partial interference controller because modeling interference as contention is too conservative. For low values of interference, it is better to let links send at faster rates and suffer some packet loss. At high values of interference, it is better to have the interfered link send at a faster rate than the interferer, to provide better throughput and fairness. However, modeling interference as contention is often better than ignoring it when interference is high. Thus in most cases, the hybrid AC controller follows the II controller for low values of interference and follows the IC controller for high values of interference.

Figure 3.1(a) shows the first topology, where I links interfere with a single link with a common interference factor a, but do not interfere with each other. Figures 3.2(a), 3.2(b),

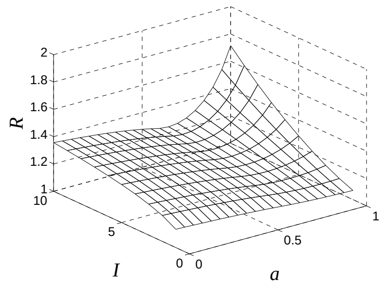


(a) I links interfering with a single link.

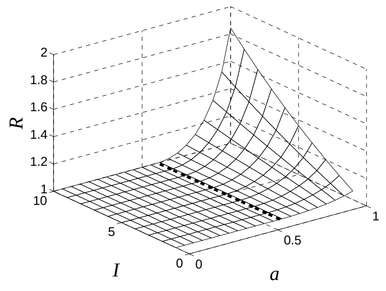


(b) N contenders in a clique with one interferer.

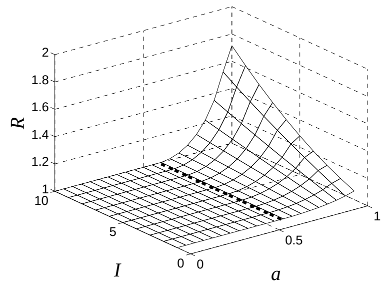
Figure 3.1: Topologies used to compare the performance of the partial interference controller and the maximal clique controller.



(a) Ratio R of performance between the partial interference controller and the IC controller.



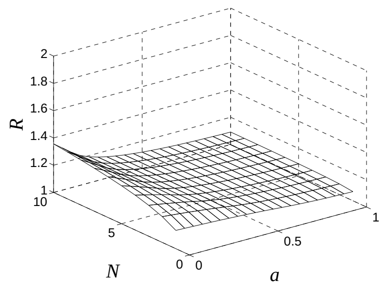
(b) Ratio R of performance between the partial interference controller and the II controller. The dotted line marks where R begins to be greater than one.



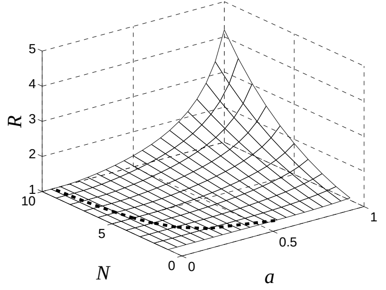
(c) Ratio R of performance between the partial interference controller and the AC controller. The dotted line marks where R begins to be greater than one.

Figure 3.2: Numerical results for the performance of classical controllers with I interferers on one link.

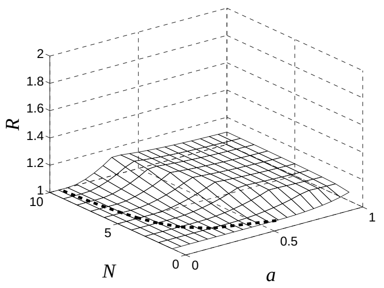
and 3.2(c) plot R for this topology for the partial interference controller against the IC, II, and AC controllers, respectively. The dotted lines show where R begins to be greater than one. Interestingly, the partial interference controller and the II controller perform exactly the same for values of a below 0.59. This is because, for low values of a, the cost of interference is offset by the gain of the interferer sending at full capacity. Thus, both the partial interference controller and the II controller calculate sending rates at full capacity for each link. For larger values of a and I, the partial interference controller outperforms the maximal clique controllers more than 1.5 times.



(a) Ratio R of performance between the partial interference controller and the IC controller.



(b) Ratio R of performance between the partial interference controller and the II controller. The dotted line marks where R begins to be greater than one.

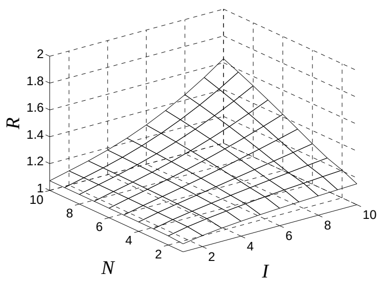


(c) Ratio R of performance between the partial interference controller and the AC controller. The dotted line marks where R begins to be greater than one.

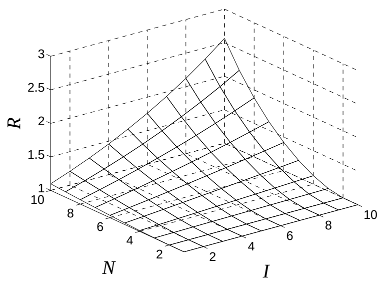
Figure 3.3: Numerical results for the performance of classical controllers with N contenders with one interferer.

Figure 3.1(b) shows the second topology, where a single link has interference factor a on N links that contend in a single clique. Figures 3.3(a), 3.3(b), and 3.3(c) plot R for this topology for the partial interference controller against the IC, II, and AC controllers, respectively. The dotted lines show where R begins to be greater than one. The partial interference controller starts performing better than the II controller at much lower values of a when N is large. This is due to the fact that the contending links already have small rates as a consequence of sharing the medium. Utilities are lowered much more by interference when sending rates are small. Thus, even for low values of a, the partial interference controller does not calculate sending rates at full capacity. However, for higher values of a and N, the partial interference controller outperforms the IC controller only about 1.1 times. The AC controller consequently does relatively well across all values.

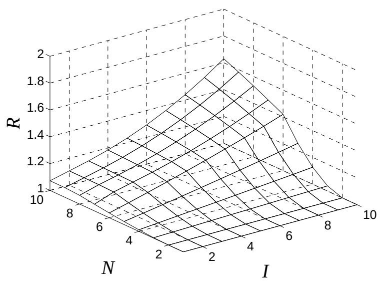
To demonstrate the worth of the partial interference model, we consider a topology combining features of the first two, where I links have a fixed interference factor $a = 0.4$ on N links that contend in a single clique. Figures 3.4(a), 3.4(b), and 3.4(c) plot R for this topology for the partial interference controller against the IC, II, and AC controllers, respectively. Experimental results in [21] show that it is typical for interference factors to



(a) Ratio R of performance between the partial interference controller and the IC controller.



(b) Ratio R of performance between the partial interference controller and the II controller.



(c) Ratio R of performance between the partial interference controller and the AC controller.

Figure 3.4: Numerical results for the performance of classical controllers with I interferers and N contenders, with $a = 0.4$.

range anywhere between zero and one in a real network, with usually at least one interferer on a link having a factor of at least $a = 0.8$, so choosing $a = 0.4$ in this topology is a reasonable comparison. The combined effect of several interferers and several contenders causes the partial interference controller to perform significantly better than the maximal clique controllers.

Chapter 4

The First-Principles Model

The design of the maximal clique and partial interference models (which we will refer to as the classical models) raises some questions. How do we know that mutually contending links (maximal cliques) implies that their rates must sum to at most 1 (or the clique capacity)? How can the maximal clique model be logically extended to consider the case of partial carrier sensing, where there is a continuous range of probabilities that nodes can sense each other, and the notion of cliques is immediately destroyed? Why, in the partial interference model, is the effect of each interfering link multiplicative? We seek to answer these questions by developing a model from a more theoretical standpoint by using random sets to represent observed times the medium is occupied. The notation used in the first-principles model is given in Table 4.1.

The following elementary assumptions are made:

- *Discretization of time.* Time is divided into large blocks of time that are further divided into equally sized slots. During each time slot, each link is either sends or doesn't.
- *Partial carrier sensing.* Due to carrier sensing, there exists a fixed probability c_{ij} that if link j is sending during a time slot, then the slot is not available for link i to send, when all other links are not sending.
- *Partial receiving interference.* There exists a fixed probability a_{ij} that if links i and j send during a time slot, then link i does not successfully receive the data, when all other links are not sending. Also, due to inherent loss, there exists a fixed probability

s_i	Sending rate of link i.
r_i	Receiving rate of link i.
d_i	Delivery ratio of link i.
a_{ij}	Receiving interference probability of link j interfering with link i.
c_{ij}	Carrier sensing probability of link j being sensed by link i.
S_i	Effective sending rate of all other links as observed by link i.
R_i	Effective (receiving) interference rate at link i due to all other links.
K_i	Set of links that contend with link i.
$\|p\|$	Number of elements in set p.
$p_1 \backslash p_2$	Set of elements in p_1 but not in p_2.
$\mathcal{P}(p)$	Set of all subsets of p except the empty set.
$\mathcal{P}_z(p)$	Set of all subsets of p with $\|p\| = z$.
$f_i(p)$	Sending rate product of links in p at link i.
$f'_i(p)$	Interference rate product of links in p at link i.
$g_i(p)$	Free space term of links in p at link i.
$h(p)$	Independence of links in p.
$\phi_i(p)$	Transparency of link i to links in p.

Table 4.1: Notation used in the first-principles model.

d_i that if link i sends data during a time slot, then it will be successfully received, when no other link is sending.

- *Uniform random selection.* For each time block T, each link has a set $F \subset T$ of available time slots in which to send, and a set $X \subset F$ when it does send. Each $t \in F$ has an equal probability of being in X.

- *Negligible indirect scheduling.* Using the notation from the above assumption, if link i carrier senses links j and k sending during $X_j, X_k \subset T$, respectively, then dependencies of $X_j \cup X_k$ on the sending times of any link $l \neq i, j, k$ is negligible.

When considering the effective rate of links j and k as perceived by link i, realistically there may be some other link l (or even a set of other links) that cause the rates of j and k to overlap more or less than usual, which could in turn affect how much link i can send. The last assumption simply states that these effects are negligible. We recognize that it could

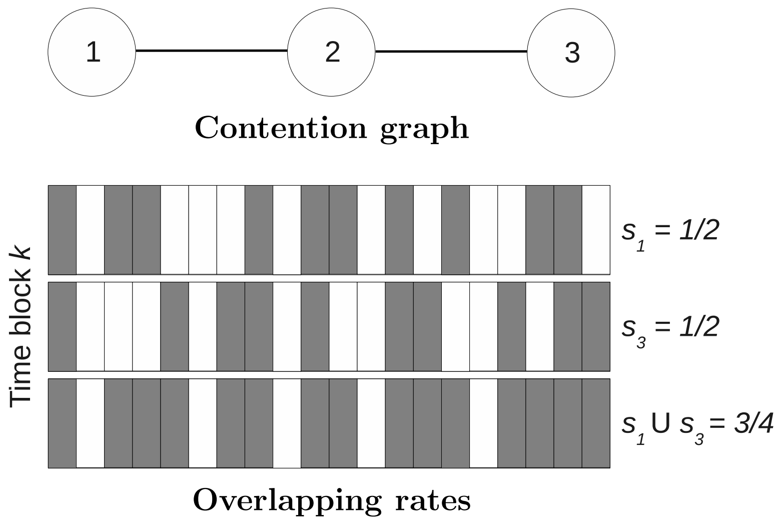


Figure 4.1: The contention graph of an example network, where one link in the middle contends with two outer links. If links 1 and 3 have rates of 1/2, and send at random times, then their effective rate or union will be 3/4 on average.

have some impact on the accuracy of the model, but fine tuning the model in this way will not be addressed in this thesis.

4.1 A Simple Example

Here we present a simple example to illustrate the design concepts behind the first-principles model. Consider the contention graph in Figure 4.1. Let us assume that at time k the rates are set at $s_1[k] = s_3[k] = 1/2$ and $s_2[k] = 0$. What is the *effective rate* S_2 of links 1 and 3, as measured by link 2? The maximal clique model (2.1) suggests that $S_2 = \max(s_1, s_3) = 1/2$. This implies that signals s_1 and s_3 overlap perfectly. If they didn't overlap at all, then $S_2 = s_1 + s_3 = 1$. In reality, there is some random overlap between the two. If we assume that the slots within the time block k during which 1 and 3 send are chosen at random, then, on average, half of the slots chosen by 1 will also be chosen by 3. Thus they overlap 1/4 of the total time, and the effective rate is $S_2 = 3/4$.

In general, the effective rate S_2 actually depends on how much link 2 is sending, since it limits the available space over which links 1 and 3 can choose random sending times. It should be easy to see that as s_2 increases, s_1 and s_3 will overlap more, which lowers S_2. The resulting formula is

$$S_2[k] = s_1[k] + s_3[k] - \frac{s_1[k]s_3[k]}{1 - s_2[k]}. \tag{4.1}$$

If we assume some kind of delay of link 2 affecting how much the others overlap, the dynamic system would behave according to

$$s_2[k + \tau] \leq 1 - S_2[k] \tag{4.2}$$

for some delay τ. The equilibrium constraint is

$$s_2 + S_2 \leq 1. \tag{4.3}$$

4.2 Union of Uniform Random Sets

A link's sending rate is restricted by how much it senses that others are sending, in other words, their effective rate. Our goal is to find a formula for this effective rate by calculating the union of each random set representing a link's sending times. We begin by defining a uniform random set:

Definition 1. *Let T and $F \subset T$ be finite sets. Also let $\xi_t : \Omega \to \{0, 1\}$ for all $t \in T$ be a collection of i.i.d. random variables on the probability space $(\Omega, \mathcal{A}, P)$. Then $X : \Omega \to 2^T$, where*

$$X(\omega) = \{t \in F : \xi_t(\omega) = 1\}, \tag{4.4}$$

is a uniform random set on F. Moreover, if F is a random set, then

$$X(\omega) = \{t \in F(\omega) : \xi_t(\omega) = 1\} \tag{4.5}$$

also defines a uniform random set on F. The set F is called the parent of X.

The deterministic set T can be considered as the entire time block. If link i sends at rate s_i, link j hears $c_{ji}s_i$ of T being occupied. Thus the $c_{ij}s_j$ that link i hears must be a random set chosen from a subset of T, namely during which j did not hear i. For this reason we include in the definition of a uniform random set a parent set, or free space F, to which it is restricted.

Now consider another link k adding to the effective rate that i senses. The effective rates of j and k by themselves overlap in the total effective rate a certain amount depending on how much they sense each other. In this sense, the above definition still does not fully explain the interaction of uniform random sets. We thus define the independence of uniform random sets:

Definition 2. *Let $\mathbb{X}$ be a collection of uniform random sets with parents $\mathbb{F}$, enumerated by $N = \{1, \ldots, n\}$. Then their independence is*

$$h(N) = E_{t\in\cap\mathbb{F}}\left[\frac{\Pr(t \in \cap\mathbb{X})}{\prod_{i\in N}\Pr(t \in X_i \cap \mathbb{F})}\right], \tag{4.6}$$

where E denotes the expected value.

Let $|\cdot|$ denote the expected size of a random set. Averaging over all t in $\cap\mathbb{F}$, we have from (4.6) that

$$\Pr(t \in \cap\mathbb{X}) = h(N)\prod_{i\in N}\Pr(t \in X_i \cap \mathbb{F}),$$
$$\frac{|\cap\mathbb{X}|}{|\cap\mathbb{F}|} = h(N)\prod_{i\in N}\frac{|X_i \cap \mathbb{F}|}{|\cap\mathbb{F}|}.$$

But $|X_i \cap \mathbb{F}|$ is found by multiplying $|X_i|$ by the probability that an element in F_i is also in $\cap\mathbb{F}$:

$$|X_i \cap \mathbb{F}| = \frac{|\cap\mathbb{F}|}{|F_i|}|X_i|$$

so that

$$|\cap \mathbb{X}| = h(N)|\cap \mathbb{F}| \prod_{i \in N} \frac{|X_i|}{|F_i|}. \tag{4.7}$$

We can now invoke the inclusion-exclusion principle

$$\left| \bigcup_{X \in \mathbb{X}} X \right| = \sum_{p \in \mathcal{P}(\mathbb{X})} (-1)^{|p|-1} \left| \bigcap_{X \in p} X \right| \tag{4.8}$$

to find the size of the union of many uniform random sets.

Theorem 1. *Let $\mathbb{X}$ be a collection of uniform random sets with parents $\mathbb{F}$, enumerated by $N = \{1, \ldots, n\}$. Then*

$$|\cup \mathbb{X}| = \sum_{p \in \mathcal{P}(N)} (-1)^{|p|-1} \left(\prod_{i \in p} |X_i| \right) \frac{\left| \bigcap_{i \in p} F_i \right|}{\prod_{i \in p} |F_i|} h(p). \tag{4.9}$$

Proof. The result follows from (4.7) and (4.8). $\square$

4.3 Constraints

The sending constraint is given by

$$s_i + S_i \leq 1, \ \forall i \in L, \tag{4.10}$$

where

$$S_i = \sum_{p \in \mathcal{P}(L_i)} (-1)^{|p|-1} f_i(p) g_i(p) h(p), \tag{4.11}$$

$$f_i(p) = \prod_{j \in p} c_{ij} s_j, \tag{4.12}$$

$$g_i(p) = \frac{\phi_i(p)}{\prod_{j \in p} \phi_i(j)}, \tag{4.13}$$

$$\phi_i(p) = 1 - s_i \sum_{p' \in \mathcal{P}(p)} (-1)^{|p'|-1} \prod_{j \in p'} c_{ji}, \tag{4.14}$$

and the independence is given by

$$h(p) = \prod_{\{i,j\} \in \mathcal{P}_2(p)} (1 - c_{ij} - c_{ji} + c_{ij}c_{ji}). \tag{4.15}$$

The receiving constraint is given by

$$r_i = d_i(1 - R_i)s_i, \tag{4.16}$$

where

$$R_i = \sum_{p \in \mathcal{P}(L_i)} (-1)^{|p|-1} f'_i(p)h(p) \tag{4.17}$$

and

$$f'_i(p) = \prod_{j \in p} a_{ij}s_j. \tag{4.18}$$

Note that (4.15) is an approximation of (4.6). If one link carrier senses another link completely ($c_{ij} = 1$) then their random sets do not intersect. If any two random sets in p do not intersect, then the intersection of p is empty, which means that $h(p)$ should equal zero. Only when all random sets are independent should it equal one.

The formulas for S_i and R_i follow immediately from Theorem 1. In the case of R_i, since there is no intermediary correlating link for the interferers, F is the entire space with size 1. However, for S_i, we need to derive $g_i(p)$, which corresponds to the term of free spaces F in the theorem.

Link j observes a free space F_j of size $1 - c_{ji}s_i$, in which link i observes an occupied space X_j of size $c_{ij}s_j$. The free space F_j consists of a portion during which link i is not sending, denoted Γ, and a portion during which link i is sending but not heard by j, denoted

Ψ_j. Their sizes are given by

$$|\Gamma| = 1 - s_i$$

and

$$|\Psi_j| = (1 - c_{ji})s_i.$$

Let $F(p) = \cap_{j \in p} F_j$ and $\Psi(p) = \cap_{j \in p} \Psi_j$. Then, since every Ψ_j is an independent uniform random set within a common space of size s_i, we have by way of (4.7),

$$\begin{aligned} |\Psi(p)| &= \frac{\prod_{j \in p} |\Psi_j|}{s_i^{|p|-1}} \\ &= s_i \prod_{j \in p} (1 - c_{ji}). \end{aligned}$$

Thus

$$\begin{aligned} \phi_i(p) := |F(p)| = |\Gamma| + |\Psi(p)| \\ &= 1 - s_i + s_i \prod_{j \in p} (1 - c_{ji}) \\ &= 1 - s_i \sum_{p' \in \mathcal{P}(p)} (-1)^{|p'|-1} \prod_{j \in p'} c_{ji}. \end{aligned}$$

We call this function the transparency of i to p, and use it to simplify the notation in $g_i(p)$.

As a side note, it is straightforward to incorporate flow constraints into the model as well, by introducing mappings $t(\cdot)$ from the hop number in the flow to the index of the link considered. Then

$$s_{t(m)} \leq r_{t(m-1)} \tag{4.19}$$

for each hop m and each flow mapping t ensures that no subsequent hop sends more than it receives.

4.4 Objective

The objective function for the first-principles model is again the sum of the log of receiving rates, as in (3.3) for the partial interference model.

Chapter 5

Reduction of the First-Principles Model to the Classical Models

We must prove that the first-principles model reduces to the classical models. This is necessary to guarantee that the more general first-principles model can yield the exact same constraints as the classical models when there is no partial carrier sensing or interference, and thus perform at least as well as these models.

5.1 Reduction to the Maximal Clique Model

The first-principles model reduces to the maximal clique model when carrier sensing is binary and symmetric. In other words, $c_{ij} = c_{ji} \in \{0, 1\}$. We prove this by showing that the set of feasible sending rates in one model is equivalent to the feasible set in the other model.

We first formally define the two sets of feasible rates based on each model:

Definition 3. *Given a contention graph (L, C), the set $\mathcal{S}_1$ consists of all vectors of sending rates s that satisfy* (2.1).

Definition 4. *Given a contention graph (L, C), the set $\mathcal{S}_2$ consists of all vectors of sending rates s that satisfy* (4.10)*, where*

$$S_i = \sum_{p \in \mathcal{P}(K_i)} \left(\frac{-1}{1 - s_i} \right)^{|p|-1} h(p) \prod_{j \in p} s_j$$

and

$$h(p) = \begin{cases} 0, & \exists i, j \in p : i \in K_j, \\ 1, & \text{otherwise.} \end{cases}$$

Theorem 2. *Given a contention graph (L, C), $\mathcal{S}_1 = \mathcal{S}_2$.*

Proof. First, note that $\mathcal{S}_1$ is equivalent to the set of s satisfying

$$s_i + \max\{S_i(j) : j \in C(i)\} \leq 1, \ \forall i \in L,$$

where $S_i(j) = \sum_{l \in L(j) \setminus i} s_l$. To show that $\mathcal{S}_1 = \mathcal{S}_2$, it suffices to show that, for any $i \in L$, the constraint boundary is equivalent in $\mathcal{S}_1$ and $\mathcal{S}_2$. Thus, letting

$$s_i = 1 - \max\{S_i(j) : j \in C(i)\},$$

we seek to show that $S_i = \max\{S_i(j) : j \in C(i)\}$.

Without loss of generality, let $i = 0$ and $K_0 = \{1, \ldots, n\}$, where $L(1) = \{1, \ldots, m\}$ is the most constraining maximal clique and $S_0(1) = \sum_{j=1}^{m} s_j$. Then

$$\begin{aligned} S_0 &= \sum_{p \in \mathcal{P}(K_0)} \left(\frac{-1}{S_0(1)}\right)^{|p|-1} h(p) \prod_{j \in p} s_j \\ &= \sum_{z=1}^{n} \left(\sum_{p \in \mathcal{P}_z(K_0)} \left(\frac{-1}{S_0(1)}\right)^{z-1} h(p) \prod_{j \in p} s_j \right) \\ &= \sum_{j=1}^{n} s_j + \sum_{z=2}^{n} \left(\frac{-1}{S_0(1)}\right)^{z-1} \left(\sum_{p \in \mathcal{P}_z(K_0)} h(p) \prod_{j \in p} s_j \right) \\ &= S_0(1) + \sum_{j=m+1}^{n} s_j + \sum_{z=2}^{n} \left(\frac{-1}{S_0(1)}\right)^{z-1} \left(\sum_{p \in \mathcal{P}_z(K_0)} h(p) \prod_{j \in p} s_j \right). \end{aligned}$$

We therefore must determine that

$$\sum_{j=m+1}^{n} s_j + \sum_{z=2}^{n} \left(\frac{-1}{S_0(1)}\right)^{z-1} \left(\sum_{p \in \mathcal{P}_z(K_0)} h(p) \prod_{j \in p} s_j \right) = 0. \tag{5.1}$$

Note that $h(p) = 0$ for any p that has at least two elements from $L(1)\backslash\{0\}$, so that

$$\sum_{p\in\mathcal{P}_z(K_0)} h(p)\prod_{j\in p} s_j = S_0(1) \sum_{p\in\mathcal{P}_{z-1}(K_0\backslash L(1))} h(p)\prod_{j\in p} s_j + \sum_{p\in\mathcal{P}_z(K_0\backslash L(1))} h(p)\prod_{j\in p} s_j.$$

This is substituted into the argument of the second sum of (5.1) to obtain

$$\begin{aligned}\left(\frac{-1}{S_0(1)}\right)^{z-1} &\sum_{p\in\mathcal{P}_z(K_0)} h(p)\prod_{j\in p} s_j = \\ &- \frac{(-1)^{z-2}}{S_0(1)^{z-2}} \sum_{p\in\mathcal{P}_{z-1}(K_0\backslash L(1))} h(p)\prod_{j\in p} s_j \;+\; \frac{(-1)^{z-1}}{S_0(1)^{z-1}} \sum_{p\in\mathcal{P}_z(K_0\backslash L(1))} h(p)\prod_{j\in p} s_j.\end{aligned}$$

The second term above for some z cancels with the first term in the corresponding $z+1$ equation, and the first term for $z = 2$ cancels with $\sum_{j=m+1}^{n} s_j$. We need only check that the second term for $z = n$ goes to zero. By inspection,

$$\sum_{p\in\mathcal{P}_n(K_0\backslash L(1))} h(p)\prod_{j\in p} s_j = 0,$$

because $|K_0\backslash L(1)| < n$. Thus, $S_0 = S_0(1)$ and $\mathcal{S}_1 = \mathcal{S}_2$. □

5.2 Reduction to the Partial Interference Model

We now show that when interferers of link i do not contend with each other, (4.16) and (4.17) from the first-principles model reduce to (3.1) in the partial interference model. To do this, we present a simple arithmetical theorem:

Theorem 3. *For some set L of indices,*

$$\prod_{j\in L}(1 - x_j) = 1 - \sum_{p\in\mathcal{P}(L)} (-1)^{|p|-1}\prod_{j\in p} x_j. \tag{5.2}$$

Proof. Without loss of generality, let $L = \{1, \ldots, n\}$. We prove by induction. The base case $n = 1$ holds trivially. Assuming (5.2) holds for n, we need to show that it holds for $n+1$.

Defining $N = L\backslash(n+1)$,

$$\begin{aligned}
\prod_{j\in L}(1-x_j) &= (1-x_{n+1})\prod_{j\in N}(1-x_j) \\
&= (1-x_{n+1})\left(1-\sum_{p\in\mathcal{P}(N)}(-1)^{|p|-1}\prod_{j\in p}x_j\right) \\
&= 1-\sum_{p\in\mathcal{P}(N)}(-1)^{|p|-1}\prod_{j\in p}x_j - x_{n+1} + \sum_{p\in\mathcal{P}(N)}(-1)^{|p|-1}x_{n+1}\prod_{j\in p}x_j \\
&= 1-\sum_{p\in\mathcal{P}(L)}(-1)^{|p|-1}\prod_{j\in p}x_j. \qquad \square
\end{aligned}$$

Applying this result to (3.1) is straightforward, replacing x_j with $a_{ij}s_j$. Since under the limiting condition of no contending interferers we have $h(p) = 1$, we see that (4.16) does indeed reduce to (3.1).

Chapter 6

Solution to the First-Principles Optimization Problem

Again assuming that d is always unity, the first-principles NUM problem is

$$\begin{aligned} \mathbf{P}: \quad & \text{maximize} && \sum_{i \in L} \ln r_i \\ & \text{subject to} && r_i = (1 - R_i)s_i, \quad \forall i \in L, \\ & && s_i + S_i \leq 1, \qquad \forall i \in L. \end{aligned} \tag{6.1}$$

Instances of this problem are frequently non-convex, due to the addition and subtraction of several rational functions in S_i, and due to similar reasons in R_i. A branch and bound solution solves the problem by successively dividing the hypercube in which the feasible set resides into smaller regions, and evaluating lower and upper bound functions for the optimal value in each region. The bounds on each region allow one to conclude that some regions need not be divided further. A good tutorial on branch and bound appears in [25]. Figure 6.1 shows an example decision tree for the branch and bound method.

To implement branch and bound, we need only develop efficient upper and lower bound functions for each sub-problem of $\mathbf{P}$. Let $\mathbf{P}_k$ be the k-th sub-problem of $\mathbf{P}$ in the algorithm. A standard interior point solver Φ operating on $\mathbf{P}_k$ is sufficient to get a lower bound and the corresponding feasible point. To get an upper bound, we need to formulate a new, convex problem $\mathbf{P}'$ such that its solution is an upper bound to the solution of $\mathbf{P}$.

To begin, we wish to replace the constraint on s_i in (6.1) with something that is convex and will enclose the old constraints. This is done by replacing the effective rate S_i with something smaller, since this will leave more room for s_i to increase. Because S_i is the

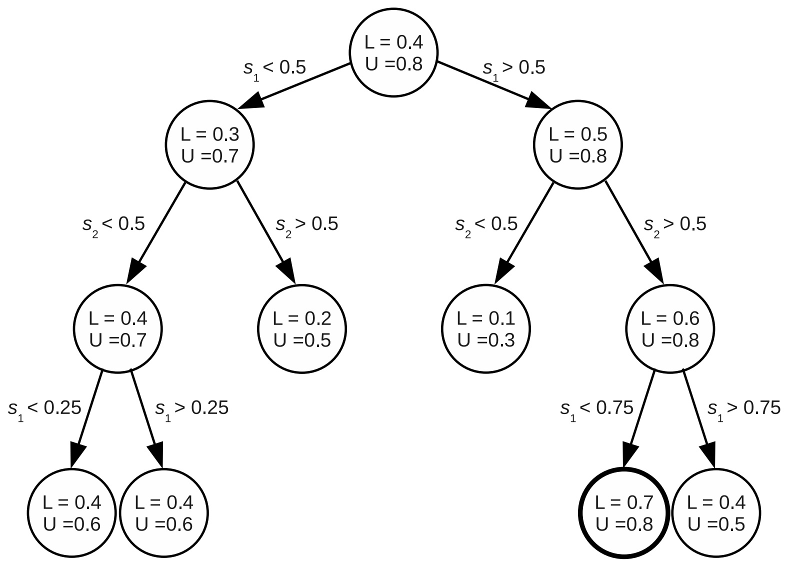


Figure 6.1: An example decision tree for the branch and bound method. Each node adds its extra constraint to the parent node's problem. Then lower and upper bounds are calculated. If the upper bound of a node is lower than the greatest lower bound, it is pruned and the subtree does not need to be considered. The bold node is the only one at this point that must continue to be partitioned, since all others are pruned.

size of the union of several sets with sizes $c_{ij}s_j$, it follows that $S_i \geq \max c_{ij}s_j$. Finally the new constraint

$$s_i + \max_{j \in L_i} c_{ij}s_j \leq 1$$

is equivalent to the family of constraints

$$s_i + c_{ij}s_j \leq 1, \quad \forall j \in L_i,$$

which are all linear.

Now we must modify the constraint on r_i in (6.1). First, note that replacing it with

$$r_i \leq (1 - R_i)s_i$$

does not change the solution to the problem, since choosing r that does not achieve equality produces a lower score, and increasing r has no effect on s. We next introduce the variable

$$y_i = r_i / s_i$$

so that the inequality becomes

$$y_i + R_i \leq 1.$$

Then, R_i is replaced in the same manner that S_i was replaced, which yields the family of constraints

$$y_i + a_{ij} s_j \leq 1, \quad \forall j \in L_i.$$

The change of variables from (s, r) to (s, y) also modifies the appearance of the objective function to

$$\sum_{i \in L} (\ln s_i + \ln y_i) .$$

The new, convex problem that bounds $\mathbf{P}$ is

$$\begin{aligned} \mathbf{P}' : \quad & \text{maximize} && \textstyle\sum_{i \in L} (\ln s_i + \ln y_i) \\ & \text{subject to} && y_i + a_{ij} s_j \leq 1, && \forall i, j \in L,\ i \neq j, \\ & && s_i + c_{ij} s_j \leq 1, && \forall i, j \in L,\ i \neq j. \end{aligned} \tag{6.2}$$

Let $\mathbf{P}'_k$ be the k-th sub-problem of $\mathbf{P}'$ in the algorithm. Then Φ operating on $\mathbf{P}'_k$ is sufficient to get an upper bound. Thus the branch and bound method, supplemented with the bound functions of $\Phi(\mathbf{P}_k)$ and $\Phi(\mathbf{P}'_k)$, solves $\mathbf{P}$ with efficient computation at each step.

6.1 Tightening the Upper Bound

The number of nodes that needs to be traversed is limited by pruning those that have upper bounds lower than the greatest lower bound. Hopefully, the relaxations introduced in $\mathbf{P}'$ are sufficiently tight that pruning will occur often. Although branch and bound with the upper bound defined thus far is guaranteed to eventually converge, it will still take an exponential number of iterations, and the pruning may not occur frequently enough to converge in a reasonable amount of time.

As an example, we implemented the algorithm in MATLAB and ran it on a somewhat dense topology with 8 links, using a standard PC. We reported the geometric mean of the receiving rates (3.12) as our performance. After the first iteration, we obtained a lower bound of 0.13 and an upper bound of 0.41. After approximately 60,000 iterations and 16 hours, the lower bound was still 0.13, and the upper bound was lowered to 0.21. Several other runs on different topologies also yielded similar results of a long convergence time, a large gap between the upper and lower bounds, and a lower bound that never moved.

These results revealed two important things about our branch and bound solution. First, the upper bound was searching over a feasible set much larger than that of the original problem. Tightening the upper bound could give much smaller gap at the first iteration, and result in more frequent pruning. Second, using a standard interior point method on the original non-convex problem might often find the optimal point or a near-optimal point. Although it is not a proof, the fact that the lower bound is not moving is a good indication that the non-convex problem is well behaved enough that we can frequently trust the initial lower bound as the optimal solution.

In order to speed the branch and bound algorithm, and to improve our confidence in our solution by tightening the gap between the upper and lower bounds, we now explore how to improve the constraints in $\mathbf{P}'$. First, note that although it is common for real wireless networks to have cases of partial carrier sensing, most networks will still have a great amount of binary carrier sensing, especially because many links will share the same sending node. Thus it is common to still find maximal cliques with $c = 1$ in the contention graph. The effective rate S_i is larger than the effective rate of any one maximal clique in L_i, and the effective rate of a maximal clique is simply the sum of their perceived rates. Consequently, we can replace the constraint on s_i in $\mathbf{P}'$ with

$$s_i + \sum_{j \in L_i(k)} c_{ij} s_j \leq 1, \quad \forall k \in C, \tag{6.3}$$

where C is the set of strict ($c = 1$) maximal cliques and $L_i(k)$ is the set of all links in clique k except i.

The same can be done with the constraint on y_i in $\mathbf{P}'$. However, even more can be done. The effective interference R_i is larger than the effective interference of any one group of independent maximal cliques. A group v of independent maximal cliques is defined such that, for any two distinct cliques $k, l \in v$, and for any links $i \in L(k)$ and $j \in L(l)$, $c_{ij} = c_{ji} = 0$. Let V be the set of all groups of independent maximal cliques, and let $R_i(v)$ be the effective interference of independent cliques in v on link i. From Theorem 3, we know that

$$1 - R_i(v) = \prod_{k \in v} \left(1 - \sum_{j \in L_i(k)} a_{ij} s_j \right).$$

We then have

$$\begin{aligned} \ln y_i &\leq \ln(1 - R_i) \\ &\leq \ln(1 - R_i(v)), \quad \forall v \in V \\ &= \ln \prod_{k \in v} \left(1 - \sum_{j \in L_i(k)} a_{ij} s_j \right) \\ &= \sum_{k \in v} \ln \left(1 - \sum_{j \in L_i(k)} a_{ij} s_j \right), \end{aligned}$$

so that the constraint

$$\sigma_i - \sum_{k \in v} \ln \left(1 - \sum_{j \in L_i(k)} a_{ij} s_j \right) \leq 0, \quad \forall v \in V \tag{6.4}$$

is convex, where $\sigma_i = \ln y_i$.

The modified problem for the upper bound is then

$$\begin{aligned}
\mathbf{P}': \quad & \text{maximize} && \sum_{i \in L} (\ln s_i + \sigma_i) && \\
& \text{subject to} && \sigma_i - \sum_{k \in v} \ln\left(1 - \sum_{j \in L_i(k)} a_{ij} s_j\right) \leq 0, && \forall i \in L,\ v \in V, \\
& && s_i + \sum_{j \in L_i(k)} c_{ij} s_j \leq 1, && \forall i \in L,\ k \in C.
\end{aligned} \tag{6.5}$$

We implemented this algorithm in MATLAB and ran it on the same topology of 8 links. After the first iteration, we obtained a lower bound of 0.13 and an upper bound of 0.18. After approximately 6,000 iterations, the lower bound was still 0.13, and the upper bound was lowered to 0.17. Although we still suspect the convergence time to be largely affected by the exponential nature of the search, the initial upper bound achieved is much better than the original algorithm's final bound after 60,000 iterations.

6.2 Implementation

In this section, we describe how the branch and bound solution was implemented in Python and MATLAB. This should not be confused with an implementation of a controller to be deployed in a real wireless network. Such a controller would require a distributed solution with fast convergence, which we do not have. The major code fragments appear in Appendix A.

The file `solve.py` is the master program that formulates and solves the problem. It takes as input a directory in which it expects to find a text file named `a` (no extension) which contains the interference factors and another named `c` which contains the contention coefficients. The format of each file is whitespace delimited rows, having the appearance of matrix form. The program produces as output a file in the same directory named `matlab.txt`. Below is an example `matlab.txt` file:

```
First-principles:
s = 0.210103 0.140290 0.136273 0.130772 0.132108 0.136342 0.140366 0.215762
r = 0.156821 0.133438 0.109429 0.114612 0.110562 0.109800 0.133591 0.155834
```

```
score = 0.126681
bound = 0.180474
difference = 0.053793
threshold = 0.010000
certainty = 0.003897
radius of uncertainty = 0.154620
active regions = 1596
initial s = 0.210103 0.140290 0.136273 0.130772 0.132108 0.136342 0.140366 0.215762
initial r = 0.210103 0.140290 0.136273 0.130772 0.132108 0.136342 0.140366 0.215762
initial score = 0.126681
difference b/t initial and best score = 0.000000
initial bound = 0.182169
difference b/t initial and best bound = 0.001695
distance b/t initial and best s = 0.000000
iterations = 10000
time = 00 : 50 : 41.8
time in secs = 3041.779740
exit status = -1

Partial interference:
predicted s = 0.177833 0.137870 0.134096 0.122698 0.122698 0.133114 0.136833 0.183827
true      s = 0.177833 0.137870 0.134096 0.122698 0.122698 0.133114 0.136833 0.183827
predicted r = 0.139483 0.131733 0.111183 0.109780 0.105590 0.110560 0.130795 0.140705
true      r = 0.135809 0.131666 0.109756 0.109780 0.105590 0.109165 0.130730 0.136132
predicted score = 0.121720
true score = 0.120413
difference from optimal = 0.006268
distance b/t predicted s and optimal s = 0.016765
infeasibility = 0.000000

Maximal clique:
predicted s = 0.216174 0.131616 0.131616 0.110874 0.110874 0.131616 0.131616 0.216174
true      s = 0.216174 0.131616 0.131616 0.110874 0.110874 0.131616 0.131616 0.216174
predicted r = 0.216174 0.131616 0.131616 0.110874 0.110874 0.131616 0.131616 0.216174
true      r = 0.165112 0.125306 0.106377 0.096907 0.092528 0.106377 0.125306 0.160418
predicted score = 0.142745
true score = 0.119719
difference from optimal = 0.006962
distance b/t predicted s and optimal s = 0.011617
infeasibility = 0.000000
```

The vectors `s` and `r` are the sending and receiving rates. The `score` and `bound` are the lower and upper bounds, respectively, to the solution to the optimization problem. The `threshold` is the cutoff at which the branch and bound method stops iterating, that is, when `bound` $-$ `score` $<$ `threshold`. The `certainty` is the normalized volume of the hypercube that has been pruned. The `radius of uncertainty` is the distance between the best `s` found and furthest `s` reported by the upper bound function among the still active regions. This distance, as well as the distance between the best `s` and the initial `s`, is normalized by the length of the diagonal of the hypercube. The `exit status` is `1` for normal, `2` for when the maximum number of iterations is reached, and `-1` for when an error occurred (such as a particular run of the interior point method not finding a feasible point).

The sections in the output on the classical controllers report both a `predicted` and a `true` value for `s`, `r`, and `score`. The predicted values are those reported by the classical models, and the true values are estimates of what they would actually obtain if the `predicted s` was infeasible. This was estimated by finding the point along the line segment from the origin to the `predicted s` at which it crosses the boundary of the feasible set. Then this `true s` was used in (4.16) to obtain `true r`, and consequently `true score`. The `infeasibility` is the fraction of the previously mentioned line segment that lies beyond the boundary of the feasible set.

The `solve.py` program creates the necessary constraint functions for MATLAB's `fmincon` optimization tool, and then opens a MATLAB session to run the `solve.m` script. This MATLAB script contains the branch and bound algorithm, and also calculates the information for the classical controllers. A granularity of 1/16 in each dimension of the vector space was used in the branch and bound algorithm as a cutoff to decide not to search any deeper in the decision tree. Also, if the upper corner of a region was feasible, its score was set as both the upper and lower bound. The upper and lower bounds were recorded up to 6 decimal places.

Chapter 7

Performance of the Classical Controllers: Numerical Results

The performance of the classical controllers was compared numerically against the performance predicted by the first-principles NUM problem for several kinds of topologies. To do so, we define the optimality of a classical controller as

$$O = P'/P^*, \tag{7.1}$$

where P' is the performance (geometric mean of receiving rates) of the classical controller and P^* is the performance (lower bound) reported by the first-principles NUM problem. Note that O is simply the inverse of the performance ratio R defined in (3.13). Thus an optimality of 1 equates to no performance loss despite the inaccuracies in the model with respect to partial carrier sensing or partial interference.

For the smaller topologies, the branch and bound algorithm converged to within a difference of 0.01 performance. However, for some of the larger topologies, the algorithm took too long to converge. We therefore will also report a certainty measure in these cases according to (7.1), where P' is the branch and bound's lower bound score and P^* is the upper bound score. This measure tells us how much higher the true optimal score might be, and thus how much worse the optimality of the classical controllers might be. If the certainty is not reported for a particular topology, then it was very close to 1.

The binary contention graphs for the classical models were built based on the independence approximation (4.15). If two links had an independence greater than 0.5, an edge was drawn between them. For the maximal clique model, a contention edge also needed

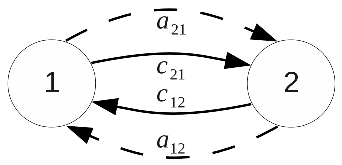


Figure 7.1: The partial contention/interference graph of the two-link topologies.

to be drawn for high interference values, as we found in §3.4. This was done by defining a function mimicking the independence, but using a instead of c. An edge was drawn on the maximal clique model's contention graph if the multiplication of the two "independence" functions (a and c) was greater than 0.5.

When computing results over a range of interference factor values a, we omitted any topologies such that there existed two links i and j where

$$a_{ij} > 1 - c_{ij}. \tag{7.2}$$

This is because such topologies cannot occur due to the relationship between a and c. If two links carrier sense each other well, their sending rates cannot overlap much and therefore they cannot interfere with each other much.

7.1 Two-Link Topologies

The optimality and infeasibility were measured on the simplest topologies of interest, with only two links. The partial contention/interference graph of this class of topologies is given in Figure 7.1. In this figure and subsequent figures, we let solid arrows denote contention (where the arrow points to the link that is sensing) and dashed arrows denote interference (where the arrow points to the link being interfered with). The values of a_{12}, a_{21}, c_{12}, and c_{21} were permutated over the range of 0 to 1, with granularity of 0.2, omitting the topologies that were unrealistic according to (7.2). The CDFs of the optimality for both models appear in Figure 7.2, and the CDFs of the infeasibility appear in Figure 7.3.

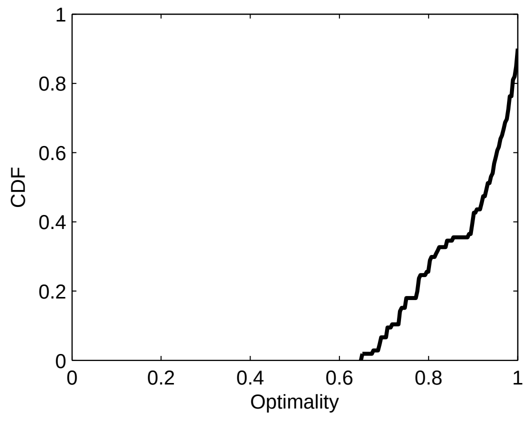
The maximal clique controller performs above 0.9 optimality in most cases, but many cases drop well below that, with the worst performing at 0.648 when $c_{12} = c_{21} = a_{12} = 0$ and $a_{21} = 0.6$. The partial interference controller, on the other hand, almost always performs above 0.9 optimality, with the worst performing at 0.776 when $c_{12} = 0.4$, $c_{21} = 0.6$, and $a_{12} = a_{21} = 0$.

In Figure 7.3, infeasibilities of zero mean that the controllers chose feasible rates. The CDFs reveal that, not only do the classical controllers dictate rates that are infeasible approximately half the time, but the infeasible rates are often a non-negligible distance from the feasible boundary. It is possible that a controller designed to choose rates that are often physically impossible to achieve will behave unpredictably when deployed on a real network.

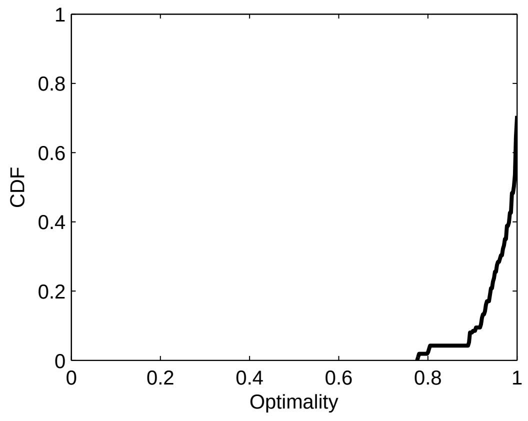
The partial interference controller performs well in most cases despite the fact that it frequently predicts infeasible rates, since as it attempts to reach those rates, it conveniently saturates at a near-optimal value. Of course, this is assuming that when the controller is implemented on a real network and the prescribed rates are infeasible, the controller will saturate at the predicted point. It is quite possible, especially given different initial rates other than zero, that the controller traverses unfair rates as it moves toward the new prescribed rates, and thus saturates at extremely unfair rates. These results show that, giving the controller implementation the benefit of the doubt, it could perform relatively well in most scenarios.

7.2 Three-Link Topologies

We tested various three-link topologies by choosing the two-link topology with the worst optimality for the partial interference controller and adding a link to it. We first tested over all interference factors a both from and to the third link, with a granularity of 0.2, and c values involving link 3 set to zero. The partial contention/interference graph of this class of topologies is given in Figure 7.4. Figure 7.5 shows the CDFs of optimality for these topologies. For the maximal clique controller, optimality is worst at 0.56 when $a_{13} = 0$,

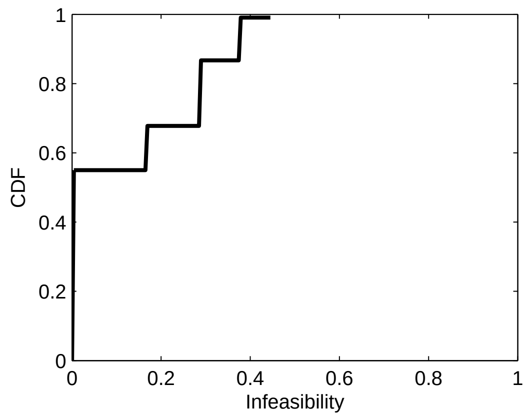


(a) Maximal clique controller

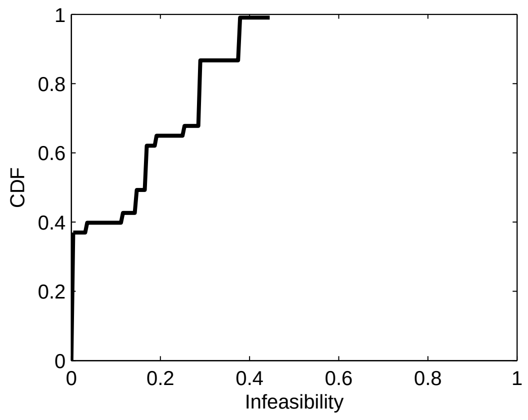


(b) Partial interference controller

Figure 7.2: CDFs of the optimality of the classical controllers for the two-link topologies.



(a) Maximal clique controller



(b) Partial interference controller

Figure 7.3: CDFs of the infeasibility of the classical controllers for the two-link topologies.

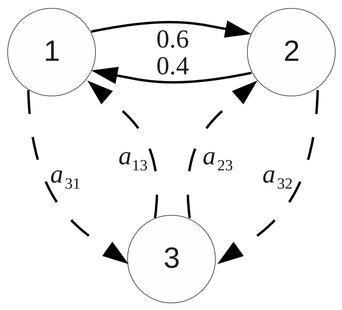
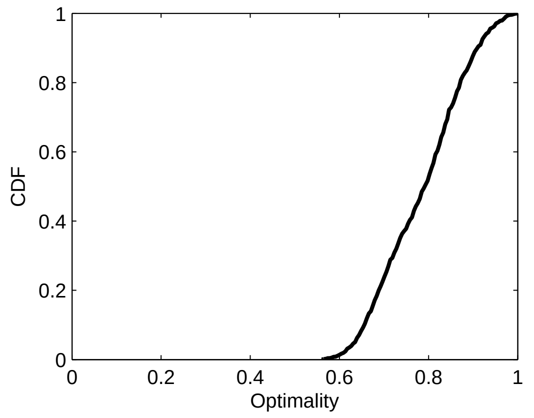
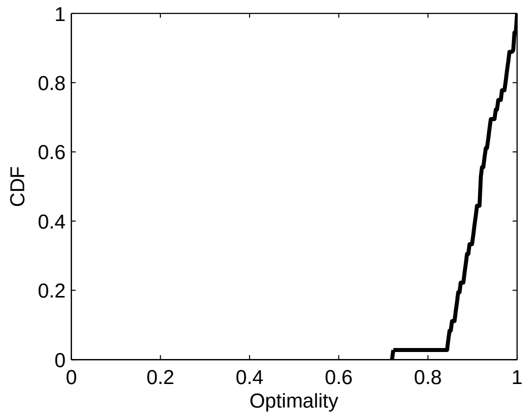


Figure 7.4: The partial contention/interference graph of the three-link topologies varied over interference.



(a) Maximal clique controller



(b) Partial interference controller

Figure 7.5: CDFs of the optimality of the classical controllers for the three-link topologies varied over interference factors a involving link 3, building off of the partial interference controller's worst case two-link topology.

$a_{23} = 0.4$, $a_{31} = 0.6$, and $a_{32} = 0.2$. However, several other topologies scored nearly as low, with a seemingly arbitrary mix of interference factors. Optimality is close to 1 largely for the cases where there is high interference ($a \geq 0.8$) for most or all parameters.

For the partial interference controller, 95% of all cases are at 0.85 optimality or better. Optimality is worst at 0.72 when $a_{13} = a_{23} = 0$ and $a_{31} = a_{32} = 1$, in other words, when links 1 and 2 (which are dependent) send near full capacity and interfere completely with link 3. There is a strict ordering on the optimality based solely on the values of a_{31} and a_{32}, or how much the two dependent links interfere with link 3. Surprisingly, this ordering is not strictly decreasing values of a. For example, the highest optimalities at 0.99 and above are achieved when a_{31} and a_{32} are equal to 0.6 and 0.8 (either order). It is difficult to ascertain why this happens because the optimality also takes into account how well the controller makes up for the partial carrier sensing between links 1 and 2.

We also tested over all contention coefficients c both from and to the third link, with a granularity of 0.2, and a values involving link 3 set to zero. The partial contention/interference graph of this class of topologies is given in Figure 7.6. Figure 7.7 shows the CDFs of optimality for these topologies. The two CDFs are identical because without any interference,

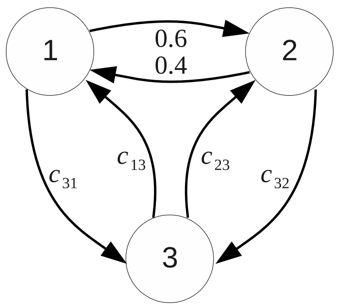
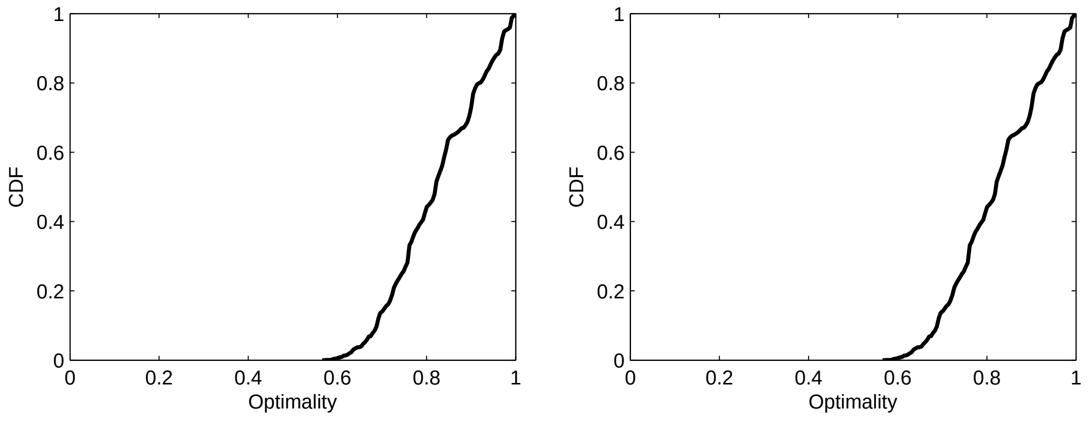


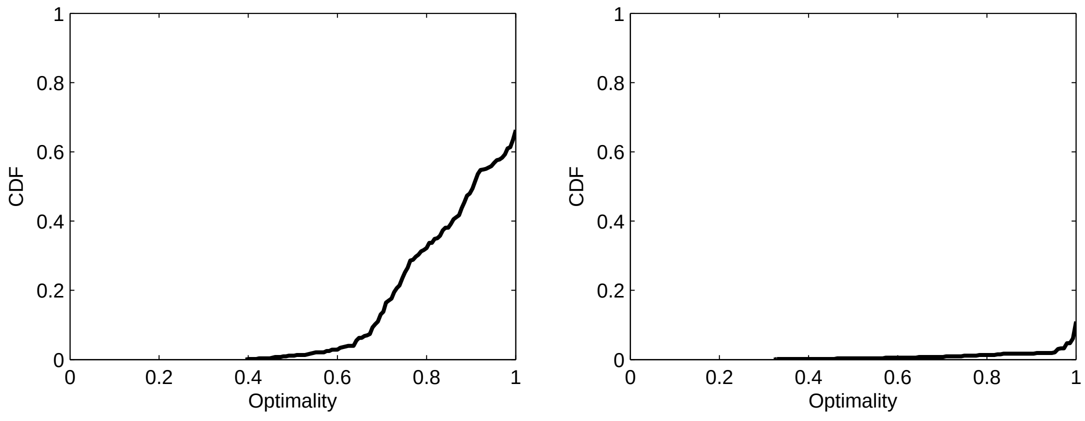
Figure 7.6: The partial contention/interference graph of the three-link topologies varied over contention.



(a) Maximal clique controller (b) Partial interference controller

Figure 7.7: CDFs of the optimality of the classical controllers for the three-link topologies varied over contention coefficients c involving link 3, building off of the partial interference controller's worst case two-link topology.

the two classical models are identical. Optimality is worst at 0.57 when $c_{13} = 0.2$, $c_{23} = 0.4$, $c_{31} = 0.4$, and $c_{32} = 0.2$. In general, the classical controllers lose significant performance when there is a significant amount of partial carrier sensing in the network (c values not close to 0 or 1). However, we have seen in the above results of varying the a parameters that introducing partial interference into a topology plagued with partial carrier sensing usually results in much better optimality for the partial interference controller.



(a) Maximal clique controller (b) Partial interference controller

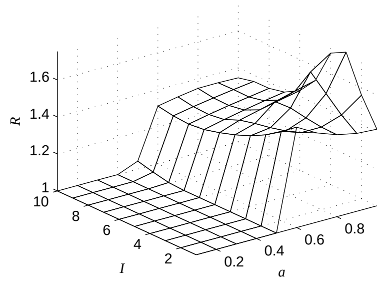
Figure 7.8: CDFs of the optimality of the classical controllers for the partial interference topologies.

7.3 Partial Interference Topologies

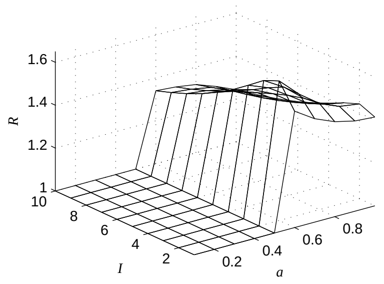
The optimality was measured on the same topologies presented in §3.4, to explore the effect of partial interference. Since the first-principles model differs from the partial interference model when the interferers contend, the results both for independent interferers and for fully contending interferers were obtained. The CDFs of the optimality for both classical models appear in Figure 7.8. These CDFs make it readily apparent that the maximal clique controller again has low optimality in very many scenarios, and that the partial interference controller almost always performs at an optimality very close to 1. We also see, however, that in the very few scenarios it performs poorly, the optimality can be extremely low, as low as 0.33. We will explore these particular scenarios in more detail.

We also plot the performance ratio R (the inverse of the optimality) using a 3-D graph for each topology, similar to Figures 3.2, 3.3, and 3.4. Figure 7.9 shows the results for the maximal clique model. In many scenarios, the maximal clique model behaves very poorly, as was already confirmed in §3.4.

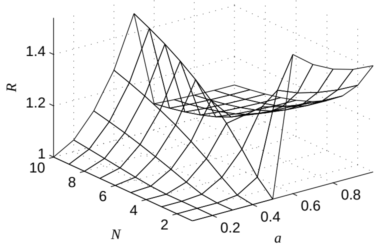
Figure 7.10 shows the results for the partial interference model. The ratio only increases non-negligibly in the case of multiple dependent interferers with very high interference



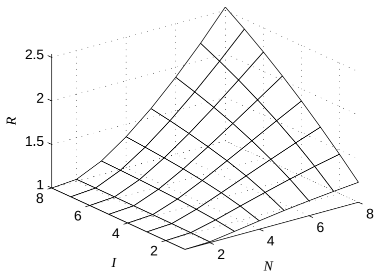
(a) I independent links interfering with a single link.



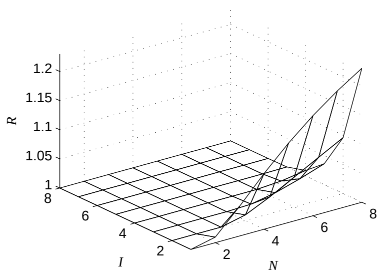
(b) I dependent links interfering with a single link.



(c) N contenders in a clique with one interferer.



(d) N contenders in a clique with I independent interferers, $a = 0.4$.



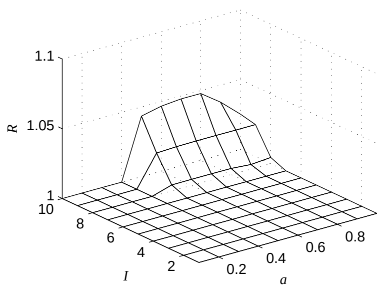
(e) N contenders in a clique with I dependent interferers, $a = 0.4$.

Figure 7.9: Numerical results for the performance ratio of the first-principles model to the maximal clique controller in scenarios with partial interference.

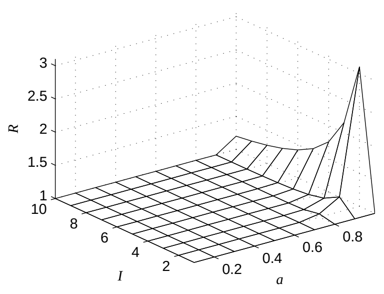
factors a, as seen in Figure 7.10(b). This is because, when there is only binary carrier sensing, the partial interference model only differs from the first-principles model when multiple interferers are contending. When this occurs, the inaccuracy of the partial interference model is significantly mitigated by smaller values of a. Note, however, that when this less common scenario occurs R can be as high as 3, meaning an optimality as low as 0.33.

7.4 Partially Dependent Interferers

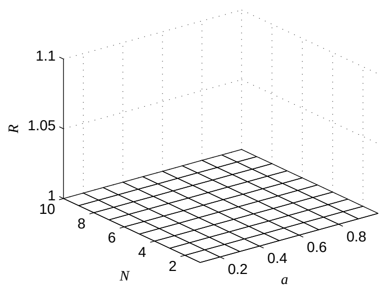
Another set of topologies we tested over was with I interferers on one link, with $a = 1$ and $c = 0.5$ between the interferers. Figure 7.11 shows the partial contention/interference graph for this class of topologies. Figure 7.12 plots the optimality of the classical models across



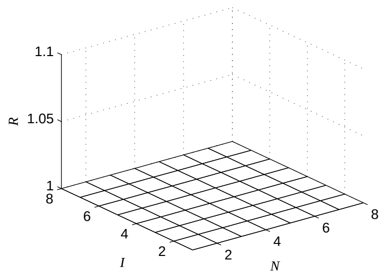
(a) I independent links interfering with a single link.



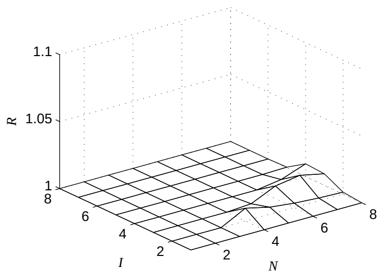
(b) I dependent links interfering with a single link.



(c) N contenders in a clique with one interferer.



(d) N contenders in a clique with I independent interferers, $a = 0.4$.



(e) N contenders in a clique with I dependent interferers, $a = 0.4$.

Figure 7.10: Numerical results for the performance ratio of the first-principles model to the partial interference controller in scenarios with partial interference.

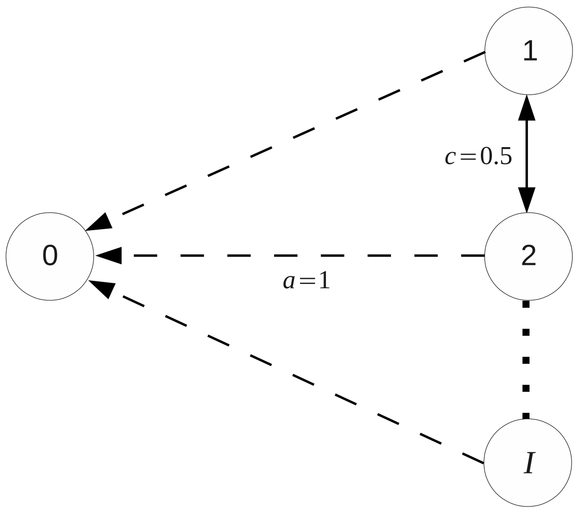


Figure 7.11: The partial contention/interference graph of the topologies with partially dependent interferers. Contention between the I interferers is $c = 0.5$ and interference on link 0 is $a = 1$.

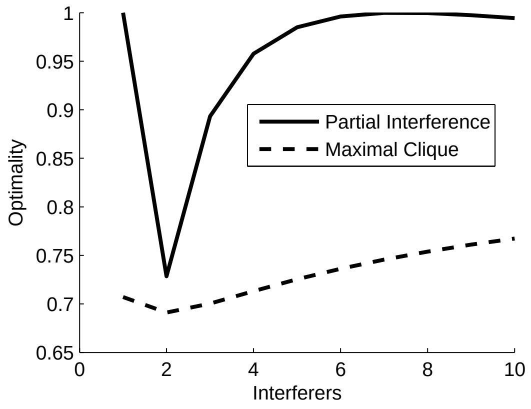


Figure 7.12: Optimality of classical models for partially dependent interferers on one link with $a = 1$, where $c = 0.5$ among interferers.

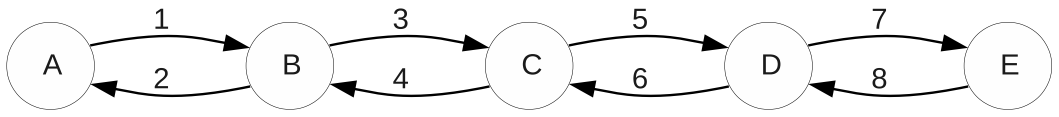


Figure 7.13: The network graph of the chain topology.

values of I. Only when $I = 2$ does the partial interference controller have significantly low optimality, whereas the maximal clique controller always does poorly. These results show that when contention between interferers is only partial, the partial interference controller performs much better in the face of complete interference ($a = 1$) than it does when interferers are completely dependent. The negative impact is most significant when $I = 2$ because with more interferers, the interferers send and receive at rates closer to that of the link being interfered with (due to more contention between them). Thus at $I = 2$ the disparity between the receiving rate of the heavily interfered link and the interfering links is largest and most unfair.

7.5 Chain Topology

The chain topology has a network graph of 5 nodes along a line, and active links both ways between each node, as seen in Figure 7.13. Since many of the links are connected by a node, there is a significant amount of perfect contention ($c = 1$), typical of a real network. It is easier to describe the exact topology of the contention/interference graph by displaying the

a and c values in matrix form:

$$a = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.4 & 0.2 & 0.4 & 0.4 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.2 & 0.0 & 0.1 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.2 \\ 0.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.1 & 0.0 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.4 & 0.4 & 0.4 & 0.4 & 0.0 & 0.0 & 0.0 \end{bmatrix}, \tag{7.3}$$

$$c = \begin{bmatrix} 0.0 & 1.0 & 1.0 & 0.6 & 0.6 & 0.3 & 0.3 & 0.0 \\ 1.0 & 0.0 & 1.0 & 1.0 & 1.0 & 0.6 & 0.6 & 0.3 \\ 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 0.6 & 0.6 & 0.3 \\ 0.6 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 1.0 & 0.6 \\ 0.6 & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 0.6 \\ 0.3 & 0.6 & 0.6 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 \\ 0.3 & 0.6 & 0.6 & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 \\ 0.0 & 0.3 & 0.3 & 0.6 & 1.0 & 1.0 & 1.0 & 0.0 \end{bmatrix}. \tag{7.4}$$

Essentially, values were chosen to reflect the distances between nodes in Figure 7.13. Note that (due to the proximity ordering chosen for the links) basing the parameters a and c on distance results in a nearly banded diagonal structure in the matrices.

The maximal clique controller achieved an optimality of 0.861. The partial interference controller achieved an optimality of 0.978. It is possible that the true optimalities are much worse, since the certainty of the first-principles branch and bound solution is only 0.709. However, based on the lengthy runs of the branch and bound algorithm as discussed in §6.1, we believe it is unlikely that the true optimal score is closer to the upper bound,

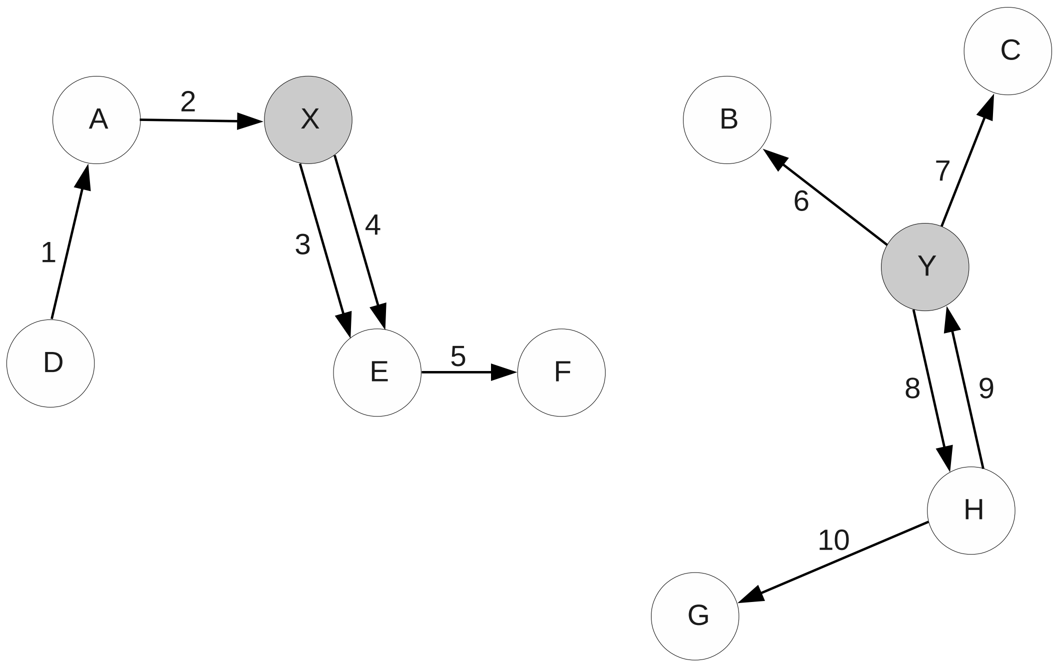


Figure 7.14: The network graph of the mesh topology. Vertices are network nodes or routers. The numbered directed edges are the active links in the network at some given moment. This network was constructed based on the positioning of nodes on one floor of a real mesh network deployed in a building. It can be thought of as a multi-hop Internet access network for users, where nodes X and Y are the access points to the cloud. Note that just about any subset of the pairs of nodes could be the links that are simultaneously active, and the choice for this particular problem is somewhat arbitrary. Keep in mind also that, although with the depicted active links it appears to be two adjacent networks, they do indeed contend and interfere with each other.

and may very well be precisely at the lower bound. The branch and bound algorithm ran on this topology for approximately 16 hours and 60,000 iterations.

7.6 Mesh Topology

Finally, we created a scenario that could be viewed as a typical flow of traffic through a mesh network providing Internet access to users, depicted in Figure 7.14. The constructed parameters of c and a between nodes are reported in Table 7.1, in the format c_{MN} for how much node M senses node N, and a^{Q}_{MN} for how much node Q interferes with transmissions from M to N. Values not shown are zero. We report contention and interference on a per-node basis for this network because it is more compact and it is actually the measurements one would obtain on a real network if following the methodology in [21]. With these values,

$c_{DA} = 0.97$	$c_{AD} = 0.97$
$c_{AX} = 1.00$	$c_{XA} = 1.00$
$c_{XE} = 0.98$	$c_{EX} = 0.98$
$c_{YH} = 0.90$	$c_{HY} = 0.93$
$c_{DX} = 0.64$	$c_{XD} = 0.84$
$c_{DE} = 0.88$	$c_{ED} = 0.81$
$c_{AE} = 0.40$	$c_{EA} = 0.33$
$c_{EH} = 0.08$	$c_{HE} = 0.21$
$c_{EY} = 0.40$	$c_{YE} = 0.55$
$a^{X}_{DA} = 0.16$	$a^{E}_{DA} = 0.10$
$a^{E}_{AX} = 0.60$	$a^{Y}_{AX} = 0.22$
$a^{D}_{XE} = 0.16$	$a^{Y}_{XE} = 0.71$
$a^{H}_{XE} = 0.04$	$a^{A}_{EF} = 0.07$
$a^{Y}_{EF} = 0.05$	$a^{H}_{EF} = 0.18$
$a^{X}_{HY} = 0.31$	$a^{E}_{HY} = 0.50$
$a^{E}_{YH} = 0.28$	$a^{Y}_{HG} = 0.03$
$a^{E}_{HG} = 0.45$	$a^{H}_{YC} = 0.04$
$a^{E}_{YC} = 0.26$	$a^{X}_{YC} = 0.21$
$a^{X}_{YB} = 0.83$	$a^{E}_{YB} = 0.40$

Table 7.1: Parameters in the mesh topology.

and with the facts that two links with the same sending node contend perfectly and cannot interfere, the appropriate c and a values between links are implied.

Like the chain topology, the mesh topology has a good mix of partial and binary contention, with some partial interference. The maximal clique controller obtained 0.924 optimality, the partial interference controller obtained 0.97 optimality, and the branch and bound algorithm returned a certainty of 0.807 for the optimal score, after running for several days and approximately 78,000 iterations. As a side note, a certainty of 0.795 was obtained after only a few minutes, and the lower bound never moved (as was the case for all topologies in this thesis).

7.7 Summary of Results

The take-home message of all these numerical results is that the maximal clique controller usually has significant performance loss, whereas the partial interference controller seems to only have significant performance loss in select topologies where the approximations introduced by the partial interference model sometimes cause the controller to either significantly under-utilize the medium or significantly starve links. Specifically, we have seen that when a network is plagued with a lot of partial carrier sensing, but also has very little interference, the partial interference model treats the partial carrier sensing as full carrier sensing, and thus under-utilizes the medium. We have also seen that when a network has several interferers that can carrier sense each other (near) perfectly, *and* when that interference is very strong (nearly 1), the partial interference model will make the mistake of having the clique of interferers send at or close to the full clique capacity, thus starving the link being interfered with.

Despite these significant failures in specific instances for the partial interference model (optimalities as low as 0.33), the results on the larger networks of the chain and mesh topologies indicate that perhaps these dangerous motifs either do not occur frequently, or their detriment is washed out by the good performance in other parts of the network. In both topologies, the partial interference controller scored above 0.97. Of course, the results of the chain and mesh topologies cannot prove such high optimality for other topologies of similar size—but combined with the near-exhaustive search over the small topologies, it is a strong indication that the partial interference model is good enough for the purposes of rate control.

Chapter 8

Conclusion

We have addressed the question of when certain classical rate controllers for wireless networks fail to perform well due to partial carrier sensing and interference. These classical controllers are developed using the network utility maximization (NUM) approach, where a sum of utility functions of the rates is maximized, subject to congestion and interference constraints. The models differ primarily on how the constraints are formulated, based on some model of how the MAC layer operates. The two classical models we have studied here are the maximal clique model, which constrains the sum of rates of mutually contending links, and the partial interference model, which builds off the former by constraining the receiving rates according to Niculescu's model of interference [21].

To discover the situations in which these classical controllers perform sub-optimally, we developed a model of wireless networks, called the first-principles model, that fully incorporates partial carrier sensing and interference, and we showed that it reduces to the classical models precisely when these effects are ignored. Thus the model predicts the same constraints as the classical models when these effects are not present, and can only be more accurate to reality when they are present. This provides us with an optimization problem that, when solved, gives an optimal score against which the sub-optimal scores of the classical controllers can be measured.

The first-principles model induces a non-convex problem, which requires a branch and bound method to solve globally. We have presented a particular branch and bound method, and we ran it for several network topologies. For larger topologies, the gap between the lower and upper bounds was still significant after running the algorithm for a long time (uncertainty as low as $lower/upper = 0.7$), but there were strong indications that the lower bound was very close to or at the optimal. In most small topologies that had a significant amount of partial carrier sensing or interference, the maximal clique controller performed poorly. For example, for the simple two-link topology, iterated over every combination of parameters, it performed below 0.9 optimality nearly 40% of the time, the worst being 0.648. The partial interference controller, however, only performed below 0.9 approximately 5% of the time, its worst being 0.776.

There were still other scenarios in which the partial interference controller performed very badly. The worst, at 0.33, consisted of two interferers on one link, where the two interferers could carrier sense each other perfectly (were dependent) and the interference was full. It also performed as badly as the maximal clique controller in scenarios with significant partial carrier sensing and no interference, since the two models differ from each other precisely when there is interference. We found indication that these scenarios, however, are isolated, meaning that small variations would allow the partial interference model to perform well again. The dependent interferers would lose their impact quickly when the interference factor was not close to 1. Even a value of 0.6 would be insignificant. Many partially contending links would still result in high performance for the partial interference model in many cases when interference was also introduced into the network. Finally, in the larger network configurations we tested, the partial interference controller performed above 0.97. We therefore conclude that the partial interference controller is likely good enough in most networks. But most importantly, we now have a tool to find out if a rate controller is performing sub-optimally (with respect to partial carrier sensing and interference effects), and we have discovered the motifs that will most likely be the cause of poor performance for the classical controllers studied here.

The next step for future research in this area is to test the accuracy of the first-principles model on real wireless networks, to see if the rates predicted are the rates actually acheived. Furthermore, the tests on a real network could benefit in understanding how rate controllers behave when the desired rates are actually infeasible. In this thesis, it is simply assumed that the classical rate controllers, when choosing infeasible rates, still remain fair in a particular sense. However, it would be interesting to find out, for example, if 802.11 might still starve certain links in such cases.

Concerning the first-principles model and its optimization problem, another interesting area for future research would be to explore ways of utilizing the benefits of the model to produce a distributed rate controller based on some approximation of the original problem. This would especially be viable if we knew that interior point methods always gave the optimal answer or close to it. We have already seen indications in our runs of the branch and bound algorithm that a simple interior point method might be sufficient. In fact, we did not find a single case where we ran branch and bound for which the lower bound would move from its initial value. This may also mean that there exists a convex reformulation of the optimization problem by a change of variables. If such a reformulation is found, it would mean that the lower bound in the branch and bound method is provably the optimal score, and even large networks could be solved with 100% certainty in a very short time. Not only would this allow for possibly using the problem or an approximation of it to develop a distributed solution, but it would also allow for exploring numerically many more large topologies to compare the classical controllers against the optimal.

Appendix A

Branch and Bound Code

In this appendix, we present primary pieces of code used to implement the branch and bound algorithm for the first-principles NUM problem, as well as the code for setting up an instance of the problem and finding the solutions that the classical controllers produce. The code was tested and debugged by checking both the form of the automated constraint functions, the branching and pruning steps, and the final output for small problems that could be verified by hand. Also, bad exit flags set by MATLAB's `fmincon` were recorded to notify the user of an invalid run.

solve.py

This Python code sets up an instance of the problem by creating the appropriate constraint functions, and then calls the MATLAB script to solve the problem.

```
# solve.py

"""
Solves for the best sending and receiving rates of a given topology.
Uses proportional fairness over links.
"""

import sys
import os
import time
import copy



```

```
# returns 2d (square) list of floats from file
def getMatrix(filename):
    matrix = []
    f = open(filename)
    lines = f.readlines()
    f.close()
    n = len(lines)
    for i in range(n):
        row = []
        stringRow = lines[i].split()
        for j in range(n):
            row.append(float(stringRow[j]))
        matrix.append(row)
    return matrix

# returns list of floats from file
def getRowVector(filename):
    row = []
    f = open(filename)
    lines = f.readlines()
    f.close()
    stringRow = lines[0].split()
    for j in range(len(stringRow)):
        row.append(float(stringRow[j]))
    return row

def powerSet(li, size=0):
    P = [] # List of lists. Power set of li
    for num in range(1, 2**len(li)):
        subset = [] # List. Subset of li
        binNum = bin(num) # Convert to binary string
        binNum = binNum[2:] # Remove the '0b'
        if size != 0: # If subset size matters
            subsetSize = 0 # Initialize
            for bit in binNum:
                if bit == '1':
                    subsetSize += 1 # Increment
            if size != subsetSize: # If not prescribed size
                continue # Skip this subset
        while len(binNum) < len(li):
            binNum = '0' + binNum # Include trailing zeros
        for i in range(len(li)):
            if binNum[i] == '1':
                subset.append(li[i]) # Each '1' indicates inclusion of element in li
        P.append(subset)
    return P

# returns string of + if size of p is odd, else -
def getSign(p):
    if len(p)%2 == 1:
        return '+'
    else:
        return '-'

def expandClique(clique, c):
    expandedCliques = []
    for i in range(clique[-1]+1, len(c)):
        canAppend = True
        for j in clique:
            if c[i][j] == 0:
                canAppend = False
                break
        if canAppend:
            expandedClique = list(clique)
            expandedClique.append(i)
            expandedCliques.append(expandedClique)
    return expandedCliques

def subsetOf(testClique, clique):
    for i in testClique:
        if i not in clique:
            return False
    return True

def contained(testClique, cliques):
    for clique in cliques:
        if subsetOf(testClique, clique):
            return True
    return False

# c is the contention graph (or matrix)
def getMaximalCliques(c):
    cliques = []
    maxCliques = []
    for i in range(len(c)):
        cliques.append([i])
    while len(cliques) > 0:
        clique = cliques.pop(0)
        expandedCliques = expandClique(clique, c)
        if len(expandedCliques) == 0:
            if not contained(clique, cliques) and not contained(clique, maxCliques):
                maxCliques.append(clique)
        else:
            for newClique in expandedCliques:
                cliques.append(newClique)
    return maxCliques

# cleans up string representation of float (getting rid of redundant zeros)
def formatFloat(x):
    f = '%f' % x
    if f.startswith('0.'):
        f = f[1:]
    if f.find('.') == -1:
        return f
    while f.endswith('0'):
        f = f[0:-1]
    if f.endswith('.'):
        f = f[0:-1]
    if f == '':
        return '0'
    return f

# ##############################################################################

class Solver:

    def __init__(self, a, c, path):
        self.a = a
        self.c = c
        self.n = len(a)
        self.constraintsIndex = 0;
        self.altConstraintsIndex = 0;
        self.bensaouConstraintsIndex = 0;
        self.wangConstraintsIndex = 0;
        self.path = path

        self.contention = self.findContentionGraph()
        self.bensaouContention = self.findBensaouContentionGraph()
        self.wangContention = self.findWangContentionGraph()

        self.cliques = self.findMaxCliques()
        self.bensaouCliques = self.findBensaouMaxCliques()
        self.wangCliques = self.findWangCliques()

        self.independentCliqueSets = self.findIndependentCliqueSets()

    def run(self):
        self.writeMatlabCode()
        cmd = 'matlab -nodisplay -nodesktop -nojvm -nosplash -r "solve_no_memory(%d, \'%s\')
            ;quit"' % (self.n, self.path)
        # os.system(cmd)

    # writes the preliminary m-files
    def writeMatlabCode(self):
        self.writeConstraints()
        self.writeAltConstraints()
        self.writeReceivingRates()
        self.writeBensaouConstraints()
        self.writeWangConstraints()
        self.writeWangReceivingRates()

    def findContentionGraph(self):
        contention = []
        for i in range(self.n):
            contention.append([])
            for j in range(self.n):
                contention[-1].append(0)
        for i in range(self.n):
            for j in range(i+1, self.n):
                if self.indep(i, j) == 0:
                    contention[i][j] = 1
                    contention[j][i] = 1
        return contention
    def findBensaouContentionGraph(self):
        contention = []
        for i in range(self.n):
            contention.append([])
            for j in range(self.n):
                contention[-1].append(0)
        for i in range(self.n):
            for j in range(i+1, self.n):
                if self.indep(i, j)*self.indepRecv(i, j) < 0.5:
                    contention[i][j] = 1
                    contention[j][i] = 1
        return contention

    def findWangContentionGraph(self):
        contention = []
        for i in range(self.n):
            contention.append([])
            for j in range(self.n):
                contention[-1].append(0)
        for i in range(self.n):
            for j in range(i+1, self.n):
                if self.indep(i, j) < 0.5:
                    contention[i][j] = 1
                    contention[j][i] = 1
        return contention

    # get all maximal cliques for the strict contention graph of the first-principles model
    def findMaxCliques(self):
        return getMaximalCliques(self.contention)

    # get all maximal cliques for the contention graph of the Bensaou model
    def findBensaouMaxCliques(self):
        return getMaximalCliques(self.bensaouContention)

    # get all maximal cliques for the contention graph of the Wang model
    def findWangCliques(self):
        return getMaximalCliques(self.wangContention)

    # get all sets of maximal sets of independent cliques for the strict contention graph
    def findIndependentCliqueSets(self):
        cliqueSets = []
```

```python
        maxCliqueSets = []
        for i in range(self.n):
            cliqueSets.append([[i]])
        while len(cliqueSets) > 0:
            cliqueSet = cliqueSets.pop(0)
            expandedCliqueSets = self.expandCliqueSet(cliqueSet)
            if len(expandedCliqueSets) == 0:
                if not self.contained(cliqueSet, cliqueSets) and not self.contained(cliqueSet, maxCliqueSets):
                    maxCliqueSets.append(cliqueSet)
            else:
                for newCliqueSet in expandedCliqueSets:
                    cliqueSets.append(newCliqueSet)
        return maxCliqueSets

    def writeConstraints(self):
        f = open('constraints.m', 'w')
        f.write('function [C,Ceq] = constraints(x)\n\nn = length(x);\ns = x(1:n/2);\ny = x(n/2+1:n);\n\nCeq = [];\n\n')
        for i in range(self.n):
            f.write(self.getSendConstraint(i))
        for i in range(self.n):
            f.write(self.getRecvConstraint(i))
        f.write('\nfor i = 1:length(C)\nif ~(C(i) < inf)||imag(C(i))~=0\nC(i) = 1000;\nend\nend')
        f.close()

    def writeAltConstraints(self):
        f = open('altConstraints.m', 'w')
        f.write('function [C,Ceq] = altConstraints(x)\n\nn = length(x);\ns = x(1:n/2);\ny = x(n/2+1:n);\n\nCeq = [];\n\n')
        for i in range(self.n):
            f.write(self.getAltSendConstraints(i))
        for i in range(self.n):
            f.write(self.getAltRecvConstraints(i))
        f.write('\nfor i = 1:length(C)\nif ~(C(i) < inf)||imag(C(i))~=0\nC(i) = 1000;\nend\nend')
        f.close()

    def writeReceivingRates(self):
        f = open('receivingRates.m', 'w')
        f.write('function r = receivingRates(s)\n\n')
        for i in range(self.n):
            f.write('r(%d) = s(%d)' % (i+1, i+1))
            R = self.getR(i)
            if R != '0':
                f.write('*(1-(%s))' % R)
            f.write(';\n')
        f.close()

    def writeBensaouConstraints(self):
        f = open('bensaouConstraints.m', 'w')
        f.write('function [C,Ceq] = bensaouConstraints(s)\n\nCeq = [];\n\n')
        for clique in self.bensaouCliques:
            f.write(self.getBensaouSendConstraint(clique))
        f.close()

    def writeWangConstraints(self):
        f = open('wangConstraints.m', 'w')
        f.write('function [C,Ceq] = wangConstraints(s)\n\nCeq = [];\n\n')
        for clique in self.wangCliques:
            f.write(self.getWangSendConstraint(clique))
        f.close()

    def writeWangReceivingRates(self):
        f = open('wangReceivingRates.m', 'w')
        f.write('function r = wangReceivingRates(s)\n\n')
        for i in range(self.n):
            f.write('r(%d) = s(%d)' % (i+1, i+1))
            for j in range(self.n):
                if i != j and self.a[i][j] != 0:
                    f.write('*(1-%s*s(%d))' % (formatFloat(self.a[i][j]), j+1))
            f.write(';\n')
        f.close()

    # independence of two links i and j
    def indep(self, i, j):
        return 1.0 - self.c[i][j] - self.c[j][i] + self.c[i][j]*self.c[j][i]

    # independence of two links i and j by receving rates, as measured by a values (not c
        values)
    def indepRecv(self, i, j):
        return 1.0 - self.a[i][j] - self.a[j][i] + self.a[i][j]*self.a[j][i]

    def expandCliqueSet(self, cliqueSet):
        expandedCliqueSets = []
        first = 1 + cliqueSet[-1][-1]
        for i in range(first, self.n):
            expandedCliqueSet = cliqueSet[:]
            isCliqueCount = 0
            canAppend = True
            for j in range(len(expandedCliqueSet)):
                clique = expandedCliqueSet[j][:]
                if self.isDependent(i, clique): # make function
                    isCliqueCount += 1
                    clique.append(i)
                    expandedCliqueSet[j] = clique
                elif not self.isIndependent(i, clique): # make function
                    canAppend = False
                    break
                if isCliqueCount > 1:
                    canAppend = False
                    break
            if not canAppend:
                continue
            if isCliqueCount == 0:
                expandedCliqueSet.append([i])
            expandedCliqueSets.append(expandedCliqueSet)
        return expandedCliqueSets

    def contained(self, testCliqueSet, cliqueSets):
        li1 = []
        for c in testCliqueSet:
            for i in c:
                li1.append(i)
        for cliqueSet in cliqueSets:
            li2 = []
            for c in cliqueSet:
                for i in c:
                    li2.append(i)
            if subsetOf(li1, li2):
                return True
        return False

    def getSendConstraint(self, i):
        self.constraintsIndex += 1
        S = self.getS(i)
        constraint = 'C(%d) = -1+s(%d)' % (self.constraintsIndex, i+1)
        if S != '0':
            constraint += S
        constraint += ';\n'
        return constraint

    def getRecvConstraint(self, i):
        self.constraintsIndex += 1
        R = self.getR(i)
        constraint = 'C(%d) = -1+y(%d)' % (self.constraintsIndex, i+1)
        if R != '0':
            constraint += '+' + R
        constraint += ';\n'
        return constraint

    def getAltSendConstraints(self, i):
        constraints = ''
        for clique in self.cliques:
            constraints += self.getAltSendConstraint(i, clique)
        return constraints

    def getAltRecvConstraints(self, i):
        constraints = ''
        for cliqueSet in self.independentCliqueSets:
            constraints += self.getAltRecvConstraint(i, cliqueSet)
        return constraints

    def getBensaouSendConstraint(self, clique):
        self.bensaouConstraintsIndex += 1
        constraint = 'C(%d) = -1' % self.bensaouConstraintsIndex
        for i in clique:
            constraint += '+s(%d)' % (i+1,)
        constraint += ';\n'
        return constraint

    def getWangSendConstraint(self, clique):
        self.wangConstraintsIndex += 1
        constraint = 'C(%d) = -1' % self.wangConstraintsIndex
        for i in clique:
            constraint += '+s(%d)' % (i+1,)
        constraint += ';\n'
        return constraint

    # returns true if adding i to clique still makes it a clique
    def isDependent(self, i, clique):
        for j in clique:
            if self.contention[i][j] == 0:
                return False
        return True

    # returns true if i is not connected to any element in clique
    def isIndependent(self, i, clique):
        for j in clique:
            if self.contention[i][j] == 1:
                return False
        return True

    def getS(self, i):
        S = ''
        for subset in powerSet(self.L(i)):
            h = self.getH(subset)
            f = self.getF(i, subset)
            if h == '0' or f == '0':
                continue
            S += getSign(subset) + f
            g = self.getG(i, subset)
            if g != '1':
                S += '*' + g
            if h != '1':
                S += '*' + h
        if S == '':
            S = '0'
        return S

    def getR(self, i):
        R = ''
        for subset in powerSet(self.L(i)):
            h = self.getH(subset)
            fp = self.getFp(i, subset)
            if h == '0' or fp == '0':
                continue
            sign = getSign(subset)
            R += '%s%s' % (sign, fp)
            if h != '1':
                R += '*%s' % h
        if R == '':
            return '0'
        if R.startswith('+'):
            R = R[1:]
        return R

    def getAltSendConstraint(self, i, clique):
        S = ''
        for j in clique:
            if j == i:
                continue
            if self.c[i][j] == 0:
                continue
            if self.c[i][j] == 1:
                S += '+s(%d)' % (j+1,)
            else:
                S += '+%s*s(%d)' % (formatFloat(self.c[i][j]), j+1)
        if S == '':
            return ''
        self.altConstraintsIndex += 1
        constraint = 'C(%d) = -1+s(%d)%s;\n' % (self.altConstraintsIndex, i+1, S)
        return constraint

    def getAltRecvConstraint(self, i, cliqueSet):
        logs = ''
        for clique in cliqueSet:
            R = ''
            for j in clique:
                if j == i:
                    continue
                if self.a[i][j] == 0:
                    continue
                if self.a[i][j] == 1:
                    R += '-s(%d)' % (j+1,)
                else:
                    R += '-%s*s(%d)' % (formatFloat(self.a[i][j]), j+1)
            if R != '':
                logs += '-log(1%s)' % R
        if logs == '':
            return ''
        self.altConstraintsIndex += 1
        constraint = 'C(%d) = log(y(%d))%s;\n' % (self.altConstraintsIndex , i+1, logs)
        return constraint

    # returns list of ints from 0 to n-1 excluding i
    def L(self, i):
        li = range(self.n)
        li.pop(i)
        return li

    # returns string for h function
    def getH(self, p):
        pairs = powerSet(p, 2)
        # iterate through c values, if any c=1, just return '0'
        # if all c=0, return '1'
        nonzeros = False
        for (i,j) in pairs:
            if self.c[i][j] == 1 or self.c[j][i] == 1:
                return '0'
            if (not nonzeros) & (self.c[i][j] != 0 or self.c[j][i] != 0):
                nonzeros = True
        if not nonzeros:
            return '1'
        num = 1.0
        for (i,j) in pairs:
            num *= self.indep(i, j)
        return formatFloat(num)

    # returns string for f function
    def getF(self, i, p):
        num = 1.0
        for j in p:
            if self.c[i][j] == 0.0:
                return '0'
            num *= self.c[i][j]
        f = ''
        if num != 1.0:
            f = formatFloat(num)
        for j in p:
            f += '*s(%d)' % (j+1,)
        if f.startswith('*'):
            f = f[1:]
        return f

    # returns string for g function
    def getG(self, i, p):
        if len(p) == 1:
            return '1'
        (phi, num) = self.getPhi(i, p)
        denominator = ''
        cancelled = False
        for j in p:
            if num == self.c[j][i] and not cancelled:
                cancelled = True
                continue
            if self.c[j][i] != 1.0:
                denominator += '(1-%s*s(%d))*' % (formatFloat(self.c[j][i]), i+1)
            else:
                denominator += '(1-s(%d))*' % (i+1,)
        denominator = denominator[0:-1]
        if cancelled:
            g = '1/(%s)' % denominator
        else:
            g = '(%s)/(%s)' % (phi, denominator)
        return g

    # returns string for f prime function
    def getFp(self, i, p):
        num = 1.0
        for j in p:
            if self.a[i][j] == 0.0:
                return '0'
            num *= self.a[i][j]
        fp = ''
        if num != 1.0:
            fp = formatFloat(num)
        for j in p:
            fp += '*s(%d)' % (j+1,)
        if fp.startswith('*'):
            fp = fp[1:]
        return fp

    # returns string for phi function
    def getPhi(self, i, p):
        num = 0.0
        for subset in powerSet(p):
            product = 1.0
            for j in subset:
                product *= self.c[j][i]
            if getSign(subset) == '+':
                num += product
            else:
                num -= product
        if num != 1.0:
            phi = '1-%s*s(%d)' % (formatFloat(num), i+1)
        else:
            phi = '1-s(%d)' % (i+1,)
        return (phi, num)


# ######################################################################

if __name__ == '__main__':
    if len(sys.argv) > 2:
        print 'Usage:'
        print 'python solve.py [path]'
        print 'path: relative path in which to find the input files named a and c, also for
            output data'
        print 'a: text file containing receiving interference coefficients'
        print 'c: text file containing carrier sensing coefficients'
        print 'Each file must be formatted like a matrix, where each new line depicts a new
            row, and each element is delimited by white space.'
        print 'For n links:'
        print '\ta is n x n'
        print '\tc is n x n'
        print 'They must be named a and c, without any file extension.'
        print 'If no argument is given, the current directory is assumed.'
        print 'Use forward slash (/) for directories, not back slash.'
        quit()

    if len(sys.argv) == 1:
        aPath = 'a'
        cPath = 'c'
        path = ''
    else:
        aPath = '' + sys.argv[1]
        cPath = '' + sys.argv[1]
        path = '' + sys.argv[1]
        if not sys.argv[1].endswith('/'):
            aPath += '/'
            cPath += '/'
            path += '/'
        aPath += 'a'
        cPath += 'c'

    a = getMatrix(aPath)
    c = getMatrix(cPath)

    if len(a) != len(c):
        print 'Matrix dimensions must agree.'
        quit()

    solver = Solver(a, c, path)
    solver.run()

# ###################################################################

def run_solver(rawpath=''):
    if rawpath == '':
        aPath = 'a'
        cPath = 'c'
        path = ''
    else:
        aPath = '' + rawpath
        cPath = '' + rawpath
        path = '' + rawpath
        if not rawpath.endswith('/'):
            aPath += '/'
            cPath += '/'
            path += '/'
        aPath += 'a'
        cPath += 'c'

    a = getMatrix(aPath)
    c = getMatrix(cPath)

    if len(a) != len(c):
        print 'Matrix dimensions must agree.'
        return -1

    solver = Solver(a, c, path)
    solver.run()
    return 1
```

solve.m

This is the MATLAB script that has at its heart the branch and bound algorithm. The algorithm follows the procedure given in [25]. The `Region` class holds the lower and upper bound information for a particular region. These lower and upper bounds are calculated by making calls to the `fmincon` function in MATLAB's optimization toolbox.

```
% solve_no_memory.m

% Does not keep a history of past iterations.

% N = number of links in network
% path = directory (with slash, if not empty string for current directory) to save output

% When a region is smaller than .1 in every dimension, its upper bound is
% set equal to its lower bound, because we don't want to search any finer
% than that.

% each iteration does two new regions
% each region does one or two calls to fmincon

% exit status, or flag:
%   1 = normal
%   2 = reached maximum iterations (maxK)
%   -1 = fmincon produced at least one bad flag

function solve_no_memory(N, path)

% user provided parameters
eps = 1e-2; % threshold: stop when U - L <= eps
maxK = N; % max number of iterations
volumeCutoff = 0;

% start timer
tic

% Bensaou model
options = optimset('Algorithm', 'interior-point', 'Display', 'off');
[bensaouS, bensaouFval] = fmincon(@bensaouObjective, zeros(1,N), ...
    [], [], [], [], zeros(1,N), ones(1,N), ...
    @bensaouConstraints, options);

% partial interference (wang) model
[wangS, wangFval] = fmincon(@wangObjective, zeros(1,N), ...
    [], [], [], [], zeros(1,N), ones(1,N), ...
    @wangConstraints, options);

% initialize
k = 1;
% Note: rectangle formed by lb and ub is in [s r./s] space.
lbinit = zeros(1,2*N);
ubinit = ones(1,2*N);
regioninit = Region(lbinit, ubinit);
regioninit = solveLower(regioninit);
regioninit = solveUpper(regioninit);
regions = regioninit;
L = regioninit.scorelow;
U = regioninit.scorehi;
sBest = regioninit.slow;
rBest = regioninit.rlow;
sinit = regioninit.slow;
rinit = regioninit.rlow;
Linit = L;
Uinit = U;
```

```
totalVolumePruned = 0;
flag = checkFlags(regioninit); % Keeps track if any one fmincon produced a bad flag

% display progress
clc
fprintf('time = %s\n', secs2hms(toc));
fprintf('iteration = %d\n\n', k);
fprintf('upper = %f\n', U);
fprintf('lower = %f\n\n', L);
fprintf('certainty = %f\n', totalVolumePruned);
fprintf('regions = %d\n', length(regions));

while U-L > eps && k < maxK
    % branch
    if totalVolumePruned < volumeCutoff
        [region, n] = getLargestRegion(regions);
        if sameVector(region.xlow, region.ub)
            [region, n] = getNextRegion(regions, U);
        end
    else
        [region, n] = getNextRegion(regions, U);
    end
    [reg1, reg2] = split(region);
    regions = [regions(1:n-1), regions(n+1:end), reg1, reg2];
    % bound
    [L, sBest, rBest] = maxScorelow(regions);
    U = maxScorehi(regions);
    % prune
    [regions, volPruned] = prune(regions, L);
    totalVolumePruned = totalVolumePruned + volPruned;
    % check flags
    flag = min(flag, checkFlags(reg1));
    flag = min(flag, checkFlags(reg2));
    % iterate
    k = k+1;
    % display progress
    clc
    fprintf('time = %s\n', secs2hms(toc));
    fprintf('iteration = %d\n\n', k);
    fprintf('upper = %f\n', U);
    fprintf('lower = %f\n\n', L);
    fprintf('certainty = %f\n', totalVolumePruned);
    fprintf('regions = %d\n', length(regions));
end

% values to report
if k == maxK && flag == 1
    flag = 2;
end
s = sBest;
r = rBest;
score = L;
bound = U;
difference = bound - score;
numActiveRegions = length(regions);
certainty = totalVolumePruned;
[furthestS, radius] = getFurthest(s, regions);
initialS = sinit;
initialR = rinit;
initialScore = Linit;
initialBound = Uinit;
initialDifference = score - initialScore;
distance = norm(s - initialS) / norm(ones(1,N));
% Note: distance and radius are normalized by the norm of the 1 vector,
% which is the maximum line distance traversing the unit hypercube (in
% s-space).
bensaouPredictedS = bensaouS;
[bensaouTrueS, bensaouInfeasibility] = findMaxFeasible(bensaouS);
bensaouPredictedR = bensaouPredictedS;
bensaouTrueR = receivingRates(bensaouTrueS);
bensaouPredictedScore = exp(-bensaouFval/N);
bensaouTrueScore = exp(sum(log(bensaouTrueR))/N);
bensaouDistance = norm(bensaouPredictedS - s) / norm(ones(1,N));
bensaouDifference = score - bensaouTrueScore;
wangPredictedS = wangS;
[wangTrueS, wangInfeasibility] = findMaxFeasible(wangS);
wangPredictedR = wangReceivingRates(wangPredictedS);
wangTrueR = receivingRates(wangTrueS);
wangPredictedScore = exp(-wangFval/N);
wangTrueScore = exp(sum(log(wangTrueR))/N);
wangDistance = norm(wangPredictedS - s) / norm(ones(1,N));
wangDifference = score - wangTrueScore;
```

```
time = toc;

% create report
txtfile = fopen(strcat(path, 'matlab.txt'), 'w');
fprintf(txtfile, 'First-principles:\ns =');
fprintf(txtfile, ' %f', s);
fprintf(txtfile, '\nr =');
fprintf(txtfile, ' %f', r);
fprintf(txtfile, '\nscore = %f\n', score);
fprintf(txtfile, 'bound = %f\n', bound);
fprintf(txtfile, 'difference = %f\n', difference);
fprintf(txtfile, 'threshold = %f\n', eps);
fprintf(txtfile, 'certainty = %f\n', certainty);
fprintf(txtfile, 'radius of uncertainty = %f\n', radius);
fprintf(txtfile, 'active regions = %d\n', numActiveRegions);
fprintf(txtfile, 'initial s =');
fprintf(txtfile, ' %f', initialS);
fprintf(txtfile, '\ninitial r =');
fprintf(txtfile, ' %f', initialR);
fprintf(txtfile, '\ninitial score = %f\n', initialScore);
fprintf(txtfile, 'difference b/t initial and best score = %f\n', initialDifference);
fprintf(txtfile, 'initial bound = %f\n', initialBound);
fprintf(txtfile, 'difference b/t initial and best bound = %f\n', initialBound-bound);
fprintf(txtfile, 'distance b/t initial and best s = %f\n', distance);
fprintf(txtfile, 'iterations = %d\n', k);
fprintf(txtfile, 'time = %s\n', secs2hms(time));
fprintf(txtfile, 'time in secs = %f\n', time);
fprintf(txtfile, 'exit status = %d\n', flag);
fprintf(txtfile, '\nPartial interference:\npredicted s =');
fprintf(txtfile, ' %f', wangPredictedS);
fprintf(txtfile, '\ntrue      s =');
fprintf(txtfile, ' %f', wangTrueS);
fprintf(txtfile, '\npredicted r =');
fprintf(txtfile, ' %f', wangPredictedR);
fprintf(txtfile, '\ntrue      r =');
fprintf(txtfile, ' %f', wangTrueR);
fprintf(txtfile, '\npredicted score = %f\n', wangPredictedScore);
fprintf(txtfile, 'true score = %f\n', wangTrueScore);
fprintf(txtfile, 'difference from optimal = %f\n', wangDifference);
fprintf(txtfile, 'distance b/t predicted s and optimal s = %f\n', wangDistance);
fprintf(txtfile, 'infeasibility = %f\n', wangInfeasibility);
fprintf(txtfile, '\nMaximal clique:\npredicted s =');
fprintf(txtfile, ' %f', bensaouPredictedS);
fprintf(txtfile, '\ntrue      s =');
fprintf(txtfile, ' %f', bensaouTrueS);
fprintf(txtfile, '\npredicted r =');
fprintf(txtfile, ' %f', bensaouPredictedR);
fprintf(txtfile, '\ntrue      r =');
fprintf(txtfile, ' %f', bensaouTrueR);
fprintf(txtfile, '\npredicted score = %f\n', bensaouPredictedScore);
fprintf(txtfile, 'true score = %f\n', bensaouTrueScore);
fprintf(txtfile, 'difference from optimal = %f\n', bensaouDifference);
fprintf(txtfile, 'distance b/t predicted s and optimal s = %f\n', bensaouDistance);
fprintf(txtfile, 'infeasibility = %f\n', bensaouInfeasibility);
fclose(txtfile);
```

References

[1] S. Xu and T. Saadawi, “Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?” *IEEE Communications Magazine*, vol. 39, no. 6, pp. 130–137, June 2001.

[2] J. Shi, O. Gurewitz, V. Mancuso, J. Camp, and E. Knightly, “Measurement and modeling of the origins of starvation in congestion controlled mesh networks,” in *INFOCOM 2008: IEEE International Conference on Computer Communications*, April 2008, pp. 1633–1641.

[3] S. Rangwala, A. Jindal, K.-Y. Jang, K. Psounis, and R. Govindan, “Understanding congestion control in multi-hop wireless mesh networks,” in *MobiCom 2008: ACM International Conference on Mobile Computing and Networking*, September 2008, pp. 291–302.

[4] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, “Rate control for communication networks: Shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[5] S. Low and D. Lapsley, “Optimization flow control—I: Basic algorithm and convergence,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, December 1999.

[6] L. Chen, S. Low, and J. Doyle, “Joint congestion control and media access control design for ad hoc wireless networks,” in *INFOCOM 2005: IEEE International Conference on Computer Communications*, March 2005, pp. 2212–2222.

[7] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu, “Impact of interference on multihop wireless network performance,” *Wireless Networks*, vol. 11, no. 4, pp. 471–487, 2005.

[8] Y. Xue, B. Li, and K. Nahrstedt, “Optimal resource allocation in wireless ad hoc networks: A price-based approach,” *IEEE Transactions on Mobile Computing*, vol. 5, pp. 347–364, 2006.

[9] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle, "Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks," in *INFOCOM 2006: IEEE International Conference on Computer Communications*, April 2006, pp. 1–13.

[10] B. Bensaou and Z. Fang, "A fair MAC protocol for IEEE 802.11-based ad hoc networks: Design and implementation," *IEEE Transactions on Wireless Communications*, vol. 6, no. 8, pp. 2934–2941, August 2007.

[11] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, and E. Rozner, "Predictable performance optimization for wireless networks," in *SIGCOMM 2008: ACM Conference of the Special Interest Group on Data Communication*, August 2008, pp. 413–426.

[12] L. Wang, D. Ripplinger, A. Rai, S. Warnick, and D. Zappala, "A convex optimization approach to decentralized rate control in wireless networks with partial interference," in *CDC 2010: IEEE Conference on Decision and Control*, December 2010, pp. 639–646.

[13] I. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Communications Magazine*, vol. 43, no. 9, pp. S23 – S30, 2005.

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

[15] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A multi-radio unification protocol for IEEE 802.11 wireless networks," in *BROADNETS 2004: International ICST Conference on Broadband Communications, Networks, and Systems*, October 2004, pp. 344–354.

[16] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill, "Estimation of link interference in static multi-hop wireless networks," in *IMC 2005: ACM SIGCOMM Conference on Internet Measurement*, October 2005, pp. 305–310.

[17] S. M. Das, D. Koutsonikolas, Y. C. Hu, and D. Peroulis, "Characterizing multi-way interference in wireless mesh networks," in *WiNTECH 2006: ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, September 2006, pp. 57–64.

[18] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 51–62, 2006.

[19] L. Qiu, Y. Zhang, F. Wang, M. K. Han, and R. Mahajan, "A general model of wireless interference," in *MobiCom 2007: ACM International Conference on Mobile Computing and Networking*, September 2007, pp. 171–182.

[20] A. Kashyap, S. Ganguly, and S. R. Das, "A measurement-based approach to modeling link capacity in 802.11-based wireless networks," in *MobiCom 2007: ACM International Conference on Mobile Computing and Networking*, September 2007, pp. 242–253.

[21] D. Niculescu, "Interference map for 802.11 networks," in *IMC 2007: ACM SIGCOMM Conference on Internet Measurement*, October 2007, pp. 339–350.

[22] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, October 2000.

[23] T. Lan, D. Kao, M. Chiang, and A. Sabharwal, "An axiomatic theory of fairness in network resource allocation," in *INFOCOM 2010: IEEE International Conference on Computer Communications*, March 2010, pp. 1–9.

[24] D. P. Bertsekas, *Nonlinear Programming.* Belmont, Massachusetts: Athena Scientific, 1995, pp. 598–601.

[25] S. Boyd and J. Mattingley, "Branch and bound methods," Lecture notes for EE364b, Stanford University, Winter 2006–07. [Online]. Available: http://www.stanford.edu/class/ee364b/notes/bb_notes.pdf