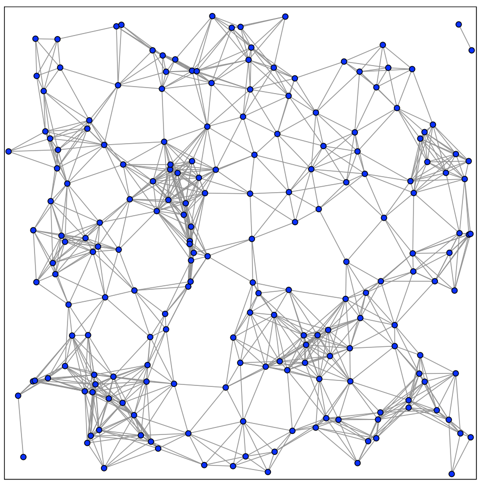
Universität Karlsruhe (TH)
Institut für Theoretische Informatik

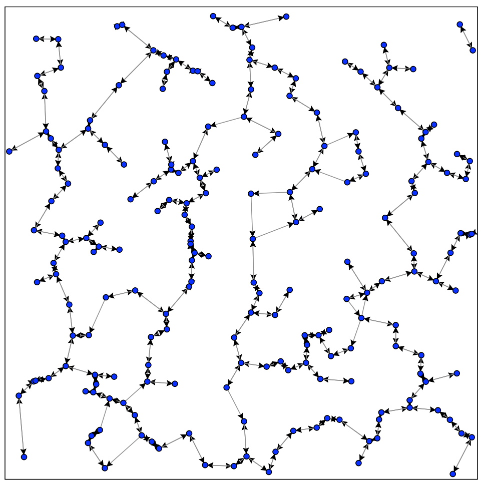
Diplomarbeit

Scheduling and Topology Control in Wireless Sensor Networks

Markus Völker

30. Oktober 2008





Betreut durch:

Universität Karlsruhe:	*Carnegie Mellon University:*
Dipl.-Inform. Bastian Katz	Ass.-Prof. Dr. Willem-Jan van Hoeve
Prof. Dr. Dorothea Wagner	Prof. Dr. R. Ravi

Acknowledgements

First of all, I want to thank my supervisors Bastian Katz and Prof. Wagner (University of Karlsruhe), as well as Ass.-Prof. van Hoeve and Prof. Ravi (Carnegie Mellon University), for all their help and support! In particular, I would like to thank Prof. Wagner for the opportunity to write my thesis at her institute, and for supporting my application for the interACT exchange program. As well, I want to thank Prof. van Hoeve and Prof. Ravi for the very obliging supervision and care during my time at Carnegie Mellon University. My most profound thanks go to Bastian Katz, for pointing me to the topic of sensor networks and for the great support and the helpful assistance throughout my work on this thesis.

I am much obliged to Agilent Technologies for the financial support during my studies. At this point, I also want to sincerely thank Stefan Weiss and Markus Walter for their support! Moreover, I am very grateful to the people behind the interACT exchange program for making my stay abroad possible.

Finally, I would like to thank Lars Hofer, Matthias Thoma, and Tatiane Utumi for proofreading this thesis, and Viswanath Nagarajan for useful discussions.

This thesis is dedicated to my parents, Alfred and Maria Völker.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig angefertigt habe und nur die angegebenen Hilfsmittel und Quellen verwendet wurden.

Karlsruhe, den 30. Oktober 2008

Unterschrift: ....

Abstract

Within recent years, wireless sensor networks became a very popular tool for distributed monitoring of various physical and environmental conditions. Thanks to the ongoing miniaturization of the required hardware, this trend is likely to continue. In this thesis, we study the problems of scheduling and topology control in wireless networks. We consider both, networks with fixed transmission powers, and networks with freely adjustable transmission powers. For the scheduling problem, several methods for the computation of optimum schedules, as well as the computation of upper and lower bounds on the length of optimum schedules, are presented and analyzed. Additionally, various possibilities for optimizing the proposed methods are described. All methods have been implemented and are used in an experimental section, to gain additional insights into the properties of wireless networks. Subsequently, the scheduling heuristics are extended to topology control algorithms. Goal of this topology control algorithms is the computation of topologies whose links can be scheduled efficiently. At the same time, the resulting topologies are required to conserve certain properties of the original network, such as connectivity or spanner properties. Furthermore, we propose a simple topology control algorithm that works local and does not require any information about node positions. The proposed topology control algorithms are compared to various existing algorithms, using several quality measures for network topologies.

A central part of this thesis is dedicated to the evaluation of constraint programming as a tool for solving scheduling problems based on the physically motivated SINR interference model. For this purpose, constraint programming is compared to integer linear programming, and several experiments are performed to highlight the pros and cons of both approaches. The constraint programs have been solved using the freely available Gecode solver, which is described to some detail in the beginning of this thesis.

Contents

1 Introduction

Networks of small sensor nodes, so-called *sensor networks*, consist of spatially distributed autonomous devices which use sensors to cooperatively monitor physical or environmental conditions, such as pressure, temperature or vibration, at different locations. Originally, the development of sensor networks was motivated by military applications such as battlefield surveillance. However, the ongoing miniaturization of the sensor nodes as well as the availability of cheaper and cheaper hardware led to an abundance of new applications, including habitat and environment monitoring, industrial machinery surveillance, healthcare and home automation.

Today, the smallest sensor nodes already have a diameter of only about one millimeter. Yet, there is no end in sight for the miniaturization and it is hoped that soon the sensor nodes will be small and cheap enough that thousands of them can be scattered in order to work cooperatively. This concept became known as *smartdust* [18] because the devices are intended to be only the size of a dust particle. Of course, these applications put high demands on the used hardware and software. Due to their tiny size, sensor nodes cannot be equipped with big batteries. Moreover, in many applications it is impossible to recharge or replace sensor nodes that run out of power. Therefore, energy conservation is very crucial in the context of sensor networks.

Since energy is the limiting factor of lifetime and operability of most ad-hoc networks, there has been a lot of research on how to conserve energy in sensor networks. One of the approaches proposed so far is *topology control*. The basic idea behind topology control is to restrict the network topology, the structure of links connecting pairs of network nodes, to a small connecting subset of all possible links in order to make routing on the topology faster and easier. Moreover, the energy needed for a transmission strongly depends on the distance between sender and receiver. The energy required for communication between two sensor nodes grows at least quadratically with their distance. Therefore, avoiding large-distance links usually helps to conserve energy.

Most of the theoretical work on sensor networks is based on oversimplified graph-based models. These models neither adequately take into account the dependence between transmission quality and sender-receiver-distance, nor do they consider that sensor nodes that are not within transmission range also interfere to a certain extent with each other. For this reason, this thesis relies on the more sophisticated *SINR model*, a physically motivated radio propagation model.

The main goals of this thesis are: providing efficient heuristics for the computation of good upper and lower bounds on the length of optimal schedules in wireless networks, to develop and study optimized methods for the computation of exact schedules, to use the aforementioned methods to analyze differences and similarities between different models for sensor networks, to propose new topology control algorithms which try to minimize the number of slots that are necessary to schedule all links of the generated topologies, to compare this topology control algorithms with existing algorithms, and, last but not least, to evaluate constraint programming and compare it to integer linear programming based on the experiments that were performed in the context of this thesis.

Organization of this thesis

In the following, we give a brief chapter outline in order to provide a quick overview on the organization of this thesis.

Chapter 2: The second chapter provides the reader with the necessary background knowledge that is not directly connected to wireless sensor networks. It starts with some basic notations and definitions in the field of *graph theory*. Subsequently, the techniques *mathematical programming* and *constraint programming* are introduced. As most of the readers are probably not so familiar with constraint programming, this topic is covered in more detail. The chapter is concluded with a detailed description of *Gecode*, the constraint programming solver that is used for the experiments in this thesis.

Chapter 3: A brief survey on sensor networks is given. First, it is shown how network topologies can be modeled using *communication graphs*. Two famous kinds of communication graphs, namely *unit disk graphs* and *quasi unit disk graphs*, are described. Next, different ways of how the *medium access layer* of the network can deal with interference and resulting transmission failures are shown. This is followed by the definition of the *scheduling problem*, one of the main topics of this thesis. Subsequently, a brief overview on how interference in wireless networks can be modeled is given. *Graph based models* of interference, as well as the more profound *SINR model*, are introduced and compared to each other. At this, the use of the SINR model is motivated. Finally, the *topology control problem* is introduced.

Chapter 4: The fourth chapter is dedicated to the *scheduling problem*. It starts with the question how one can decide if a set of transmissions can be scheduled at the same time, and how one can determine *optimal transmission powers* such that all transmissions satisfy the SINR constraint without wasting energy. Afterwards, constraint programs (CPs) and integer linear programs (ILPs) for the computation of optimum solutions of the scheduling problem are given. To push up the size of exactly solvable problems, several optimizations for the CPs and ILPs are discussed. However, real world instances of the scheduling problem are intractable with exact algorithms. Thus, efficient heuristics for the computation of upper and lower bounds on the length of optimal schedules are discussed.

Chapter 5: This chapter deals with the *topology control problem.* An overview of existing algorithms for the topology control problem is given and several *quality measures* for network topologies are described. The list of well-established quality measures is extended with new measures that are based on the SINR model. After that, the scheduling heuristics from chapter four are extended to topology control algorithms. Goal of this algorithms is the computation of topologies that can be fast scheduled and that, at the same time, preserve connectivity as well as certain spanner properties. Furthermore, another topology control algorithm, which tries to minimize the SINR model interference, is given. This chapter concludes with the description of a CP for the optimal solution of the aforementioned topology control problem.

Chapter 6: The experimental chapter starts with a description of implementation and testing environment. Afterwards, the SINR model parameters used for the experiments are described and motivated. Subsequently, the methods of Chapter 4 and Chapter 5 are used to examine some basic properties of optimum schedules and of the presented topologies. Among others, the advantages of using variable transmission powers instead of fixed transmission powers, the differences between graph-based and SINR-based models of interference, the length of optimal schedules depending on sender density, and the quality of the upper and lower bounds on schedule lengths, which are produced by the heuristics, are examined. The performance of the ILPs and CPs is analyzed empirically in various ways and both approaches are compared with each other. Moreover, the benefit of using random restarts during the solution of the constraint programs is analyzed. Finally, the topologies presented in this thesis are compared to each other using numerous quality measures.

Chapter 7: The most important results of this thesis are resumed and some conclusions are given. In particular, constraint programming is compared to mathematical programming based on the results achieved during the experiments. This chapter concludes with a brief outlook.

Chapter 8: The last chapter of this thesis gives a summary of the main achievements of this thesis in German. Such a summary is obligatory for a diploma thesis, which is not written in German, to be accepted at University of Karlsruhe.

2 Preliminaries

In this chapter we will introduce the basic notions and main concepts that will be used in the remainder of this thesis. Section 2.1 recalls some common terminology in connection with graphs and networks. Sections 2.2 and 2.3 deal with mathematical programming and constraint programming, respectively. Both are optimization techniques that are suitable for various kinds of problems and allow the user to define a problem in a formal way. The problem can subsequently be solved using highly optimized solvers. Finally, in Section 2.4, the freely available constraint programming solver Gecode is introduced in some detail.

2.1 Graphs and Networks

Definition 2.1 (Graph, Digraph)**.** An *undirected graph* or *graph* is a pair $G = (V, E)$, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of *unordered* pairs of vertices, called *edges*. An edge between $u \in V$ and $v \in V$ is denoted by $\{u, v\}$. Two vertices that are connected by an edge are called *adjacent* or *neighbors*. In contrast to an *undirected graph*, a *directed graph* or *digraph* $G = (V, A)$ consists of a finite set of vertices V and a set $A \subseteq V \times V$ of *ordered* pairs of vertices, called *arcs*. An arc from $u \in V$ to $v \in V$ is denoted by (u, v).

Definition 2.2 (Path)**.** A *path* P in a graph $G = (V, E)$ is a sequence $(v_1, v_2, \ldots, v_k)$ of distinct vertices $v_1, \ldots, v_k \in V$ such that $\{v_i, v_{i+1}\} \in E$ for $1 \leq i < k$. Similarly, a *directed path* in a digraph $G = (V, A)$ is a sequence $(v_1, v_2, \ldots, v_k)$ of distinct vertices $v_1, \ldots, v_k \in V$ such that $(v_i, v_{i+1}) \in A$ for $1 \leq i < k$. The number of edges of a path is called its *length*.

In the following, all definitions are related to *undirected* graphs if not stated otherwise. The notions for *directed* graphs are defined analogously and can be found in the standard literature on graph theory, for example in [6].

Definition 2.3 (Induced Subgraph)**.** An *induced subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' = \{\{u, v\} | u \in V', v \in V', \{u, v\} \in E\}$.

Definition 2.4 (Independence)**.** Pairwise non-adjacent vertices or edges are called *independent*. A set of vertices or edges is called *independent* if no two of its elements are adjacent.

Definition 2.5 (Neighbor, Vertex Degree)**.** The set of *neighbors* of a vertex v in a graph $G = (V, E)$ is denoted by $N_G(v)$, or briefly by $N(v)$. The *degree* $d_G(v) = d(v)$ of a vertex v is the number $|E(v)|$ of edges at v. For undirected graphs this is equal to the number of neighbors of v. The number $d(G) := \sum_{v\in V} d(v)/|V| = 2|E|/|V|$ is called the *average degree* of G.

Definition 2.6 (Connectivity)**.** We call a non-empty graph G *connected* if any two of its vertices are linked by a path in G. G is called *k-connected* (for $k \in \mathcal{N}$) if $|G| > k$ and any two of its vertices can be joined by k independent paths. A maximal connected subgraph of an undirected graph G is called a *connected component* of G.

Definition 2.7 (Symmetry)**.** A directed graph $G = (V, A)$ is *symmetrical* if each pair of nodes u, v linked by an arc (u, v) in one direction is also linked in the other direction, i.e., if $(u, v) \in A \Rightarrow (v, u) \in A$. Thus, a symmetrical digraph is equivalent to an undirected graph.

Definition 2.8 (Sparseness)**.** A graph $G = (V, E)$ is *sparse* if its average node degree is bounded by a small constant. This is equivalent to the condition that the number of edges of G is about linear in the number of vertices. For a graph to be sparse, it is not required that the maximum degree of a single node is bounded.

Definition 2.9 (Weighted Graph)**.** In many situations it is useful to assign to each edge $e = \{u, v\} \in E$ a *weight* $w(e)$. For example, in the context of sensor networks this weight could be the distance between sensor nodes u and v or the power required for a transmission between u and v. Such a graph G with weighted edges is referred to as a *weighted graph.*

Definition 2.10 (Minimum Spanning Tree, Minimum Spanning Forest)**.** Given a connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph $G' = (V, E')$ of G which is a tree and connects all vertices from V. If G is a weighted graph, then we can assign a *weight* to each spanning tree by summing up all the weights of edges in the spanning tree. A *minimum spanning tree (MST)* is now every spanning tree with weight less than or equal to the weight of every other spanning tree. If the graph G is not connected, then every connected component has a minimum spanning tree. The union of the minimum spanning trees of all connected components is called a *minimum spanning forest (MSF).*

Definition 2.11 (t-Spanner)**.** A *t-spanner* of a weighted graph G=(V,E) is a spanning subgraph G' of G in which every two vertices are at most t times as far apart from each other on G' than on G. The number k is called *dilation.*

2.2 Mathematical Programming

In computer science, *mathematical programming* refers to techniques that allow the user to define optimization problems in a formal way and which provide the necessary tools for solving the problems using mathematical methods. The user only has to describe the problem, and not how the problem can be solved. In this work, we will use an optimization technique called *integer linear programming (ILP)* to find optimal solutions for small instances of several *NP-hard* optimization problems. In Section 2.2.1 we start with a short introduction into the closely related field of *linear programming (LP)*. Then, in Section 2.2.2, the main aspects of *integer linear programming* are described. At this point, we will confine ourselves to some basic definitions and properties of mathematical programming. For more details on the extensive theory of linear programming and the algorithms to solve LPs and ILPs there is an abundance of literature available. See for example the book on the theory of linear and integer programming by Alexander Schrijver [38].

2.2.1 Linear Programming

Linear Programming deals with the maximization or minimization of a linear *objective function* such that the solution fulfills a set of linear *constraints*. The objective function together with the constraints is called a *linear program* (LP). This can be formalized as follows:

Definition 2.12 (Linear Program)**.** Let $A \in \mathbb{R}^{m \times n}$ be a $m \times n$ matrix, $b \in \mathbb{R}^m$ a m-dimensional vector and $c \in \mathbb{R}^n$ a n-dimensional vector. The corresponding *linear program* is then given as

$$\begin{array}{ll} \textit{minimize} & c^T x \\ \textit{subject to} & Ax \leq b \end{array}$$

where $x \in \mathbb{R}^n$ is an n-dimensional vector of real variables.

Problems in which the objective function is maximized or in which the linear constraints are given as $Ax = b$ or $Ax \geq b$ can easily be transformed to this standard representation of linear programs.

There are several algorithms available for the solution of linear programs. The simplex algorithm proposed by Dantzig in 1947 was one of the first of them and, although it does not have polynomial worst-case running time, it is still frequently used, because it is extremely fast even on huge real world examples. It solves linear programs by constructing an admissible solution at a vertex of the solution polyhedron and then walking along edges of the polyhedron to vertices with successively higher values of the objective function until the optimum is reached.

For a long time it has not been known whether linear programming is solvable within polynomial time. This was resolved in 1979 by Leonid Khachiyan with the ellipsoid method [21], which had a worst-case polynomial time. The ellipsoid method itself did not find its way into industrial applications as the simplex algorithm outperformed it for all but some specially constructed problems. However, it lead to the development of the so-called interior point methods, which do not progress along points on the boundary of a polyhedral set but instead move through the interior of the feasible region. One of them is the interior point projective method proposed by N. Karmarkar in 1984 [19] which was the first polynomial-time algorithm that was efficient enough for real-world problems. Today, good implementations of simplex-based methods and interior point methods are believed to have similar performance for routine applications.

Thanks to the diverse possible applications of linear programming in industrial optimization and operations research, there are many highly optimized LP-solvers available, for instance ILOG CPLEX [5] and Xpress-MP [49]. Usually, they implement different LP algorithms and can solve problems with tens of thousands decision variables and millions of constraints.

2.2.2 Integer Programming

In many applications the decision variables are not continuous but have to be chosen from a finite set of integer values. In this case, linear programming usually is not applicable to determine the optimum solution. Instead, an extension of linear programming, the so-called *integer programming*, can be used. In an *integer (linear) program* (IP, ILP), some or all decision variables are required to be integers. Unfortunately, integer programming is NP-hard and in most cases the time to solve an integer program actually grows exponential with the number of integer variables. If some of the decision variables are allowed to be real, which is often the case, we use the notation *mixed integer programming.* A *mixed integer program* (MIP) is defined similar to the LP.

Definition 2.13 (Mixed Integer Program)**.** Let $A \in \mathbb{R}^{m \times n}$ be a $m \times n$ matrix, $b \in \mathbb{R}^m$ a m-dimensional vector and $c \in \mathbb{R}^n$ a n-dimensional vector. Let further $\mathcal{I} \subseteq \{1, \dots, n\}$. Then the corresponding *mixed integer program* is given as

$$\begin{array}{ll} \textit{minimize} & c^T x \\ \textit{subject to} & Ax \leq b \\ & x_i \in \mathbb{Z} \quad \forall i \in \mathcal{I} \end{array}$$

where $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ is an n-dimensional vector of variables.

Depending on $\mathcal{I}$ we get an ordinary LP for $\mathcal{I} = \emptyset$ and an IP if $\mathcal{I} = \{1, \dots, n\}$. As for the LP, there are many commercial and non-commercial solvers available. For the solution of the MIPs in this thesis the ILOG CPLEX solver was used.

2.3 Constraint Programming

Constraint programming (CP) is a relatively new programming paradigm which evolved within the last years especially for the solution of combinatorial problems. Similar to linear programming, CP is a declarative programming paradigm. Instead of describing how to solve the problem, the problem itself is modeled in a rather formal way and the solution of the problem is left up to some specialized CP solver. For this purpose, the solver usually provides the user with a set of predefined models and constraints, which can be extended almost arbitrarily according to the special needs of the application. Interestingly, constraints can be added, removed, or modified during the execution of the solver.

Constraint programming has proven to be useful in many fields of combinatorial optimization such as operations research, electrical engineering, molecular biology, and natural language processing. Actually, scheduling is one of the major applications of constraint programming though usually the constraints used for the scheduling problems are simpler than the non-linear constraints that evolve for scheduling in the SINR model. One of the goals of this thesis is to examine how well constraint programming is suited to deal with the rather complex SINR model.

2.3.1 Notations and Definitions

In the following we give a short overview on some notations and definitions related to constraint programming. Most definitions follow the ones used in [36].

A *constraint satisfaction problem (CSP)* consists of:

- a set of variables $X = \{x_1, \ldots, x_k\}$
- for each variable x_i, a finite set $D(x_i)$ of possible values (the *domain* of x_i)
- a finite set of *constraints* restricting the values that the variables can simultaneously take

The domain $D(x)$ of a variable x is often, but not necessarily, a set of integers or an enumerated set of values. Another possibility are *set variables* whose values are sets.

A *constraint* C on X is a subset of the Cartesian product of the domains of the variables in X, i. e., $C \subseteq D(x_1) \times \cdots \times D(x_k)$. We call each tuple $(d_1, \ldots, d_k) \in C$ a *solution* to C. Each solution assigns the value d_i to the variable x_i, for all $1 \leq i \leq k$. We also say that the assignment of a solution *satisfies* C. If $C = \emptyset$, then there exists no assignment that fulfills C and we say that C is *inconsistent*. There are two ways to specify a constraint: *intentionally*, e. g. $x_1 \neq x_2$, or *extensionally* as a set of allowed tuples of values, e.g. $\{(1,2),(2,1)\}$. Today, constraints are almost always expressed intentionally.

A *solution to a CSP* is an assignment of a value $d \in D(x)$ to each $x \in X$, such that all constraints are satisfied simultaneously. Possible goals of solving a CSP are to decide whether a CSP has a solution or not, to find some solution of the CSP, or to enumerate all solutions. Often it is not enough to find just *some* solution, but we want to find the *optimum solution* to a CSP with respect to certain criteria. This leads us to the constraint optimization problem.

A *constraint optimization problem (COP)* consists of:

- a CSP P with variables $x_1, \ldots, x_k$
- an *objective function* $f : D(x_1) \times \cdots \times D(x_k) \to \mathbb{Q}$

A solution d to P that minimizes (maximizes) the value of $f(d)$ is called an *optimum solution* to the minimization (maximization) COP.

There are many ready-to-use constraint programming libraries available for Java and C++, many of which are free. Usually, they solve the CSPs and COPs by a combination of:

- systematic search through the space of possible variable assignments (using backtracking)
- constraint propagation (using the constraints to derive new information about the problem)
- stochastic local search
- linear programming

For our implementations and the experiments we used the freely available CP solver Gecode. In the following section, we will describe the main capabilities of Gecode.

2.4 Gecode: Generic Constraint Development Environment

Gecode [7] is a free open source CP solver. The first stable version of Gecode was released in the end of 2005 and since then, there have been many improvements and new versions. The latest updates have been within the last months, so it seems that the Gecode project is still alive and likely to be continued.

Besides being free, there are many reasons why to choose Gecode over other CP solvers. First of all, it is very portable. It is written in standard compliant C++ and runs on a wide range of hardware (32bit and 64bit) and operating systems (e.g., Unix/Linux, MacOS X, Windows). Extensive reference documentation is available and there are also several small examples for famous optimization problems available, which help to get started with Gecode. Unfortunately, the more complex programming tasks are not documented as well, but most questions can be answered with a look into the freely available source code of Gecode. Considering the performance, Gecode seems to be competitive even with commercial state-of-the-art solvers like *ILOG CP Solver* [8]. Last but not least, Gecode can be easy extended with new propagators (as implementations of constraints), variable domains, branching strategies, and search engines. In the following, we will give a short introduction into these different components of a Gecode CP formulation.

2.4.1 Models and Constraints

Gecode offers two main models for the formulation of constraint programs, *finite domain integers* and *finite integer sets*. Finite domain integers are more common and all our CPs are based on them. Therefore, we confine ourselves to finite domain integers in this short introduction.

The constraint satisfaction problem (CSP) is given as a set of variables and a set of constraints. The general procedure to solve the CP is as follows: In the beginning, the domain $D(x) \subset \mathbb{Z}$ of each variable x is a finite set of integers. The variables are, one after another, assigned values from their domains. After each assignment, the constraints are used to thin out the domains of the not yet assigned variables as much as possible. If, at some point, the domain of a variable gets empty, then we immediately know that the current assignment does not yield a feasible solution. In this case, a backtrack step is performed and the variable that has been assigned most recently will be assigned the next value from its domain. If all variables could be assigned a value, then a valid solution is found. The details of this search approach will be explained in the following sections. Let us first have a look on some of the constraints for finite domain integers that are already provided by Gecode.

Domain constraints: The domain of a variable or a set of variables can be defined by domain constraints. They allow to give lower and upper bounds on the variable or to define the domain as a set of integers. They also can be used to check if the value of a variable is an element of a given set of integers and then, to assign the result to some Boolean variable.

Relation constraints: They can be used to enforce arbitrary relations $(>, \geq, <, \leq, =, \neq)$ between a variable and a constant or between two variables. They also can be used to check if a relation is satisfied and then, to assign the result to some Boolean variable.

Distinct constraints: Distinct constraints can be used to enforce that all variables of a given set are distinct. Furthermore, it is possible to give offsets to the variables so that the variables plus the corresponding offsets have to be distinct.

Channel constraints: Given two lists of integers $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ it can for example be enforced that $x_i = j \leftrightarrow y_j = i$ for all $1 \leq i \leq n$.

Graph constraints: One example is the *circuit constraint*. Given a set x of integers, the graph with edges $i \rightarrow j$ where $x_i = j$ must have a single cycle covering all nodes and thus form a circuit.

Scheduling constraints: Some constraints for scheduling problems with a set of machines and a set of tasks that have to be assigned to the machines are given. The tasks can have start and end dates as well as durations and resource requirements. The machines can have certain amounts of resources available.

Sorted constraints: Given two lists of integers $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ it can be enforced that x and y are equally reordered such that x is sorted in increasing order.

Cardinality constraints: There are several *count constraints* available, which for example allow to specify or restrict the number m of variables in a set x of variables that are assigned a given value n. Instead of n one can also use the value of another variable y. Furthermore, it is possible to count the number of variables in the set x which are equal to n or y and write the result to some variable z.

Arithmetic constraints: They can be used to apply arithmetic functions on the variables, for instance to determine the minimum or maximum of a set of variables, or to compute absolute values, products, roots, etc.

Linear constraints: They can be used to determine the (weighted) sum of a list of variables or to enforce some relation $(>, \leq, \ldots)$ between this sum and a given constant or variable.

It is possible to extend Gecode with new constraints by implementing own propagators. The concept of propagators is introduced in the next section.

2.4.2 Propagators and Filtering

In Gecode constraints are implemented as *propagators*. Every time when a variable x is assigned or the domain of x is modified, the propagators of those constraints that are affected by x are executed. The propagator now checks if the smaller domain of x helps to thin out the domains of other variables, meaning that values from other domains can be removed without changing the set of solutions. This process is called *filtering*. If the propagator is able to filter other domains, then this in turn possibly triggers the execution of other propagators or another execution of the same propagator. This is repeated until the domains reach a fix point or one of the domains becomes empty.

At this constraint programming distinguishes between two kinds of filtering: *complete filtering* and *partial filtering*. The filtering is complete with respect to constraint C if removing any additional value from one of the domains would alter the set of solutions to C. This can be formalized as follows (cf. [36]):

Definition 2.14 (Domain Consistency)**.** Let C be a constraint on the variables $x_1, \ldots, x_k$ with respective domains $D(x_1), \ldots, D(x_k)$. We say that C is *domain consis-*

tent if for every $1 \leq i \leq k$ and $v \in D(x_i)$, there exists a tuple $(d_1, \ldots, d_k) \in C$ such that $d_i = v$. A CSP is domain consistent if each of its constraints is domain consistent.

Establishing domain consistency for non-binary constraints is in general NP-hard even though there are several important constraints for which complete filtering is quite efficient. In contrast to complete filtering, *partial filtering* does not necessarily eliminate all infeasible values. For example, it is often much easier to determine new upper and lower bounds on the variables than to eliminate all infeasible values. Let, for some variable x, $L(x)$ and $U(x)$ denote lower bound and upper bound of x, respectively. Given two variables x_1 and x_2 with domains $D(x_1) = \{5, \ldots, 61, 85\}$ and $D(x_2) = \{1, \ldots, 41\}$. This means $L(x_1) = 5, U(x_1) = 85, L(x_2) = 1, U(x_2) = 41$. Given the constraint $x_1 + x_2 = 100$ it is now easy to see that every valid solution has to fulfill $x_1 \geq 59$ and $x_2 \geq 15$. Thus, only by knowing $U(x_1) = 85, U(x_2) = 41$, we can immediately update $L(x_1)$ to 59 and $L(x_2)$ to 15. This gives us $D(x_1) = \{59, 60, 61, 85\}$ and $D(x_2) = \{15, \ldots, 41\}$. This process to shrink the domain intervals as much as possible without losing any solutions is called filtering with *bound consistency*. Formally:

Definition 2.15 (Bound Consistency)**.** Let C be a constraint on the variables $x_1, \ldots, x_k$ with respective lower and upper bounds $L(x_1), U(x_1), \ldots, L(x_k), U(x_k)$. We say that C is bound consistent if for every $1 \leq i \leq k$, there exists a tuple $(d_1, \ldots, d_k) \in C$ such that $d_i = L(x_i)$ and there exits a tuple $(e_1, \ldots, e_k) \in C$ such that $e_i = U(x_i)$

In this example filtering with domain consistency would have given the more accurate domain $D(x_2) = \{15, 39, 40, 41\}$. However, in every case where complete filtering is too costly or even intractable, partial filtering is preferable.

One remarkable property of constraint programming is the possibility to add or alter propagators during the search process. If, for instance, all but two variables of an n-ary constraint are fixed, then it is sometimes advantageous to replace the propagator by a similar binary propagator that is more efficient. Another application of this feature will be explained in the next section in the context of constraint optimization problems.

2.4.3 Search Engines for CSPs and COPs

Gecode offers several *search engines* which roughly control the search process. The most common search engine for CSPs is a simple *depth-first search*. The variables are assigned one after another. After each assignment filtering takes place. If the domain of a variable becomes empty a backtrack step is performed and the next possible assignment is checked out. This is repeated until all variables have been successfully assigned a value or the domain of the first variable in the search tree is empty.

One problem with the depth-first search is, that if we do a bad assignment for one of the first variables in the search tree, then we have to visit all nodes of the possibly huge subtree before the bad decision can be revised. This can possibly be avoided by using *limited discrepancy search* (LDS). LDS works best if it is used in combination with a good heuristic that determines the order in which a variable is assigned the values from its domain. The idea is that, given the heuristic performs well, usually only a few variables in the search tree are assigned bad values. LDS systematically searches all paths that differ from the heuristic path in at most a small number of decision points, or discrepancies. Starting with discrepancy 0, the nodes of the search tree are searched in increasing order of discrepancies. Further information on this topic can be found in [14].

In Gecode constraint optimization problems are solved using a *branch-and-bound search*. Initially, a simple depth-first search is performed to find a valid solution $(d_1, \ldots, d_k)$ to the underlying constraint satisfaction problem. The corresponding value $z = f(d_1, \ldots, d_k)$ of the objective function gives an upper bound on the solution. Now, the possibility of constraint programming to add new constraints during runtime is used to enforce $f(x_1, \ldots, x_k) < z$. This, of course, invalidates the current solution of the CSP. A backtracking step is performed and the depth-first search continues with the additional constraint. If the new CSP is unsatisfiable, then $(d_1, \ldots, d_k)$ was an optimal solution with objective value z and we are done. If, on the other hand, the new CSP has a solution, then we get a new upper bound and repeat the process until we reach an unsatisfiable CSP, which proves the optimality of the last valid solution. Of course it is also possible to define some timeout and thus get the best solution that can be found within the given time.

Similar to the branch-and-bound search is the *depth-first restart best solution search*. The only difference is that every time after an improved solution with objective value z was found and the constraint $f(x_1, \ldots, x_k) < z$ was added the depth-first search starts with the additional constraint from the beginning instead of extending the current solution.

2.4.4 Branchings

One crucial point that heavily influences the performance of the search is the order in which the variables are visited in the search tree, and the order in which the values from the domains are assigned to the corresponding variables. Both is realized in Gecode with so-called *branchings*. There is an abundance of standard branchings available in Gecode.

The next variable to be assigned in the search tree can be chosen to be, among others: the first unassigned variable, the variable with smallest or largest domain minimum, the variable with smallest or largest domain maximum, the variable with smallest or largest domain size, or the variable with minimum or maximum number of dependent propagators. In many situations it is a good choice to select the variable with smallest domain size. This minimizes the probability of a bad assignment and possibly it can be shown relatively quick that there does not exist a feasible assignment for the variable.

If the next variable in the search tree is chosen it has to be decided, which value to select first for branching. For this purpose, the following options are available: the smallest value, the median value, the maximum value, the lower half of the domain, and the upper half of the domain.

There often exist good heuristics to decide which variable to assign next or which of the possible values to select next. Gecode offers the possibility to implement such heuristics in custom branchings and to use them in the CP. This usually pays off with improved performance, therefore one should consider to implement an own branching if the standard branchings are not sufficient.

2.4.5 Performance Measures

The most common performance measure for CPs is the number of failures, what is equivalent to the number of backtracks. This is because the number of failures is independent of the underlying hardware and it gives a good indication on how good the filtering and the branchings work. Thus, the failures allow for a good comparison between different CPs. The number of propagations is also of interest. The less often the propagators have to be executed the better. However, the number of propagations does not tell much about the overall running time as different propagators might have different running times. Unfortunately, both measures are inapplicable to compare the performance of a CP with completely different approaches like integer programming. For this purpose, using the running time of the CP seems to be the best choice. Depending on the application the memory consumption might also be important. In our experiments, Gecode usually did not use more than some negligible megabytes of memory, therefore we will not consider this measure in the following.

2.4.6 Extending Gecode

Gecode offers enough functionality to model most combinatorial problems using only the available propagators, search engines, and branchings. If, however, some necessary functionality is unavailable, then Gecode can be extended almost at will. Besides propagators, search engines, and branchings, even new types of variables can be implemented. If performance is substantial, then the implementation of good branching heuristics is often worthwhile. Unfortunately, while using the basic functionality of Gecode is quite straightforward, extending Gecode is more complicated. A good understanding of how Gecode works is necessary and as there is not much documentation available on how to extend Gecode, one has to learn this task by reading and understanding the source code of Gecode. Fortunately, Gecode's source code is very well structured so that it usually is possible to find the necessary code passages within reasonable time. Thanks to the extendibility of Gecode it should be possible to model almost all combinatorial problems. This is a great advantage over mathematical programming approaches. And in contrast to an integer program, a CP usually gets faster the more constraints are used.

2.4.7 C++ Interface vs. Java Interface

Gecode is written in C++ and originally designed to be a C++ library for constraint programming. However, there exists a Java interface, Gecode/J, which makes Gecode available in Java programs and provides most of the functionality of Gecode. As most of our implementations were done in Java, we first used Gecode/J in our Java framework. Gecode/J was easy to learn and convenient to use. Unfortunately, we soon realized that Gecode/J was not as developed as the C++ implementation of Gecode and in particular not optimized for performance. As we intended to compare constraint programming with integer linear programming with respect to performance, we decided to use the optimized C++ interface. Nevertheless, if performance is not so crucial and one is only interested in the number of failures and propagations, then Gecode/J is worth to be considered.

3 Survey on Wireless Sensor Networks

The success or failure of transmissions in wireless networks depends on a big number of factors. This chapter deals with some of the models that have been proposed to make wireless sensor networks tractable for theoretical considerations. The reader will be provided with the background knowledge about wireless sensor networks that is necessary in the context of this thesis. The SINR interference model will be introduced and motivated by comparison with other models of interference. Furthermore, the problems covered in this thesis, namely the scheduling problem and the topology control problem, are defined.

3.1 Communication Graphs

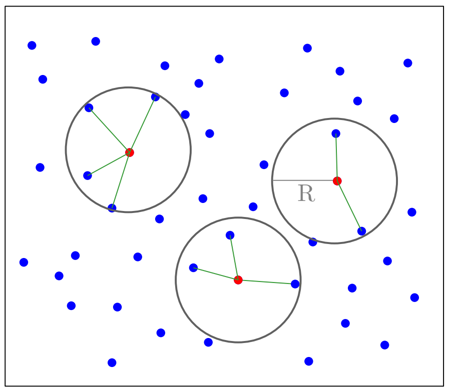
It seems very natural to model sensor networks using graphs. Every sensor node corresponds to a vertex of the graph and every pair of sensor nodes that is theoretically able to communicate directly is connected by an edge. The resulting graph is the *communication graph* of the sensor network.

Definition 3.1 (Communication Graph)**.** Given a sensor network and a graph $G = (V, E)$ with V representing the set of sensor nodes. If there is an edge $e_{uv} = \{u, v\} \in E$ if and only if the sensor nodes corresponding to u and v are able to communicate directly with each other, then we call G the *communication graph* of the sensor network

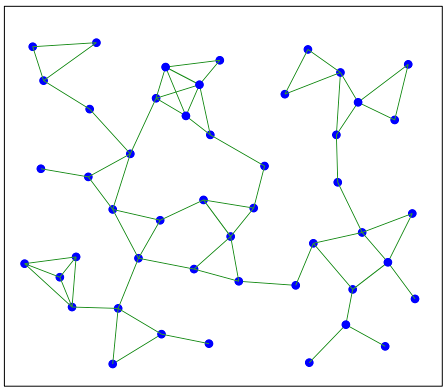
Usually, it can be assumed that the transmission power of a sensor node is restricted and that this also restricts the maximum transmission range R of the sensor. In many applications all sensor nodes furthermore share the same maximum transmission range R. This means that in the communication graph there is an edge between two nodes u and v if and only if the distance $\operatorname{dist}(u, v)$ between u and v is less than or equal to R. In this case the distances are usually normalized such that $R \equiv 1$ and the communication graph is a *unit disk graph*.

Definition 3.2 (Unit Disk Graph)**.** A graph $G = (V, E)$ is a *unit disk graph (UDG)* if and only if there is an embedding of the nodes in the plane such that there exists an edge $\{u, v\}$ between node u and node v if and only if the Euclidean distance between u and v is less than or equal to 1.

Figure 3.1(a) shows a set of sensor nodes in the plane with three exemplary transmission ranges of radius R. Next to it, the corresponding unit disk graph is depicted.



(a) Nodes which are within transmission range are connected



(b) Resulting Unit Disk Graph

Figure 3.1: Illustration of the Unit Disk Graph model

However, in reality things are a little more complicated. For example, there often are obstacles such as trees or walls in the line-of-sight between sensor nodes and absorb their radio waves. Therefore, considering only the distance between two nodes is not sufficient to decide whether a direct communication is possible. This leads to the somewhat more refined model known as *quasi unit disk graph*. Like in the unit disk graph there is no communication possible if the distance between two senders is greater than the maximum transmission range R. Moreover, if the distance between sender and receiver is less than or equal to some r with $r < R$, then it is assumed that the transmission is possible by all means. In the case that the distance is between r and R a communication could be possible but does not have to be. Introducing a parameter $d = r/R$ we get the so-called *d-quasi unit disk graph* model.

Definition 3.3 (d-Quasi Unit Disk Graph)**.** Given some parameter $d \in [0, 1]$, a graph $G = (V, E)$ is a d-quasi unit disk graph (d-QUDG) if and only if there is an embedding of the nodes in the plane such that dist(u,v) $\leq d \Rightarrow \{u, v\} \in E$ and dist$(u, v) \geq 1 \Rightarrow \{u, v\} \notin E$.

As long as we are in situations without concurrent transmissions there is nothing wrong about the aforementioned communication graphs. On the contrary, if we know that the communication graph is an unit disk graph, then this gives us additional information which can be used to design efficient algorithms. It can for instance be shown that every unit disk graph in which two nodes are never closer than some constant distance has bounded degree. This assumption that there is some minimum distance between any two nodes is referred to as the $\Omega(1)-$*Model* and has proven useful in a variety of cases [23].

Unfortunately, things are getting much more complicated when there is more than one active sender involved at the same time. In this case, concurrent transmissions will interfere with each other and that can result in failure of some or all of the transmissions.

3.2 How to deal with Interference

Let us have a quick look on the possibilities to deal with interference and transmission failures before we get deeper into the topic of interference modeling. In the OSI model, it is the task of the *Medium Access Layer* (MAC Layer) to manage the access to the shared medium in wireless networks. Therefore, the MAC Layer has to take care that no transmissions get lost due to interference. This is done by either retransmitting unsuccessful transmissions or by scheduling all transmissions such that the interference does not lead to transmission failures. Several *MAC Protocols* have been proposed to achieve this goal:

ALOHA: The most basic and one of the first protocols was *ALOHA* [35]. Every packet is send as soon as it is generated. Received packets are acknowledged by the receiver with an acknowledgement message. If the sender does not get the acknowledgment, he just tries to resend the message after a random period. However, this approach usually results in a very poor network usage.

Carrier Sense Multiple Access (CSMA): In contrast to the ALOHA protocol, a sender using the CSMA method [44] first checks if other nodes are transmitting before it starts a transmission. If the channel is clear, it starts immediately to transmit. If not, the sender waits for all active transmissions to finish and starts to transmit after some random back-off time in order to avoid starting its transmission concurrently with other waiting nodes.

CSMA with Collision Avoidance (CSMA/CA): In networks that use the CSMA protocol it can happen that two senders s_1 and s_2 both want to send to the same receiver r but are unable to hear each other. Thus, they send at the same time and both transmissions fail due to the interference at r. This problem is known as the *hidden terminal problem*. The CSMA/CA protocol was developed with three-way-handshake in order to avoid the hidden terminal problem. Here, the sender s first sends a *Request-To-Send* (RTS) message. If the addressed receiver r is willing to accept the transmission, it answers with a *Clear-To-Send* message. This CTS message can be received by all other senders in transmission range of r and prevents them from communicating with r. As soon as sender s receives the CTS message, it starts with the transmission. Finally, when the transmission is finished, r informs the neighboring nodes that the medium is idle again using an *acknowledgment message* (ACK). The CSMA/CA protocol is used in most of today's wireless networks. The most famous example is the IEEE 802.11 wireless LAN protocol family [33].

Time Division Multiple Access (TDMA): TDMA protocols are based on an idea completely different from the aforementioned contention based protocols: Instead of transmitting in a trial-and-error-fashion, the nodes agree in advance on a schedule. This schedule assigns to every node a time slot such that the node can send freely without interfering with concurrent transmissions. At this, only such nodes are allowed to share the same time slot that are so far from each other that the caused interference can be ignored. This approach helps in several ways to conserve energy: nodes can go to an energy-saving sleep mode between their assigned time slots, there is less contention-introduced overhead, nodes can transmit with less transmission power as they can estimate the occurring interference, and theoretically there are no collisions which result in energy-consuming retransmissions. However, TDMA based protocols usually do not allow to dynamically change frame lengths and time slot assignments. Therefore, they usually do not scale as good as contention based protocols. Nevertheless, if durability is of greater importance than network performance (that probably is true in many applications of wireless sensor networks), then TDMA based methods seem to be better suited than their contention based rivals. Some proposals for TDMA algorithms can be found in [15, 32].

The methods and algorithms described in this thesis are completely aimed at the TDMA method: the scheduling algorithms help to determine good schedules and to research the basic properties of optimum schedules, and the goal of the proposed topology control algorithm is to find a subset of the communication links such that the connectivity of the network is preserved and the number of slots necessary to schedule all selected links in an TDMA fashion is minimized.

3.3 Scheduling Problem

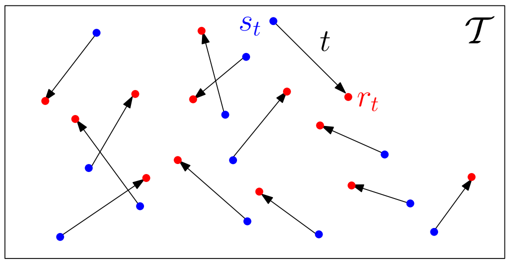
The basic idea of scheduling is to assign all requests from a multiset $\mathcal{T}$ of transmission requests to different time slots such that all transmissions that are assigned to the same time slot can be active concurrently without running into danger of failures. At this, a *transmission request* $t = (s, r)$ consists of a sender s and a receiver r. For convenience, we will refer to transmission requests simply as *transmissions* in the following. The same sender-receiver-pair (s, t) can be contained in $\mathcal{T}$ multiple times, representing different transmissions.

Additionally, every transmission $t \in \mathcal{T}$ has assigned a transmission power P_t. We distinguish between scheduling with *fixed transmission powers* where all transmissions share the same transmission power $P_{\max}$, and scheduling with *variable transmission powers* where every transmission can have a distinct transmission power $P_t \in [0, P_{\max}]$. At this, the transmission power of a single sender can vary between different transmissions. The process of determining good transmission powers is called *power control* and in most cases power control is part of the scheduling algorithm.

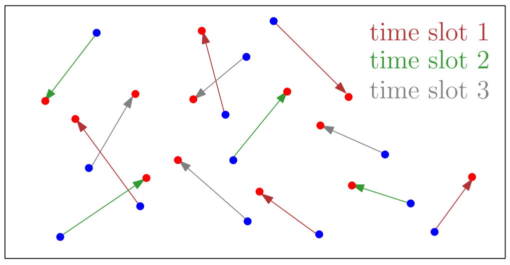
We say that a set $T \subseteq \mathcal{T}$ of transmissions with associated transmission powers P_t for all $t \in T$ is *valid* with respect to some interference model (cf. Section 3.4) if the model allows to schedule all transmissions from T concurrently without failures. A sequence $\mathcal{S} = T_1, T_2, \ldots, T_k$ of transmission sets with associated transmission powers and $\bigcup_{i=1}^{k} T_i = \mathcal{T}$ is a *schedule* of $\mathcal{T}$. A schedule $\mathcal{S}$ is *valid* if every transmission set T_i of the schedule is valid. Single transmission sets of a schedule are allowed to be empty (an empty transmission set is always valid) and we also refer to the transmission sets of a schedule $\mathcal{S}$ as *time slots* or simply *slots*. The *length* or *makespan* of a schedule $\mathcal{S}$ is the number of slots in $\mathcal{S}$. The *scheduling problem* can now be formalized as follows:

Definition 3.4 (Scheduling problem with fixed transmission powers)**.** Given a set $\mathcal{T}$ of transmission requests and common transmission power P. Find a schedule $\mathcal{S}$ of $\mathcal{T}$ such that $\mathcal{S}$ is valid and has minimum length among all valid schedules of $\mathcal{T}$.

Definition 3.5 (Scheduling problem with variable transmission powers)**.** Given a set $\mathcal{T}$ of transmission requests and maximum transmission power $P_{\max}$. Compute transmission powers $P_t \in [0, P_{\max}]$ for all transmissions $t \in \mathcal{T}$ and a schedule $\mathcal{S}$ of $\mathcal{T}$ such that $\mathcal{S}$ is valid and has minimum length among all valid schedules of $\mathcal{T}$ with every possible power assignment.



Input: Set $\mathcal{T}$ of transmissions $t = (s_t, r_t)$



Output: Assignment of transmissions to time slots

Figure 3.2: Input and output of the scheduling problem

The situation for fixed transmission powers is depicted in Figure 3.2. The input consists of a set $\mathcal{T}$ of transmissions and some common transmission power P. The output of the scheduling algorithm is an assignment of the transmissions to three different slots (which is equivalent to a partition of $\mathcal{T}$ into three valid transmission sets T_1, T_2, T_3).

3.4 Interference Models

Now let us have a closer look on how interference can be modeled and how we can decide if a set of transmissions is valid. Every transmission $t = (s, r)$ involves a *sender*

s and a *receiver* r. The sender transmits the information in form of radio waves, which are usually emitted by some kind of omnidirectional antenna. If there is a concurrent transmission $t' = (s', r')$, then the radio waves emitted by s' interfere with the radio waves from s at receiver r. Thus, the transmission quality of t is affected negatively. In the worst case this means that none of the concurrent transmission attempts is successful. There have been many proposals how to model this interference. Especially in theoretical computer science, people often use interference models that are directly based on communication graphs like the unit disk graph. In the following, we will introduce such graph-based models as well as the physically motivated SINR model on which this thesis is based. Finally, both types of models are compared against each other and the use of the SINR model is motivated.

3.4.1 SINR Model

We start with the description of the so-called SINR model which is used in this thesis. The SINR model is physically motivated and is widely believed to resample reality very closely. Its introduction into algorithms for wireless networks is addressed to [13]. There are several definitions of the SINR model in use, which mainly differ in granularity. For example, some definitions allow to model obstacles and random influences on the transmission quality. In the context of this thesis those random influences are undesired as the goal is an unbiased study of basic properties of wireless networks. Therefore, we confine ourselves at this point with the most basic model in which the distance between sensor nodes is the only defining factor for signal strength. This model is also known as the *geometric SINR model* (or short, $SINR_G$). A more refined model can, for instance, be found in [20]. Note, however, that all proposed algorithms and heuristics can be used with more complicated SINR models without any modifications.

Given a wireless network with n nodes $v_1, \ldots, v_n$. Assume node v_i starts a transmission to node v_j using transmission power $P_i > 0$. The signal from v_i will reach the receiver v_j with *signal strength* S_{ij}:

$$S_{ij} = \frac{P_i}{d(v_i, v_j)^\alpha} \tag{3.1}$$

At this, $d(v_i, v_j)$ is the distance between sender v_i and receiver v_j and α is the so called *path loss exponent*. It is usually assumed that α is a position-independent constant with a known value between 2 and 5. In this thesis, $\alpha = 4$ is used. However, the value of α does not have significant impact on the results of this thesis.

In general, v_i does not have to be the only active sender. Let $\mathcal{U} \subseteq \{v_1, \ldots, v_n\}$ be the set of all senders that are active concurrently with v_i. The radio waves emitted by every concurrent sender v_u also reach our receiver v_j with signal strength S_{uj} and interfere with the signal from v_i. This leads to the following interference I_j at receiver v_j:

$$I_j = \sum_{v_u \in \mathcal{U}} P_u d(v_u, v_j)^{-\alpha} = \sum_{v_u \in \mathcal{U}} S_{uj} \tag{3.2}$$

Finally, the model takes into account some *background noise* $\eta_j > 0$ at receiver v_j. This background noise combines all interfering influences at v_j that are not caused by concurrent transmissions of the considered network nodes. Usually it is assumed that all nodes share the same background noise η.

So how can we decide if transmission $t = (v_i, v_j)$ is successful? The necessary condition is that the ratio of transmission signal strength to the sum of all interfering influences exceeds some constant κ. This leads to the *SINR inequality* or *SINR constraint*:

$$\gamma_{ij} := \frac{S_{ij}}{\sum_{v_u \in U} P_u d(v_u, v_j)^{-\alpha} + \eta} = \frac{S_{ij}}{I_j + \eta} > \kappa \tag{3.3}$$

Here, γ_{ij} is the so-called *Signal to Interference and Noise Ratio (SINR)* from which the SINR model got its name. The constant κ is the *minimum SINR* that is necessary for v_j to successfully decode a message. κ depends on hardware and software characteristics of the receiving node and can in general be different for every node. For modern hardware κ lies somewhere between 5 and 15. For the sake of simplicity it will be assumed in the following that all network nodes share the same minimum SINR $\kappa = 10$.

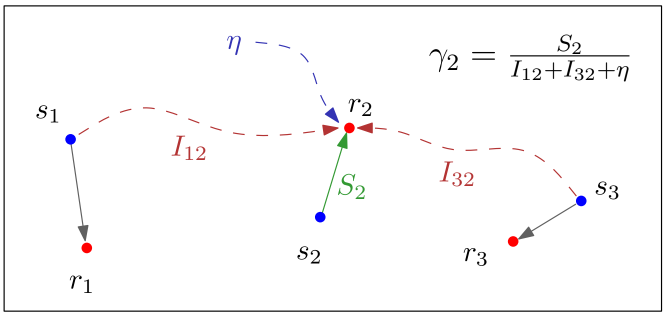


Figure 3.3: Illustration of the signal-to-interference-plus-noise-ratio (SINR)

The situation is illustrated in Figure 3.3. There are three concurrent transmissions: $t_1 = (s_1, r_1)$, $t_2 = (s_2, r_2)$, and $t_3 = (s_3, r_3)$. We will have a closer look at transmission t_2. The radio waves emitted by s_2 are received at r_2 with signal strength S_2. Moreover, the radio waves emitted from s_1 and s_3, though not designated for r_2, will reach r_2 with signal strengths I_{12} and I_{32} and interfere with the signal from s_2. Additionally, the omnipresent background noise η is interfering with t_2. Now, transmission t_2 is successful if and only if SINR γ_2 exceeds κ, the lower bound on the SINR for a successful transmission.

3.4.2 Conflict Graphs and Graph-based Interference Models

In this section we will see how the scheduling problem can be reduced to coloring problems in graphs. There are two possibilities how to assign time slots: they can be assigned to nodes, or, alternatively, to transmissions (which corresponds to edges in the communication graph). Assigning the time slots to transmissions offers several advantages. It can, for instance, be considered that communications over smaller distances result in better signal quality and that thus more interference can be tolerated. For this reason, let us concentrate on the case that time slots are assigned to transmissions. A frequently used approach is to use a so-called conflict graph, which defines the transmission pairs that cannot be active concurrently.

Definition 3.6 (Conflict graph)**.** Let $G = (V, E)$ be the communication graph of a wireless network. The *conflict graph* of G is the graph $\mathcal{C} = (E, C)$ which has a node $e \in E$ for every edge of the communication graph G, and an edge $\{e, f\} \in C$ for every pair of transmissions $e, f \in E$ if and only if transmission e and transmission f cannot be scheduled successfully at the same time.

Conflict graphs can be defined on arbitrary sets of transmissions and not only on communication graphs. Such an example of a conflict graph can be seen in Figure 3.4. The

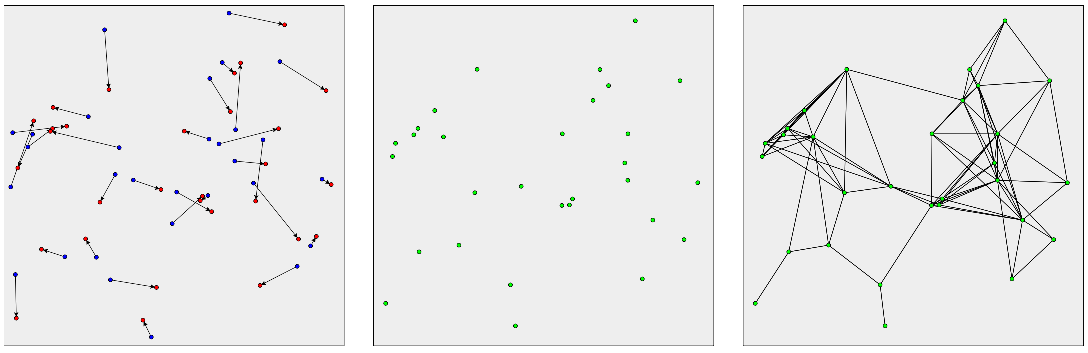

(a) Transmission set (b) Conflict graph nodes (c) SINR conflict graph (fixed p.)

Figure 3.4: Transmission set and corresponding SINR conflict graph

left picture shows the input, a set of 30 randomly placed transmissions. In the middle picture, the nodes of the conflict graph are shown. Every transmission of the input set is replaced by one conflict graph node. The picture to the right shows one possible conflict graph, the conflict graph defined by the SINR model with fixed transmission powers. We will see later, how this conflict graph is defined.

Now, the basic idea of graph-based scheduling is that every transmission set T that is an independent set in $\mathcal{C} = (E, C)$ can be scheduled simultaneously. Thus, a coloring of the conflict graph $\mathcal{C}$ defines a valid schedule. A time-optimal schedule can then be computed as a coloring with minimum number of colors. Unfortunately, this concept oversimplifies reality. The main problems with graph-based methods are shown in Section 3.4.3.

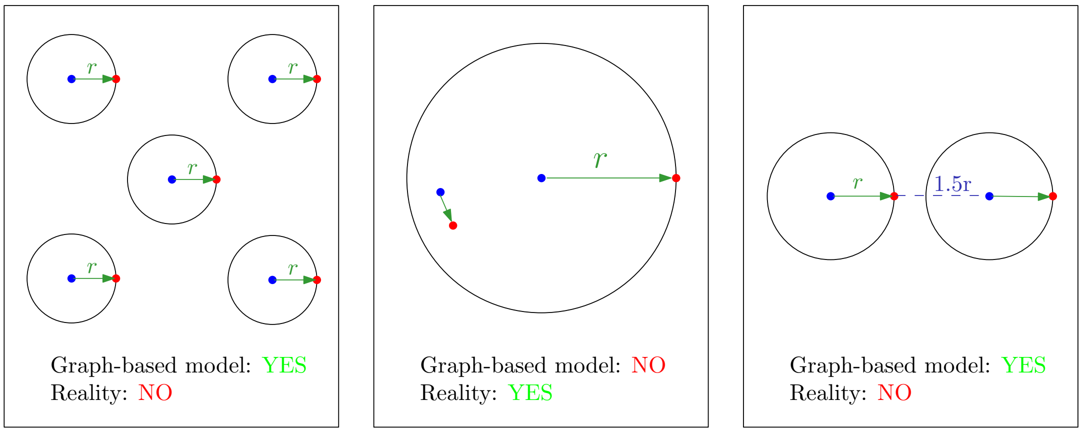
Theoretically, the edges of a conflict graph can be chosen arbitrarily. But usually, they are defined by geometric properties or neighborhood relations of the input set. For the remainder of this thesis the conflict graphs defined by the SINR constraint are of special importance.

In the following, the *SINR conflict graph (fixed power)* will be defined as the unique conflict graph whose edges are given by the pairs of transmissions that cannot be executed simultaneously in the SINR model with fixed transmission powers. Similarly, we define the *SINR conflict graph (variable power)* as the unique conflict graph whose edges are given by the pairs of transmissions that, given an arbitrary valid power assignment, cannot be executed simultaneously in the SINR model with variable transmission powers.

In Section 6.3.2, we will describe some additional conflict graph models, which are frequently used in conjunction with graph-based scheduling, and compare them experimentally with the SINR conflict graphs.

3.4.3 Graph-based Models vs. SINR Model

It is widely accepted that the SINR model reflects reality very well. Thus, it seems to be natural to evaluate the appropriateness of graph-based models by comparing them against the SINR model. Figure 3.4.3 shows three major differences between most graph-based models and reality (or the SINR model): First, interference from differ-



(a) Interference does not accumulate (b) Sender-receiver-distance is not taken into account (c) Range of interference is too limited

Figure 3.5: Three examples of why graph-based models are unrealistic

ent concurrently sending nodes sums up in reality. This for instance means that the simultaneous execution of three transmissions might be impossible although each transmission pair could be scheduled together. This is usually not considered in graph-based models. Second, most graph-based models do not take the sender-receiver-distance into account. The closer sender and corresponding receiver are positioned, the more stable is the transmission. Hence, it is not sufficient to consider only the distance between receiver and interfering senders, but one also has to take the signal strength into account. Lastly, most graph-based models underestimate the range of interference. Especially methods that work on the communication graph are unable to model interference between nodes in different connected components. In Section 6.3.2 the last two issues are examined on random transmission sets.

3.5 Topology Control

The *topology* of a network defines the structure of links connecting pairs of nodes of the network. Each communication between two nodes of the network is routed based on the *network topology*. Usually, the network topology is represented by a communication graph. The primary goal of *topology control* is to choose a connecting subset from all possible links such that the overall network performance remains good while routing on the topology is faster and easier thanks to the reduced number of links. Moreover, the restriction to energy efficient links can help to save transmission energy and thus to extend the lifetime of the sensor nodes. As pointed out by Chandrakasan *et al.* [4], this is of special importance in sensor networks because the available power is often limited.

The *topology control problem* can be described is as follows: Given a network $\mathcal{N}$ and underlying graph $G = (V, E)$ where V is the set of all nodes of $\mathcal{N}$ and E is the set of all possible links between node pairs in $\mathcal{N}$. At this, there exists an edge $(u, v) \in E$ if and only if sensor node v lies within transmission range of node u. It is often assumed that all sensor nodes have the same transmission range. In this case, the graph G is undirected. It is now the task of topology control to choose a subgraph $G' = (V, E'), E' \subseteq E$ such that the restriction to the links in E' helps to improve the performance and lifetime of the network. Usually, the resulting topology has to fulfill some minimum requirements. For example it could be required that the number of remaining links is only linear in the number of network nodes or that the length of a shortest path in G' is, for all node pairs from V, not more than twice as long as the corresponding shortest path in G. Section 3.5.1 gives a brief overview on such quality criteria for network topologies. Topology control was mentioned for the first time in [41]. A good survey on local algorithms for topology control is given in [47].

Figure 3.6 gives a first example of topology control. The left figure shows a unit disk graph with all possible communication links. The right picture depicts the resulting network after application of topology control. The resulting topology is still connected, but consists of much less links and it is obvious that especially the long-distance links have been discarded.

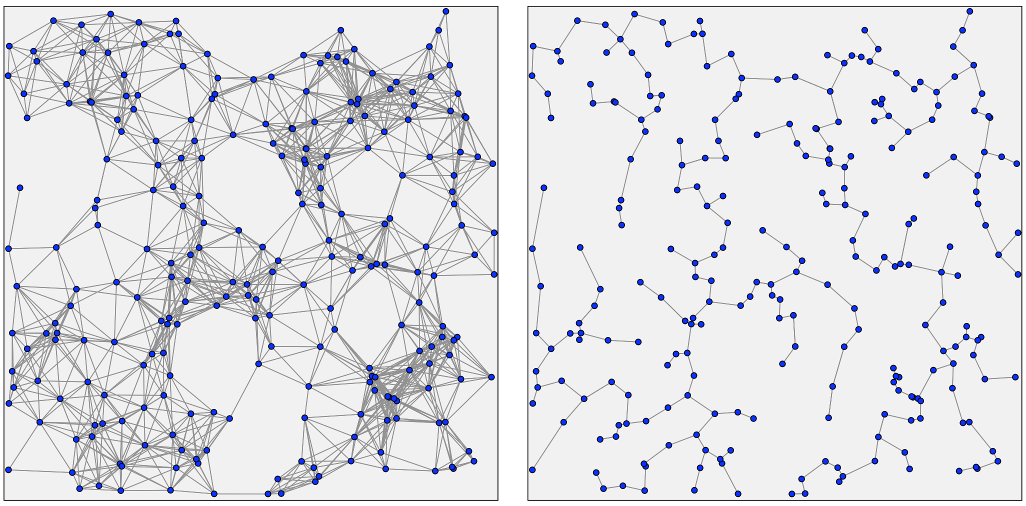

(a) Unit Disk Graph (b) Connectivity-preserving topology

Figure 3.6: Network topology before and after topology control

3.5.1 Quality Criteria for Topologies

Let $G = (V, E)$ denote the network graph before running the topology control algorithm. V is the set of sensor nodes and E the set of possible communication links. Running the topology control algorithm on G will yield a subgraph $G' = (V, E')$ of G. There have been a lot of proposals on how to measure the quality of resulting topologies and on minimum requirements that every topology control algorithm should accomplish. This section gives a short overview on possible quality criteria. A good survey on this topic can also be found in [47].

Connectivity: The most basic requirement for topology control algorithms is to preserve the *connectivity* of input graph G. This means that for every pair $\{u, v\}$ of sensor nodes that is connected by a (directed) path from u to v in G there has to be a (directed) path from u to v in G'. In addition, some topology control algorithms also guarantee k-vertex-connectivity respectively k-edge-connectivity. This means that at least k vertices respectively k edges have to be removed for the graph to become disconnected. Of course, increasing k results in better reliability of the network.

Sparseness: One of the main goals of topology control is to compute simple topologies that are easy to maintain. The complexity of a topology is strongly dependent on the number of edges. Therefore, the computed topology should contain only few edges. Preferably, the number of edges should be linear in the number of nodes, i. e., the average vertex degree should be constant. Such a topology is called *sparse*. In general there is a trade-off between network connectivity and sparseness. If nodes abandon links to too many far away-neighbors, either the network becomes partitioned or the routing paths become non-competitively long.

Spanner-property: Topology control also has some negative effects: After removing possible communication links the shortest paths between several pairs of nodes from G can become longer. Moreover, communication between some node pairs can become more power-consuming. As energy-conservation is one of the main goals of topology control, it is sometimes expected that the new topology G' is an *energy-spanner* of G. This means that every pair of nodes is connected in G' by a path p' such that sending messages along p' does not need more than a constant factor t times the energy that is needed for the nodes to communicate over an power-optimal path p in G. Of course it might also make sense to require other spanner-properties for G', such as G' being a distance-spanner (using Euclidean distances) or a hop-spanner.

Interference: It is known that higher interference at a receiver r means that any node who wants to send to r has to use higher transmission power or, even worse, that some transmissions might become infeasible due to the interference. Therefore, it is very popular among topology control algorithms to try to minimize or at least reduce interference. Several interference measures as well as algorithms to minimize the corresponding interference have been proposed. In the experimental section, we will examine how this different measures correlate with each other and how much they influence the length of a time-optimal schedule.

Symmetry: In many scheduling protocols receivers confirm the reception of data packets by acknowledgment messages (ACK). Those ACK messages have to be routed back to the sender. The simplest possibility to do so is to send the ACK message along the same path that the original message used. For this to be possible, the communication links have to be bidirectional. Therefore, it is mostly required that G' is symmetric (or undirected).

Planarity: Several efficient scheduling algorithms require that no two edges of the topology intersect. Therefore, in order to make those algorithms applicable, the computed topology has to be planar.

3.6 Related Results

In [11], Goussevskaia et al. examine the complexity of scheduling in the geometric SINR model. They show that the scheduling problem with fixed transmission powers is NP-complete by giving a polynomial-time reduction of the partition problem to the scheduling problem. Furthermore, they prove the NP-completeness of the one-shot-scheduling problem with a polynomial-time reduction of knapsack. The task of the one-shot-scheduling problem is to fill a single time slot with as many transmissions as possible from a given set of transmission requests. Finally, they give centralized approximation algorithms for the scheduling problem and the one-shot-scheduling problem with fixed transmission powers.

To this day, it is not known whether or not the problem of scheduling with variable transmission powers is NP-complete. This problem is regarded as a most fundamental problem in the field of sensor networks [28].

In [30], Moscibroda et al. examine several *scheduling complexities*. They define the scheduling problem for a network property Ψ to be the problem to find a schedule $\mathcal{S}$ of minimal length $T(\mathcal{S})$ such that the union of all successfully transmitted links satisfies property Ψ. Then, the *scheduling complexity* of a property Ψ is the minimal number of time-slos T such that there always exists a valid schedule $\mathcal{S}$ for Ψ of length $T = T(\mathcal{S})$. They show that for scheduling with fixed transmission powers the scheduling complexity is $\Omega(n)$ even for the simple property $\Psi_{\min}$ that every node should send at least one transmission successfully. Furthermore, they show that the scheduling complexity for $\Psi_{\min}$ is also $\Omega(n)$ if the transmission powers of all senders are proportional to the minimal power required for transmitting over the wireless link, meaning $P_t = \rho \cdot d(s,r)^\alpha$ for the transmission power P_t of transmission $t = (s,r)$, given some constant ρ. Both lower bounds are based on networks in which some communication links are exponentially longer than others. They also prove that the scheduling complexity of strong connectivity in wireless networks is at most $O(\log^4 n)$ and that thus non-linear power assignments are able to produce much more efficient schedules than linear power assignments or fixed power assignments in worst-case scenarios.

4 Scheduling

The first section of this chapter deals with the problem of power control: Given a set of transmissions, how can one decide whether there exists a power assignment such that all transmissions can be scheduled at the same time, and what is an optimum power assignment.

In Section 4.2, CPs and ILPs for the computation of optimum schedules are given. Of course, one cannot expect to solve big instances of the scheduling problem due to its NP-completeness. Nevertheless, several optimizations for the CPs and ILPs are discussed which significantly push up the size of solvable problems.

As there is probably no way to solve big instances of the scheduling problem exactly, Section 4.3 deals with heuristics for the scheduling problem. The idea of the heuristics is to fill the time slots one by one with as many transmissions as possible. In the case of fixed transmission powers, this is achieved by choosing transmissions that maximize the minimum signal-to-noise ratio. In the case of variable transmission powers, this is achieved by choosing transmissions that minimize the maximum transmission power. The heuristics also build the foundations for two topology control heuristics in Section 5.2.1.

Lastly, in Section 4.4, several techniques to determine lower bounds on the number of slots of an optimum schedule are discussed. Those lower bounds provide good approximations to optimal solutions. Thus, they will be used in the experimental section to give estimates on the length of optimum schedules. Furthermore, they can be used to substantially speed up the ILPs and CPs.

4.1 Power Control

Before we deal with the actual scheduling problem with many time-slots, let us have a quick look on the problem to determine if and how a set of transmissions can be scheduled to the *same* slot. If we have fixed transmission powers, everything is clear: Based on the transmission powers we are able to compute signal strength and interference and just have to check whether all transmissions have a sufficient SINR.

Unfortunately, if we allow senders to adjust their transmission powers, things get more complicated because, then we also have to choose good power levels for all senders. Let us assume that we already have found optimum power levels for a set of transmissions.

```
Input: Set $\mathcal{T}$ of transmissions, minimum SINR $\kappa$, background noise $\eta$
Output: Minimum transmission powers $P_t$ for all $t \in \mathcal{T}$

foreach $t \in \mathcal{T}$ do
    $P_t \leftarrow 0$
maxChange $\leftarrow P_{max}$
while maxChange $> \varepsilon P_{max}$ do
    maxChange $\leftarrow 0$
    foreach $t \in \mathcal{T}$ do
        oldPower $\leftarrow P_t$
        $P_t \leftarrow \kappa\ P_t/$ Sinr($t$,$\mathcal{T}$,$\eta$)
        if $P_t$ - oldPower $>$ maxChange then
            maxChange $\leftarrow P_t$ - oldPower
        if $P_t > P_{max}$ then
            foreach $t \in \mathcal{T}$ do
                $P_t \leftarrow \infty$
            break
```

Algorithm 4.1: Minimum transmission powers for a set of concurrent transmissions

If we now want to add a single new transmission t, then this possibly starts a chain reaction. The neighbors of t have to suffer from additional interference and thus need to increase their transmission powers. This in turn results in an increased interference at their neighbors and forces them to increase their transmission powers. This can go back and forth throughout the network. At the first moment one might assume that this leads to an endless process in which nodes have to increase their powers alternatingly. However, we will see soon that this process either terminates after some cycles or some node has to exceed its maximum transmission power. That, of course, would tell us that the transmissions cannot be scheduled concurrently.

This leads us to Algorithm 4.1 to determine minimum power levels for a set $\mathcal{T}$ of transmissions such that all transmission can be successfully scheduled at the same time: First, assign every sender just enough transmission power to reach its corresponding receiver if the only interference is caused by the background noise. Now that we have assigned every sender some transmission power, we get a lower bound on the interference from concurrent transmissions. Thus, in the next round, we assign every sender just enough power to reach its receiver taking account of background noise and the transmission powers of the previous round. This procedure is repeated until either the transmission power increase falls below some threshold, meaning that a fix point is reached, or some sender exceeds its maximum transmission power.

Algorithm 4.1 assumes the availability of a function *Sinr($t, \mathcal{T}, \eta$)* that computes the SINR of transmission t, given the transmission powers t' of all transmissions $t' \in \mathcal{T}$ and ambient noise η. This function depends on the details of the used SINR model and can be easily implemented with running time $O(n)$. Thus, the whole while-loop has running time $O(n^2)$. The constant ε, with $0 < \varepsilon \ll 1$, defines the minimum increment $\varepsilon P_{\max}$ that at least one power level has to get every round. By this, ε controls the precision of the solution. If fast convergence is required, and a good but not necessarily optimal power assignment is sufficient, then one can use a constant κ^* that is slightly bigger than κ for the computation of the new power levels. As soon as all SINRs are at least κ^* the method can terminate. In this case, the ratio of κ^* and κ defines the precision and the quality of the solution.

Lemma 4.1. *Algorithm 4.1 always terminates.*

Proof: If, at some point, no transmission power is increased by at least $\varepsilon P_{\max}$, then the algorithm has found a feasible power assignment and terminates by definition. So let us assume that at least one transmission is incremented by at least $\varepsilon P_{\max}$ every round. Obviously, this implies that after a finite number of steps some transmission power exceeds $P_{\max}$. This means that the transmission set cannot be scheduled concurrently and the algorithm stops. ■

Lemma 4.2. *The power assignment of Algorithm 4.1 is optimal in the sense that there exists no valid power assignment such that any of the transmissions uses a transmission power less than the power computed by Algorithm 4.1.*

Proof: This can be proven easily by induction. In the beginning, all transmission powers are set to zero and thus provide a lower bound for an optimum solution. So let us assume that at the beginning of the while loop no transmission power overestimates the power in an optimal solution. Then, the interference that we consider is a lower bound of the interference in an optimal solution and the new powers that we compute in turn also do not exceed the powers of an optimal solution. ■

As the optimum power assignment problem is a very basic problem, similar algorithms have been presented in the past. Compare for example [12] and the references given there. It is also possible to compute optimum transmission powers using a linear program. The formulation of the LP is straightforward. However, the scheduling methods that we will introduce later in this thesis have to determine optimum powers very frequently. For this purpose, Algorithm 4.1 is better suited than an LP because it is very fast and does not require an LP solver. Moreover, a very beneficial property of Algorithm 4.1 is that it can be used incrementally: if we already know optimum powers for a set $\mathcal{T}'$ of transmissions and only want to add some more transmissions, then we do not have to start all over again with the computation. Instead, we use the known transmission powers from $\mathcal{T}'$ as lower bounds and compute the initial transmission powers of the new transmissions based on them. Afterwards, the algorithm can be used in its original formulation. Thus, the computation converges very fast, usually within some steps. Either way, in realistic cases, the number of executions of the while-loop can be assumed to be mainly dependent on the precision ε and not so much on the number of transmissions. Therefore, the overall running time of Algorithm 4.1 is $O(n^2 f(\varepsilon))$ for some function f. As ε is a constant, this is equivalent to $O(n^2)$.

4.2 Exact Scheduling Algorithms

As mentioned before, the scheduling problem with fixed transmission powers is NP-hard and the complexity of scheduling with variable powers is yet unknown. Thus, one cannot expect to solve big instances of those scheduling problems in reasonable time. Nevertheless, developing optimized exact methods is still worthwhile as it provides means to estimate the quality of heuristics and to analyze basic properties of optimal schedules based on optimal solutions for small to medium-sized instances. Therefore, we will study optimized CPs and ILPs for the scheduling problem in the following sections.

4.2.1 Constraint Programming

In the first part of this section, we give a CP formulation for scheduling with fixed powers that is mainly based on standard models and constraints provided by Gecode. Subsequently, we show how Gecode can be extended with custom propagators to realize the SINR constraint for variable transmission powers. At this, we use Algorithm 4.1 in the propagator to determine optimum power levels. This section is concluded with a discussion of several approaches that can help to speed up the computation of the CPs.

4.2.1.1 Fixed Transmission Power

The scheduling problem with fixed transmission powers can easily be modeled using the *finite domain integers* model provided by Gecode and described in Section 2.4.1. In order to demonstrate the simplicity of CP definitions in Gecode (as long as Gecode does not have to be extended), we will give brief code snippets of the most important parts of the CP formulation. Variables and arrays are italicized, Gecode methods are highlighted in blue, Gecode classes are underlined, and Gecode constants are colored green.

Let us assume that we already know an upper bound *maxSlots* on the maximal number of slots necessary for an optimal schedule. This can be the number of slots of a schedule that was computed by a simple heuristic or, in the worst case, just the number of transmissions that have to be scheduled. In comparison to the ILPs, which are described later, the CPs are not so sensitive to the quality of this upper bound. For every transmission $t \in \mathcal{T}$ we introduce a *finite domain integer* variable *slot* [*t*] whose domain represents the feasible slot assignments for t during the execution of the CP solver and which finally will contain the slot assignment of an optimum solution. In the beginning, all domains of the variables *slot* [*t*] are set to $\{1, 2, \ldots, maxSlots\}$.

```
IntVarArray slot = IntVarArray(this, n, 1, maxSlots);
```

At this, $n = |\mathcal{T}|$ denotes the number of variables, 1 and *maxSlots* define lower and upper bound of the domains, respectively, and **this** is a pointer to the enclosing object which is a member of the Gecode class Space and represents a solution space in Gecode.

Additionally, we introduce a variable *slotNumber* which represents the number of slots used in the current (partial) solution. *slotNumber* can take on values between 1 and *maxSlots* and is defined as maximum of all *slot*[*t*] variables.

```
IntVar slotNumber = IntVar(this, 1, maxSlots);
max(this, slot, slotNumber);
```

In order to get some deeper insight into constraint programming, let us have a closer look on how this *max*-constraint works. Obviously, as soon as all *slot*[*t*] variables are assigned, *slotNumber* contains the maximum of them. The more interesting part is how the *max*-constraint works during the search process. If we define a new upper bound on *slotNumber*, something that we will do later on, then the domains of all *slot*[*t*] variables are adjusted accordingly. If we assign one of the *slot*[*t*] variables to a value that is greater than the current lower bound of *slotNumber*, then this lower bound is adjusted accordingly. And finally, if we do not assign a variable *slot*[*t*], but only filter some of the possible values from its domain, then this also leads to an adjustment of the lower bound of *slotNumber*, if now the smallest value in the domain of *slot*[*t*] is bigger than the smallest value in the domain of *slotNumber*.

Determining the number of transmissions per slot is rather unspectacular. For every transmission $t \in \mathcal{T}$ we introduce a variable *slotCount*[*t*]. Then, we use Gecode's count-constraint as follows:

```
IntVarArray slotCount = IntVarArray(this, maxSlots, 0, n);
for (int i = 1; i <= maxSlots; i++)
    count(this, slot, i, IRT_EQ, slotCount[i]);
```

We generate one count-propagator per slot. The count-propagator of the *i*-th slot counts the number of variables in the array *slot* whose values are equal to *i*. The count-propagator could also be used to count the number of variables that are less than *i*, greater than *i*, or unequal *i*, by using the constants IRT_LE, IRT_GR, or IRT_NQ, respectively, instead of IRT_EQ. If we set an upper bound *b* on one of the *slotCount* variables and there are already *b* *slot* transmissions assigned to the corresponding slot, then Gecode would remove this slot from all unassigned variables. If thereby one of the domains would become empty, then the partial solution could not be extended to a complete solution and a backtrack step would be performed.

Now let us get to the implementation of the SINR constraint. Here we get the first problem, if we only want to use propagators that are predefined by Gecode: There is, to our knowledge, no propagator available which works with real numbers. Thus, the SINR constraint cannot be implemented directly. The problem can be solved by mapping the real numbers to integers.

```
for (int j = 0; j < n; j++) {
    BoolVarArgs sameSlot = BoolVarArgs(n);
    for (int i = 0; i < n; i++)
        sameSlot[i] = post(this, ~(slot[i] == slot[j]));
    IntArgs c(n);
    for (int i = 0; i < n; i++)
        c[i] = (int)(M / buffer[j] * interference[i][j]);
    linear(this, c, sameSlot, IRT_LQ, M, ICL_BND);
}
```

The outer loop iterates over all n transmissions and constructs one constraint per transmission. For every transmission j, an BoolVarArgs array *sameSlot* of size n is constructed. The post method constructs for every variable *sameSlot*$[i]$ an propagator such that *sameSlot*$[i] = 1$ if and only if transmission i and transmission j are assigned to the same slot. Now, we linearly scale the interferences from all senders to the receiver of transmission j such that the interference $c[i]$ from sender of transmission i to receiver of transmission j equals some big constant M if the interference equals the maximum interference that can be tolerated by transmission j. This maximal tolerable interference will be referred to as *interference buffer of j* in the following.

The new interference value is finally converted to an integer to be usable with Gecode. The error introduced by the conversion from real to integer is less than $1/M$ of the interference buffer of transmission j. As we chose M to be a big constant, this error is negligible. Finally, we have to require that the sum of all interferences from simultaneously active senders does not exceed our normalized interference buffer M. This constraint is posted by the linear method which translates into $\sum_{i=1}^{n} c[i] \cdot \textit{sameSlot}[i] \leq M$. By the optional parameter ICL_BND we tell the propagator that it is sufficient to enforce bound consistency (cf. Definition 2.15).

It is noteworthy that the propagator does not simply take care that the constraint is fulfilled by the current (partial) variable assignment. Instead, every time one of the variable domains is updated, it checks which of the yet unassigned transmissions still would fit into the corresponding time slot. If for some transmission a future assignment to this slot would violate the constraint, then the slot is immediately removed from the domain of that transmission variable. If only one slot remains in the domain, then the transmission is immediately assigned to this slot. If a variable domain becomes empty, then the current partial solution is invalid and backtracking is performed.

Last but not least we have to define the kind of branching that we want to use. The standard branchings which are provided by Gecode can be selected using the branch method.

```
branch(this, slot, INT_VAR_SIZE_MIN, INT_VAL_MIN);
```

The second parameter of branch specifies the array of variables on which we want to branch. In our case this is the *slot* array. The third parameter determines how the variable that is assigned next is elected. With INT_VAR_SIZE_MIN we tell Gecode to use the variable with smallest domain size. This choice should be good for several reasons: Possibly the corresponding transmission can only be placed in a small number of slots. This means that the transmission interferes with many of the already assigned transmissions and thus, if the problem cannot be solved with the given number of slots, it is likely that the transmission belongs to the transmission subset that cannot be scheduled within the given time slots. Moreover, this strategy can help to decompose the problem into subproblems and deal with one subproblem after another. Of course, instead of using the variable with smallest domain size, we could have used all the possibilities listed in Section 2.4.4. The last parameter defines the order in which the remaining domain values of the selected variable are checked out. With INT_VAL_MIN the smallest value of the domain will be chosen. This does not give an advantage over choosing the largest value or some random value, but for a better assignment one has to define his own branching (cf. Section 4.2.1.3).

So far, we have only defined a constraint satisfaction problem. Now we have to extend this CSP to a constraint optimization problem by telling Gecode that we want to minimize the number of slots. To do this, we use the ability of constraint programming to add new constraints during the execution of the CP solver. If we use the branch-and-bound search engine, then, every time a new solution is found, the method constrain of the solution space instance is called. This method can be overridden in our class ScheduleInt which is derived from Gecode's Space class.

```
void ScheduleInt::constrain(Space* s) {
    ScheduleInt* lastSol = static_cast<ScheduleInt*>(s);
    rel(this, slotNumber, IRT_LE, lastSol->slotNumber.val());
}
```

The rel constraint enforces some relation between two variables or between a variable and an integer. Here we define that the slot number has to be smaller than the slot number in the solution that we just found. This invalidates the current solution. The CP solver has to backtrack so far that none of the transmissions is assigned to the last slot of the current solution. That could, in extreme cases, also mean that all current assignments have to be discarded. Starting from the resulting partial solution, the solver tries to find a solution having one slot less than the best solution so far. If no such solution can be found, then we know that the last solution found was optimal with respect to the number of slots.

Now we are finished with our initial formulation of the COP for scheduling with fixed powers. In the following sections we will see how the performance of the COP can be significantly enhanced by adding further constraints and by defining own branchings.

4.2.1.2 Variable Transmission Power

As mentioned before, the standard propagators of Gecode do not support real values. We were able to get around this problem for fixed transmission powers by mapping the reals to integers, accepting a negligible error. Unfortunately, things get more complicated when we want to allow arbitrary transmission powers. It still would be possible to model the power of each transmission by some integer variable, but this would involve n new variables with large domains if we do not want to restrict ourselves to some few power levels. This would significantly increase the computational complexity, so probably only very small problems could be solved by this approach. Fortunately, Gecode can be extended with user-defined propagators and thus allows to model the scheduling problem with variable transmission powers efficiently.

We do not want to go too much into detail about how to define own propagators in Gecode, so let us focus on the main ideas of an efficient implementation of the SINR constraint. The first decision one has to make is which events should trigger the execution of the propagator. It can be triggered if one of the involved variables is changed, if one of the boundaries of an involved variable is changed, or if one of the variable domains is changed. In our problem all variables interact with each other and the situation for the SINR constraint only changes when one of the variables is actually assigned. Therefore, we decided to create one single propagator, which is called every time some variable is assigned or unassigned.

For the following it is important that usually between two executions of the propagator only minor changes do occur. One possibility is that new variables have been assigned. Mostly this is only one variable, but sometimes it happens that the last assignment removed all but one slot from some variables and then those variables are assigned at once. The second possibility is that backtracking took place and some of the variables are again unassigned. We will use this insight, that the situation does not change too much between consecutive executions, in order to avoid recomputations.

For every transmission and every time slot, we store the interference that the transmission currently would receive if it were assigned to that slot. Furthermore, we save for every slot which transmissions are currently assigned to it, what transmission power they currently need to communicate successfully, and how much interference they could tolerate additionally if they sent with maximum power instead of the current transmission power. Each time the propagator is called, this information can be updated efficiently. We first determine which transmissions have been additionally assigned and to which slots they have been assigned. For all slots with changes we use Algorithm 4.1 described in Section 4.1 to update the necessary transmission powers. In most of the cases we only have to add one more transmission and thus Algorithm 4.1 usually converges very fast. In rare cases the execution of the power assignment algorithm shows that the additional transmissions do not fit into the selected slot, and then we have to do backtracking.

Next, we have to take care of the filtering. Again we consider only slots with changes. Based on the new transmission powers, we update for all transmissions that are not yet assigned the interference that they received if they were placed in the respective slot. If this interference exceeds the maximum interference that the transmission can tolerate using maximum transmission power, the slot is removed from the transmission's domain. Additionally, we compute for every slot the minimum transmission power that the sender needs to reach its receiver. With this transmission power we can compute lower bounds on the interference that this transmission would cause to the receivers that are already assigned to the respective slot. For all transmissions already in the slot we check if this additional interference could be tolerated. If this is not the case, then the slot is removed from the domain of the transmission in consideration. Thus, we are able to filter infeasible slots from the variable domains relatively early.

Lastly, we have to consider what to do if the propagator is called after backtracking was performed. Fortunately, Gecode helps us at this. During the branchings, copies of the states of all variables are saved. When backtracking is performed, the respective copy is reloaded and we can just continue as if the intermediate assignments had never occurred. The remaining parts of the COP formulation, such as which search engine and which branchings should be used, are analogous to the ones for scheduling with fixed transmission powers in the last section.

In the next section we will see how the COP can be optimized further.

4.2.1.3 Optimizations

In the following, we will discuss some ideas how to increase the performance of the CSPs and COPs. Some of them turned out to work very well, others actually had bad influence on the performance. We will also try to explain why they succeeded or why they failed.

Symmetry breaking

The set of optimum solutions of a scheduling problem contains an enormous number of solutions that are identical except for unimportant properties such as the order of the slots. If the optimum solution consists of n time slots, then there are, for instance, $n!$ solutions which only differ by the slot order. One first idea to deal with these symmetries was to require that the slot size, meaning the number of transmissions of a slot, was monotonically decreasing in all valid solutions. This can be easily enforced by adding the following relational constraint:

```
for (int i = 0; i < maxSlots-1; i++)
    rel(this, slotCount[i], IRT_GQ, slotCount[i+1]);
```

To our first surprise, the performance went down instead of up. However, this can be easily explained: In order to find the optimum solution, Gecode tries to find any solution that fulfills the constraints and then adds a new constraint which requires that the next solution has to be better than the current one. At this, the aforementioned symmetries have no effect at all. Besides, finding any solution with arbitrary order of slot sizes is much easier than finding a solution where the slot sizes decrease. Only when the CP solver tries to prove that no solution with smaller number of slots exists, the additional constraint helps because it eliminates some of the symmetries. But, this usually does not outweigh the performance losses that we cause during the computation of the optimum solution.

Fixing one transmission

It might sound banal, but even fixing only one transmission to the first slot can already help to speed up the computation noticeably. If n is the number of slots in an optimal solution and one uses no optimization at all, then this trivial additional constraint gives a remarkable speedup of factor $n-1$ for proving that no schedule with only $n-1$ slots exists because it eliminates all possibilities where the fixed transmission is not in the first slot. Fixing two *random* transmissions in either the same slot or in different slots possibly alters the set of optimum solutions.

Fixing a maximum clique of the conflict graph

The ultimate solution for breaking symmetries that are caused by the ordering of the slots is the computation of a maximum clique of the SINR conflict graph and then assigning the transmissions of that clique to different slots. This surely does not alter the set of optimal solutions, as we know that no pair of transmissions from the maximum clique can be scheduled together. As we will see later in Section 6.3.5, the size of a maximum clique usually is about the number of slots in an optimum solution. Thus, we can fix the position of almost all slots and eliminate the corresponding symmetries. Furthermore, we immediately assigned all the transmissions of the clique and thus reduced the size of our problem. As the run time usually grows exponentially with the problem size, this can be a tremendous improvement. Moreover, the maximum clique gives us a lower bound on the number of slots of an optimal solution. Therefore, if we find a schedule that has the length of the size of the maximum clique, then we do not have to prove the optimality of that schedule. Usually, proving the optimality is much more time consuming than finding an optimal solution. So this can make a really big difference.

Generating the additional propagators in Gecode is straightforward. Let us assume that we already have given the transmission IDs of a maximum clique in a list of integers named clique. We now just fix the i-th transmission to the i-th slot.

```
int i = 0;
list<int>::iterator it;
for (it = clique.begin(); it != clique.end(); it++)
    rel(this, slot[*it], IRT_EQ, i++);
```

One might argue that computing a maximum clique is NP-hard. That is true in general, but we will see in Section 4.4 that the structure of the conflict graph usually allows us to compute the maximum clique in reasonable time. And the savings that we can achieve at the computation of an optimal schedule should almost always justify the additional effort. If we consider a big problem and only intend to find a good schedule using the CP, then we can use a heuristic to find a big clique instead of an exact algorithm to compute a maximum clique.

Custom branchings

Defining own branchings offers several possibilities for improvements. The first responsibility of a branching is to select the variable that should be assigned next. In our implementation of a custom branching, we stayed with the standard approach to select the variable with smallest domain size. But probably it would be better to select among all unassigned transmissions the one that interferes most with the other transmissions, based on some appropriate interference measure.

The second decision in a custom branching concerns the values that should be assigned to the selected variable and the order of those values. This offers another great opportunity for symmetry breaking. A standard branching would try to assign all values that are left in the domain of the variable in some order. This means that if there is more than one empty slot, the transmission will be assigned to all those empty slots one after another and thus we will get the redundant solutions mentioned before. If we implement our own branching, then we can avoid this by filtering all empty slots but one from the domain. We also can improve the order in which we check out the different slots. Instead of simply using the slot with smallest id, we can reorder the slots such that slots in which the transmission receives less interference are tried first.

Adding more efficient constraints

Gecode allows to specify the expected running time of propagators. Some possibilities are constant, linear, quadratic, cubic, and exponential running time (in the number of involved variables). Furthermore, one can specify if the constants hidden by the big-O-notation are rather big or small. We can use this to introduce additional propagators that are redundant but faster than the SINR constraint. For example we could add an inequality constraint for every pair of transmissions that cannot be scheduled together.

```
for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
        if (!scheduleable[i][j])
            post(this, slot[i] != slot[j]);
```

This actually leads to a small constant speedup, but as it does not reduce the number of failures and backtracks the advantage is rather small.

Limited Discrepancy Search

Another possibility for performance improvements is the use of *limited discrepancy search* (LDS) as mentioned in Section 2.4.3 about Gecode search engines and described in [14]. Usually, the most time consuming task of solving a COP is to prove the optimality of the final solution. At this LDS does not offer an advantage because all possible assignments have to be checked systematically. Therefore, we did not check out LDS in detail. However, if someone wants to find good solutions for problems that are too big to be solved optimally, having a closer look at LDS might be interesting.

Random Restarts

Another technique which is mainly of interest for problems which are too big to be solved to optimality within the given time is the use of *random restarts*. The goal is similar to the goal of limited discrepancy search: to avoid that some bad decisions are made early in the search tree and that the whole subtree has to be searched before the decisions are revised. Instead of running the CP solver once with timeout T, it is executed z times with timeout T/z. Before each execution, the order of the input variables is shuffled. This increases the possibility that strongly interacting variables are arranged beneficial. It is even possible to control the restarts such that no variable assignment is produced more than once. This guarantees that the optimum solution will be found at some point and that it is possible to prove unsatisfiability with intermediate restarts. Further information on this topic can be found in [10].

4.2.2 Integer Linear Programming

Both, the scheduling problem and the technique of integer linear programming are very famous in the scientific community. Therefore, it is not so surprising that we were not the first ones who thought of using ILPs to solve scheduling problems. Another ILP for scheduling with fixed transmission powers can be found in [2]. Unlike the ILP presented in this thesis, it does not have any optimizations. The objective function and SINR constraint are also implemented slightly different. They also give a column generation method for the scheduling problem. In their experimental results, this method worked extremely fast. For example, for a set with 134 transmissions it only needed 4 minutes and 22 seconds to find a schedule on a computer with only 400 MHz. However, they only tested 6 networks, which, in our opinion, seem to be very advantageous for a column generation approach. For example, the schedule with 134 transmissions had a length of 70 time slots. This means that on average less than two transmissions could be scheduled together.

We did not encounter an ILP formulation for scheduling with variable transmission powers in the literature. Nevertheless, we would not be surprised if it does exist somewhere.

4.2.2.1 Fixed Transmission Power

Let $\mathcal{T}$ be the set of transmission requests, and $n = |\mathcal{T}|$ be the number of transmissions. Let us further assume that we know an upper bound $Z_{\max}$ on the number of necessary slots. Again, $Z_{\max}$ could be the result of a scheduling heuristic or just $Z_{\max} = n$. In order to keep things short, let us define the set $\mathcal{S}$ of possible slots, $\mathcal{S} = \{1, \ldots, Z_{\max}\}$. Additionally, we have to know the parameters of the SINR model: background noise η, minimum SINR κ, path-loss exponent α, and for every transmission t the fixed transmission power P_t. Of course, we could also use the same power P for all transmissions.

For every transmission $t \in \mathcal{T}$ and for every $i \in \mathcal{S}$, we introduce a binary variable x_{ti} which tells us whether transmission t is active in time slot i.

$$x_{ti} = \begin{cases} 1, & \text{if transmission } t \text{ is active in time slot } i \\ 0, & \text{if transmission } t \text{ is inactive in time slot } i \end{cases} \tag{4.1}$$

Additionally, we introduce an integer variable Z, which represents the number of time slots in our solution. We want to minimize Z, so our objective function is obviously

$$\min Z \tag{4.2}$$

Now we have to ensure that Z actually represents the number of slots in use. This can be done with constraint

$$Z \geq x_{ti} i \quad \forall t \in \mathcal{T}, i \in \mathcal{S} \tag{4.3}$$

Using equation

$$\sum_{i \in \mathcal{S}} x_{ti} = 1 \quad \forall t \in \mathcal{T} \tag{4.4}$$

we make sure that every transmission is scheduled exactly once.

For our formulation of the SINR constraint we introduce some new constants. For every transmission $t \in \mathcal{T}$, let

$$S_t = \frac{P_t}{\operatorname{dist}(s_t, r_t)^\alpha} \tag{4.5}$$

be the signal strength from sender s_t to receiver r_t. Furthermore, for every pair of transmissions $s, t \in \mathcal{T}, s \neq t$, let

$$I_{st} = \frac{P_s}{\mathrm{dist}(s_s, r_t)^\alpha} \tag{4.6}$$

denote the interference that sender s_s causes at receiver r_t if s_s is active. We define $I_{tt} = 0$. Next, we compute for every transmission t the maximum interference

$$I_t^{\max} = \sum_{s \in \mathcal{T} \setminus \{t\}} I_{st} \tag{4.7}$$

that can occur if all senders are active concurrently. Thus, $I_t^{\max}$ is an upper bound on the interference that has to be expected for transmission t. Finally, we compute for every transmission the maximum interference B_t that transmission t can tolerate without transmission failures

$$B_t = \frac{S_t}{\kappa} - \eta \tag{4.8}$$

Using this, we can formulate the SINR constraint:

$$I_t^{\max} x_{ti} + \sum_{s \in \mathcal{T}} I_{st}\, x_{si} \leq I_t^{\max} + B_t \quad \forall t \in \mathcal{T}, i \in \mathcal{S} \tag{4.9}$$

The summands with $I_t^{\max}$ make sure that the constraint is only used for the slot in which t is active.

Optimizations

As with the constraint program, we need additional constraints to make the ILP practicable. Some of the ideas that we used for the CPs are directly transferable to the ILPs. Similar to the CP, enforcing a decreasing slot size did not bring performance enhancements on average. In some cases the ILP became faster, in others it was slowed down. Quite different from the CP, using several restarts with random input orders did not help for the ILP. This is not surprising, as the ILP is not so dependent on the input order. The ILP solver can choose in every step an arbitrary variable to be updated. This is a big advantage compared to CP solvers like Gecode, which usually keep the already assigned variables fixed until it is proven that they cannot be extended to a valid solution. However, as mentioned in the CP section, this disadvantage can partly be avoided by using techniques like limited discrepancy search.

Again, the best of all optimization approaches that we checked out was the computation of a maximum clique of the SINR conflict graph and then fixing the transmissions of the clique to different slots. Like for the CP, the advantages are that we decrease the problem size and that we additionally eliminate symmetries by fixing slot positions.

Let us assume that we found a maximum clique of the corresponding SINR conflict graph (with fixed powers) of size $|c_f|$ and that $c_f(i)$ gives us the i-th transmission in the maximum clique. Then, we can fix the transmissions of the maximum clique using the constraints:

$$x_{c_f(j),i} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \qquad \forall j \in \{1, \dots, |c_f|\}, i \in \mathcal{S} \tag{4.10}$$

The complete ILP for scheduling with fixed transmission powers is depicted in Algorithm 4.2. It uses the aforementioned constants.

Input: $\mathcal{T}, \mathcal{S}, I_{st}, I_t^{\max}, B_t, c_f$
Variables: Z, x_{ti}
Output: $x_{ti} = 1 \Leftrightarrow$ transmission t is active in slot i; number of slots Z
Constraints:

$$\begin{array}{rl} \text{minimize } Z & \\ Z \geq i\, x_{ti} & \forall t \in \mathcal{T}, i \in \mathcal{S} \\ \sum_{i \in \mathcal{S}} x_{ti} = 1 & \forall t \in \mathcal{T} \\ I_t^{\max} x_{ti} + \sum_{s \in \mathcal{T}} I_{st}\, x_{si} \leq I_t^{\max} + B_t & \forall t \in \mathcal{T}, i \in \mathcal{S} \\ x_{c_f(j)i} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} & \forall j \in \{1, \dots, |c_f|\}, i \in \mathcal{S} \end{array}$$

Algorithm 4.2: Schedule with minimum number of slots (ILP, fixed power)

4.2.2.2 Variable Transmission Power

The more complex problem to schedule with power control can be modeled using a mixed integer program (MIP). Unfortunately, it seems to be necessary that we introduce one power variable per transmission and slot. We also tried to model the problem using only one power variable per transmission, but this resulted in a quadratic program which could not be solved by CPLEX, because it was not positive semi-definite.

Again, let $\mathcal{T}$ be the set of transmission requests, $n = |\mathcal{T}|$ the number of transmissions, $\mathcal{S} = \{1, \dots, Z_{\max}\}$ the set of available slots, η the background noise, κ the minimum SINR, and α the path-loss exponent. Furthermore, we have to know the maximum transmission power $P_{\max}$.

Like in the case of fixed transmission powers, we use an integer variable Z for the number of time slots in our solution. For all transmissions t and slots i, we use binary variables x_{ti} with $x_{ti} = 1$ if and only if transmission t is active in slot i. Additionally, we introduce real variables P_{ti} which represent the transmission power of transmission t in slot i.

Equations 4.3 and 4.4 from the ILP for fixed powers, as well as objective function 4.2, can be reused in our new MIP. We restrict the transmission powers to the valid values using the constraints:

$$0 \leq P_{ti} \leq P_{\max} \quad \forall t \in \mathcal{T}, i \in \mathcal{S} \tag{4.11}$$

For the formulation of the new SINR constraint, we introduce some abbreviations:

The signal power S_{ti} of transmission t in slot i is computed as:

$$S_{ti} = \frac{P_{ti}}{\operatorname{dist}(s_t, r_t)^{\alpha}} \tag{4.12}$$

Similarly, the interference I_{sti} at the receiver of transmission t due to transmission s in slot i is given as:

$$I_{sti} = \frac{P_{si}}{\operatorname{dist}(s_s, r_t)^{\alpha}} \tag{4.13}$$

An upper bound on the interference at the receiver of t due to transmission s can be computed as:

$$I_{st}^{max} = \frac{P_{\max}}{\operatorname{dist}(s_s, r_t)^{\alpha}} \tag{4.14}$$

This helps to define an upper bound on the maximum interference at the receiver of t

$$I_{t}^{max} = \sum_{s \in \mathcal{T} \setminus \{t\}} I_{st}^{max} \tag{4.15}$$

Finally, the interference buffer B_{ti} of transmission t in slot i in dependence of signal power S_{ti} is given as:

$$B_{ti} = \frac{S_{ti}}{\kappa} - \eta \tag{4.16}$$

This allows a compact formulation of the SINR constraint:

$$I_{t}^{max} x_{ti} + \sum_{s \in \mathcal{T} \setminus \{t\}} I_{sti} \leq I_{t}^{max} + B_{ti} \quad \forall t \in \mathcal{T}, i \in \mathcal{S} \tag{4.17}$$

However, note that this constraint uses many abbreviations. For example, B_{ti} is neither a constant nor a variable, but a term that is dependent on the power of transmission t in slot i.

Lastly, we add the maximum clique optimization: Let us assume that we found a maximum clique of the corresponding SINR conflict graph (with variable powers) of size $|c_v|$ and that $c_v(i)$ gives us the i-th transmission in the maximum clique. Then, we can fix the transmissions of the maximum clique using the constraints:

$$x_{c_v(j),i} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \quad \forall j \in \{1, \ldots, |c_v|\}, i \in \mathcal{S} \tag{4.18}$$

The MIP works as follows: Every transmission has to be assigned to exactly one slot. If the transmission is assigned to a slot, then it has to fulfill the SINR constraint in that slot. Due to background noise η, this is only possible if the transmission power is set to an appropriate value. If this is done, then this causes interference at all other transmissions that are assigned to the same slot. This, in turn, forces them to send with higher power. Thus, the MIP solver computes a slot assignment as well as a valid power assignment for a schedule with minimum length.

The final MIP, using the aforementioned abbreviations, is shown in Algorithm 4.3. It uses $O(n \cdot |\mathcal{S}|)$ variables and constraints. Therefore, it might be worthwhile to compute a good upper bound on the number of slots using some scheduling heuristic.

Input: $\alpha, \kappa, \eta, P_{\max}, \mathcal{T}, \mathcal{S}, c_v$

Variables: Z, x_{ti}, P_{ti}

Output: $x_{ti} = 1 \Leftrightarrow$ transmission t is active in slot i; number of slots Z, transmission powers P_{ti}

Constraints:

$$\begin{array}{rl}
\text{minimize } Z & \\
0 \leq P_{ti} \leq P_{\max} & \forall t \in \mathcal{T}, i \in \mathcal{S} \\
Z \geq i\, x_{ti} & \forall t \in \mathcal{T}, i \in \mathcal{S} \\
I_t^{max} x_{ti} + \sum_{s \in \mathcal{T} \setminus \{t\}} I_{sti} \leq I_t^{max} + B_{ti} & \forall t \in \mathcal{T}, i \in \mathcal{S} \\
\sum_{i \in \mathcal{S}} x_{ti} = 1 & \forall t \in \mathcal{T} \\
x_{c_v(j)i} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} & \forall i \in \mathcal{S}, j \in \{1, \ldots, s_v\}
\end{array}$$

Algorithm 4.3: Schedule with minimum number of slots (ILP, variable powers)

4.2.2.3 Additional Objectives

The MIP can be easily extended to fulfill additional objectives. For example it might be interesting to compute a schedule that is time-optimal and that has the smallest sum of transmission powers among all time-optimal schedules. This can be achieved with the following objective function:

$$\text{minimize} \quad (n+1) \cdot P_{\max} \cdot Z + \sum_{t \in \mathcal{T}, i \in \mathcal{S}} P_{ti} \tag{4.19}$$

Note, that in an optimal solution it holds: $0 < \sum_{t \in \mathcal{T}, i \in \mathcal{S}} P_{ti} \leq n \cdot P_{\max}$.

4.3 Scheduling Heuristics

In the following, we will describe two heuristics for the scheduling problem. They can, for example, be used to achieve good solutions for big instances of the scheduling problem, for an upper bound on the necessary number of slots, or for fast computation of good initial solutions for the ILPs. Section 4.3.1 deals with scheduling with fixed transmission powers, Section 4.3.2 explores the problem with variable powers. In the remainder of this chapter, let $\mathcal{T}$ denote the set of transmission requests and $n = |\mathcal{T}|$.

4.3.1 Fixed Transmission Power: MaxSINR

The main idea of the heuristic consists in processing one time slot after another and trying to fill each slot with as many not yet scheduled transmissions as possible. This way, transmissions that are put together in a slot usually fit together very well, meaning that they do not interfere too much with each other. In my student research paper [46], it is shown that one can achieve a good slot utilization by starting with an empty transmission set $\mathcal{A}$ and consecutively adding the transmission t that maximizes the minimum SINR which occurs if all transmissions in $\mathcal{A} \cup \{t\}$ are executed at the same time.

Note, that the minimum SINR of all transmissions in $\mathcal{A}$ is strictly monotonically decreasing with growing $\mathcal{A}$. This means, if at some point the addition of a transmission $t \in \mathcal{T}$ to the set of active transmissions $\mathcal{A}$ would result in a SINR $\gamma < \kappa$, it surely would also lead to a signal-noise-ratio less than κ in every further pass of the loop. Thus, as soon as the addition of a transmission t to $\mathcal{A}$ leads to a SINR less than κ, we can remove it from the set $\mathcal{T}$ of possible transmissions. This substantially improves the performance of the method.

This approach to fill a time slot efficiently is outlined in Algorithm 4.4. It uses a function *MinimumSINR* which, given a set $\mathcal{A}$ of concurrently active transmissions, computes the minimum occurring SINR of all transmissions in $\mathcal{A}$. In every step of the outer loop, at least one transmission is removed from $\mathcal{T}$. Therefore, the outer loop is executed not more than $O(n)$ times. The inner loop is executed $O(n)$ times as well. If we implemented MinimumSINR naively, then it had a running time of $O(n^2)$. This can be improved to $O(n)$, by storing for each transmission $t \in \mathcal{A}$ the interference at the receiver of t, caused by all transmissions in $\mathcal{A}$. If we add a transmission, then this information can be updated in $O(n)$ and the transmission with minimum SINR can, based on this data, also be determined in $O(n)$. All together we get a running time of $O(n^3)$.

Now that we know how to fill a single slot efficiently, we just have to fill one slot after another until there are no transmissions left to schedule. This approach is outlined in Algorithm 4.5. At this, *FillSlot* is the method described above.

```
Input: Set $\mathcal{T}$ of transmissions, required signal-noise-ratio $\kappa$
Output: Set $\mathcal{A}$ of concurrently feasible transmissions
$\mathcal{A} \leftarrow \emptyset$
maxMinSinr $\leftarrow -1$
bestT $\leftarrow$ undefined
while $\mathcal{T} \neq \emptyset$ do
    forall t $\in \mathcal{T}$ do
        sinr $\leftarrow$ MinimumSINR($\mathcal{A} \cup$ {t})
        if sinr > maxMinSinr then
            maxMinSinr $\leftarrow$ sinr
            bestT $\leftarrow$ t
        if sinr < $\kappa$ then
            $\mathcal{T} \leftarrow \mathcal{T} \setminus$ {t}
    if maxMinSinr $\geq \kappa$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cup$ {bestT}
        $\mathcal{T} \leftarrow \mathcal{T} \setminus$ {bestT}
    else
        $\mathcal{T} \leftarrow \emptyset$
```

Algorithm 4.4: Good utilization of a time slot (fixed transmission powers)

```
Input: Set $\mathcal{T}$ of transmissions
Output: Schedule $\mathcal{S} = (S_1, S_2, ..., S_n)$ such that $\mathcal{S}$ is a partition of $\mathcal{T}$
$i \leftarrow 0$
while $\mathcal{T} \neq \emptyset$ do
    $\mathcal{S}_i \leftarrow$ FillSlot($\mathcal{T}$)
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{S}_i$
    $i \leftarrow$ i+1
```

Algorithm 4.5: Computation of a good schedule

4.3.2 Variable Transmission Power: MinPower

If we allow that senders adjust their transmission powers, then they try to choose the powers such that the resulting SINR equals the minimal SINR that allows for successful communication. For this reason, Algorithm 4.4 is not directly applicable. However, the idea of Algorithm 4.4 can be transfered to scheduling with variable transmission powers: Instead of maximizing the minimum SINR, we just minimize the maximum power that is used in the slot. This leads to Algorithm 4.6.

```
Input: Set $\mathcal{T}$ of transmissions
Output: Subset $\mathcal{A}$ of $\mathcal{T}$ with big cardinality
$\mathcal{A} \leftarrow \emptyset$
$P^*_{min} \leftarrow 0$
while $P^*_{min} \leq P_{max}$ do
    $t_{opt} \leftarrow$ undefined
    $P^*_{min} \leftarrow \infty$
    foreach $t \in \mathcal{T}$ do
        $c_t \leftarrow$ MaximumPower($\mathcal{A} \cup \{t\}$)
        if $c_t < P^*_{min}$ then
            $P^*_{min} = c_t$
            $t_{opt} \leftarrow t$
    if $P^*_{min} \leq P_{max}$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{t_{opt}\}$
        $\mathcal{T} \leftarrow \mathcal{T} \setminus \{t_{opt}\}$
return $\mathcal{A}$
```

Algorithm 4.6: Good utilization of a time slot (variable transmission powers)

In every step of the while-loop, we check for every transmission $t \in \mathcal{T}$ how the transmission powers were affected if we put t in the time slot. For this purpose, we use function *MaximumPower*, which computes the optimum transmission powers for transmission set $\mathcal{A}$ using algorithm 4.1 from Section 4.1. Then, it returns the maximum of all transmission powers. Finally, we choose the $t_{\text{opt}} \in \mathcal{T}$ that resulted in the minimum maximum transmission power. We add t_{opt} to $\mathcal{A}$ and remove it from $\mathcal{T}$. We are finished as soon as either $\mathcal{T}$ is empty, or no transmission can be added without exceeding the maximum transmission power.

As we will show in Section 6.3.5, the approximation ratios of Algorithm 4.4 for fixed transmission powers and those of Algorithm 4.6 for variable transmission powers are very similar.

4.4 Computation of Lower Bounds for Optimum Schedules

In this section, we will discuss methods for the computation of lower bounds for optimum schedules. Lower bounds have several useful applications. For example, they can be used to classify the quality of scheduling heuristics. Moreover, in cases where a good solution is sufficient, the lower bound can help to decide if a computed solution already fulfills the requirements. Sometimes, the lower bound equals the length of an optimum schedule. Then, if we find a schedule whose length equals the lower bound, we immediately know that we are done and do not have to prove optimality or look for better solutions.

Conflict Graph Cliques

By definition, pairs of transmissions that are connected by an edge in the conflict graph cannot be scheduled concurrently. For this reason, the size of a set of transmissions that are all pairwise connected in the conflict graph gives a lower bound on the number of slots of every valid schedule. Such a set of transmission forms a clique in the conflict graph. As we are interested in good bounds, the maximum clique seems to be the tool of choice. We already utilized the maximum conflict graph clique to speed up the CPs and ILPs in Section 4.2. In this section, we will have a closer look on the properties of conflict graph cliques.

Figure 4.1 shows a unit disk graph of 200 nodes which are distributed on an area of 8×8 maximum transmission radii. The UDG has 859 edges that represent possible communication links. In order to examine communications in such a topology, let us replace every link of UDG by two transmission requests, one in each direction. By doing this, we model that every node wants to send one message to every other node within transmission range. The conflict graph of the resulting transmission set consists of 1718 nodes, one for every transmission. If a pair of transmissions cannot be scheduled simultaneously, then they are connected by an edge in the conflict graph. We computed big cliques in the conflict graph and highlighted the involved transmission red in the UDG.

It is striking that all transmission of the clique lie within the same area. This is not surprising, as the interference caused by a sender falls of at least quadratically with distance. However, this locality is extremely important for the feasibility of maximum clique computations. It is well known that the problem of computing a maximum clique is NP-complete. Usually, this means that the running time grows exponentially with problem size. For sufficiently widespread networks, this does not hold in our case. In most realistic networks, nodes are placed apart from each other, to cover an area as big as possible with as few as possible nodes. This restricts the density of nodes per area. Thus, the maximum number of edges incident to any transmission in the conflict graph is bounded by some constant b. This also means that no clique can have size greater

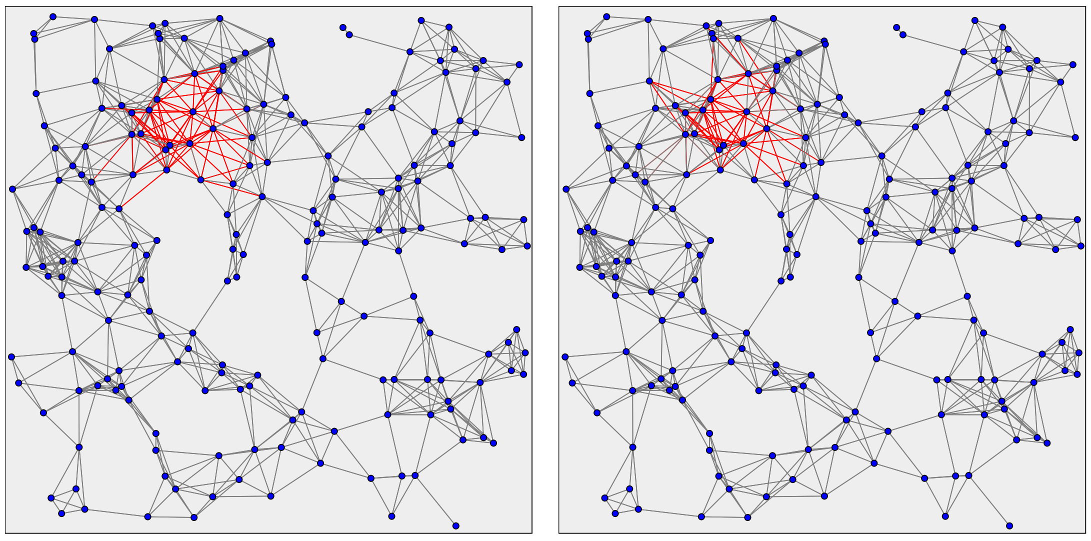

(a) Clique of SINR conflict graph (fixed power) (b) Clique of SINR conflict graph (variable power)

Figure 4.1: Unit disk graphs with maximum conflict graph cliques

than b. As soon as we decide to take one node to our clique, the size of the remaining problem is reduced to $O(b)$. Thus, every of the $O(n)$ subproblems can be solved in time $O(2^b)$. As b is usually rather big, this still might be intractable. Nonetheless, if we consider b to be constant, the time complexity for finding a maximum clique is in $O(n)$ for sufficiently large networks.

Our implementation of the maximum clique computation was able to find maximum cliques up to size 30-35 within reasonable time. In contrast to node density, the size of the network only played a minor role for the computation time. Unfortunately, for networks with densities like the network in Figure 4.1, this is not sufficient. The left picture shows a clique of the SINR conflict graph (fixed power), the right picture a clique of the SINR conflict graph (variable power). They consist of 133 transmissions and 116 transmissions, respectively. Thus, at least with our implementation, this network is out of reach for computation of a maximum clique.

In such cases, when a maximum clique cannot be computed in reasonable time, one can use heuristics to compute big cliques. One simple, yet efficient, heuristic is as follows: Of all nodes in the conflict graph, choose one with maximum degree. If several nodes have the same degree, choose the one whose corresponding transmission has the smallest sender-receiver-distance. In the following, we maintain a list of candidates for our clique. This list is initialized with the neighbors of our selected node. For every node n in the list of candidates, we compute the intersection of the list of neighbors of n and the list of clique candidates. Now, among all nodes in the list of candidates, we choose the one that had the most neighbors in the list of candidates. Again, in case of a tie, we select the node with smaller sender-receiver-distance. The selected node is removed from the list of candidates and added to our clique. Furthermore, all nodes that are not neighbors of the selected node are removed from the list of candidates. This process is repeated until the list of candidates is empty. In our tests, this simple heuristic performed very well. The cliques in Figure 4.1 also have been computed using this heuristic.

Chromatic Number of the Conflict Graph

The *chromatic number* of a graph G is the smallest number of colors needed to color the vertices of G such that no two adjacent vertices share the same color. Given such a coloring of the vertices of our conflict graph, we can assign all transmissions with same color to the same time slot. This gives us a schedule in which all transmissions can transmit pairwise simultaneously. The chromatic number of a graph must be greater than or equal to the size of a maximum clique. Therefore, the chromatic number provides a better lower bound than the maximum clique. Yet, this slight improvement can only be achieved with tremendous additional computational effort. Like the maximum clique problem, the computation of chromatic numbers is NP-complete [40]. Moreover, the locality that helps to speed up the computation of maximum cliques does not exist for the computation of chromatic numbers. As we will see in Section 6.3.5, the lower bound provided by the maximum clique is already almost optimal. For this reason, we do not consider the computation of the chromatic number of the conflict graph being a good lower bound heuristic.

LP relaxation of the ILPs and MIPs

For every ILP or MIP, one can replace the requirement that variables are integer by appropriate continuous constraints. Objective function and constraints are not modified. The resulting problem is called the *LP relaxation* [1] of the original problem. Now that we have only continuous variables, the LP relaxation can be solved in polynomial time using an LP solver. Thus, the LP relaxation usually can be solved for very big problem instances. In general, the solution values of the relaxed problem are not integer. But in many problems, they can be rounded to a feasible solution of the integer program. The objective value of an optimum solution of the relaxed problem gives a lower (upper) bound on the optimum solution of the original minimization (maximization) problem.

To evaluate the quality of this lower bound, we solved the LP relaxation for different formulations of the scheduling problem. Unfortunately, the results were very poor. In the LP relaxation, it is allowed that transmissions are not completely scheduled in a slot, but only fractions of them. What happened was, that all transmissions were distributed over several slots. As a result, the interferences were much lower and the value of the objective function was not a good approximation on the length of an optimal schedule of the original problem.

5 Topology Control

Most of the topology control algorithms proposed recently are *local*. This means, that every node of the network only has to know his own neighborhood in order to decide which links he wants to keep and which links he wants to discard. In realistic scenarios, this locality is a useful property for algorithms used in sensor networks. Especially in ad-hoc networks, nodes usually only know their neighborhood, and it would be too expensive to maintain positions of all involved nodes.

Yet, there are some useful applications for centralized algorithms. For example, if the considered network is static, it can be worthwhile to have a one-time negotiation phase in the beginning, in which all nodes agree on a global schedule. Every link of the topology is assigned to a time slot such that a collision-free transmission is guaranteed. This schedule is repeated perpetual. Especially in networks where the nodes only have to communicate infrequently, and where power-awareness is of special importance, such a TDMA approach (cf. Section 3.2) might be interesting. Of course, the length of the schedule strongly influences the network performance and latency. Therefore, methods that compute topologies that can be scheduled in a minimal number of slots are of high interest.

In this chapter, we describe two algorithms that compute topologies with the goal of minimizing the number of slots necessary to schedule all topology links. One of them is based on the scheduling heuristics from the last chapter, the other one uses an approach similar to the LIFE algorithm (cf. chapter 5.1.2), but using another interference measure that is motivated by the SINR model. Both algorithms can be used to find connected topologies, as well as topologies which are spanners of certain graph properties.

In order to classify the quality of the produced topologies, we will compare them to a number of existing topology control algorithms in the experimental section. Those algorithms are described in Section 5.1.

We conclude this chapter with a description of a CP that computes topologies whose links can be scheduled using minimum number of slots.

5.1 Overview of existing Algorithms and Topologies

Additional to the topology control algorithms that we will describe later, we implemented several existing algorithms and compared them to our algorithms and to each other. These algorithms, as well as some other famous topology control algorithms, are described in the following. Example topologies of most of the described algorithms can be found in the experimental chapter.

5.1.1 Gabriel Graph and Relative Neighborhood Graph

As stated before, local algorithms are of special importance in the field of wireless networks because they allow for a distributed computation and thus help to avoid network traffic and to save energy. Therefore, it is not surprising that well-known locally definable graphs have been among the first proposals for topology control. Two famous examples are Gabriel graphs and relative neighborhood graphs.

In a Gabriel graph, a pair of vertices is only be connected by an edge, if the disk with the vertices as diameter contains no further vertices. This situation is depicted in figure 5.1(a). The area that is shaded gray has to be empty. In our special case of network topologies, the nodes furthermore have to be within transmission range to be connected.

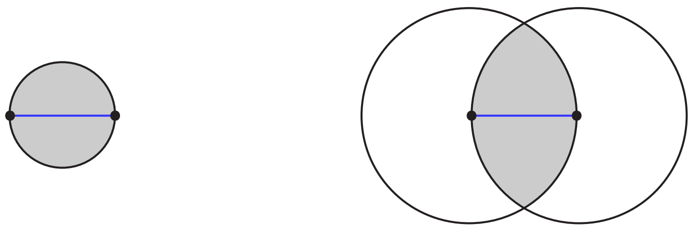

(a) Gabriel Graph (b) Relative Neighborhood Graph

Figure 5.1: Edges in Gabriel graphs and relative neighborhood graphs

In a relative neighborhood graph, two vertices can only be connected by an edge if the intersection of the two disks centered at the endpoints of the edge, with their radii equal to the distance between the two vertices, is empty. This area is highlighted in gray in figure 5.1(b). Again, in the case of network topologies, an edge is not allowed to exceed the maximum transmission radius.

If a node knows its neighborhood, then it can easily decide for all possible links if they belong to the Gabriel graph or to the relative neighborhood graph. Thus, those topologies can be computed extremely efficient in a distributed fashion.

Both, Gabriel graph and relative neighborhood graph are known to be connected, planar and sparse. Further information on relative neighborhood graphs and their relatives can be found in [16].

5.1.2 LIFE and LISE

In [3], Burkhart et al. examine the question if so-far proposed topology control algorithms really help to reduce interference. They show that neither sparseness nor low node degree are guarantees for low interference. Furthermore, they precisely define their notion of interference. This definition of interference is based on the question, how many nodes are affected by communication over a certain link. Given a network $G = (V, E)$ and a transmission $(u, v) \in E$ from sender $u \in V$ to receiver $v \in V$, they define $D(u, r)$ to be the disk centered at u with radius $r = \text{dist}(u, v)$. If u sends to v, according to their definition, all nodes within the disk $D(u, r)$ are affected. Additionally, they define the coverage $\text{Cov}(u, v)$ of an edge (u, v) as the number of network nodes that are affected, if nodes u and v communicate with each other. At this, it is assumed that u and v choose their respective transmission powers such, that they exactly reach each other. Formally:

$$\text{Cov}(e) := |\{w \in V \mid D(u, |u,v|) \,\text{covers}\, w\}| \,\cup\, |\{w \in V \mid D(v, |v,u|) \,\text{covers}\, w\}|$$

Based on this interference measure, they further introduce two scheduling algorithms.

Low Interference Forest Establisher (LIFE) LIFE is a global algorithm which finds a connectivity-maintaining, interference-optimal topology $G' = (V, E')$, meaning that the connectivity is preserved and the maximum coverage $\text{Cov}^*(E')$ of all edges $e \in E'$ is minimized. The algorithm works similar to Kruskal's minimum spanning forest algorithm, using the coverage of the links as edge weights.

Low Interference Spanner Establisher (LISE) In contrast to LIFE, LISE establishes a topology that is a distance-spanner. With slight modifications, the algorithm can also be used to compute a hop-spanner topology. First, the algorithm computes a link with minimum coverage Cov_{min}. Then, it adds all links with coverage Cov_{min} to the topology. If the desired spanner-property is fulfilled, the algorithm stops. If not, it again determines an edge with minimum coverage Cov_{min} among the remaining edges and adds all edges with coverage Cov_{min} to the topology. This is repeated until the desired spanner-property is satisfied. Again, the topology is interference-optimal, meaning that the maximum interference among all edges of the topology is minimized. Furthermore, in order to compute a t-spanner, every node only has to know its $(\frac{t}{2})$-neighborhood. Thus, the algorithm is local.

5.1.3 Minimum Spanning Tree Algorithms

There exist several algorithms, which, like the LIFE algorithm, define some weight on the edges and subsequently compute a minimum spanning tree of the weighted graph. Probably the most famous of them is the Euclidean minimum spanning tree (EMST), which uses the Euclidean distance as edge weights. The EMST topology can also be used to bound the maximum node degree, as every EMST of a finite set of points in the plane has a maximum node degree of six [29].

5.1.4 Cone Based Topology Control (CBTC)

The basic idea of the CBTC algorithm is that every node u transmits with the minimum power $p_{u,\alpha}$ required to ensure that in every cone of degree α around u, there is some node that u can reach with power $p_{u,\alpha}$. In [24, 25], Li et al. show that using $\alpha = 5\pi/6$ is a necessary and also a sufficient condition to guarantee that network connectivity is preserved. Besides, they propose three optimizations that further reduce power consumption and prove that they retain network connectivity.

A node u is said to be a *boundary node* if u has still an α-gap at the end of the algorithm. In the basic CBTC(α) algorithm, all *boundary nodes* would broadcast with maximum power. The *Shrink-Back Optimization* reduces the transmission powers of all *boundary nodes* as much as possible without reducing the cone coverage of the nodes.

Let E_α be the set of edges of the topology $G_\alpha = (V, E_\alpha)$ computed by the CBTC(α) algorithm. The second optimization, the *Asymmetric Edge Removal*, removes all directed edges $(u, v) \in E_\alpha$ from the topology G_α for which $(v, u) \notin E_\alpha$. Li *et al.* prove that this optimization preserves connectivity for $\alpha \leq 2\pi/3$.

The so called *Pairwise Edge Removal* is the third and last optimization proposed in [25]. They show that, for $\alpha \leq 5\pi/6$, if there is an edge from u to v and from u to w, then the longer edge is *redundant* and can be removed as long as $d(v, w) < \max(d(u, v), d(u, w))$. Since the transmission power of each node should be reduced, only the redundant edges with length greater than the longest non-redundant edge are removed.

One of the advantages of their algorithm is that it does not need the exact positions of the sensor nodes but only directional information. According to the authors, CBTC was the first algorithm that simultaneously achieved a variety of useful properties, such as symmetry, sparseness, and good routes [25].

5.1.5 XTC Algorithm

The *XTC* algorithms was proposed by Wattenhofer *et al.* in [48]. Compared to most previously proposed algorithms, it is really simple: Two vertices u and v are connected if they are within transmission range and if there is no vertex w which is closer to either u or v than u and v are to each other. XTC is strictly local and does not require availability of node position information. Besides, the underlying network graph does not need to be a Unit Disk Graph. Instead, *XTC* works on every general weighted network graph.

For the special case that the network graph is a Unit Disk Graph, the resulting topology has bounded degree, is a planar graph, and—on average-case graphs—a good spanner [48].

5.2 Topologies which can be efficiently scheduled

In the following sections, we study the problem of computing time-optimal topologies. We define a time-optimal topology as follows:

Definition 5.1 (Time-optimal topology)**.** Given an input graph $G = (V, E)$ and some graph property Ψ, a topology $G' = (V, E')$, $E' \subseteq E$, is a *time-optimal* topology of G regarding Ψ, if it fulfills Ψ and if there does not exist another topology $G^*(V, E^*)$, $E^* \subseteq E$, which also fulfills Ψ and whose edges can be scheduled with less time slots than the minimum number of slots required to schedule all edges of G'.

At this, every edge $\{u, v\}$ of a topology is regarded as two transmissions (u, v) and (v, u), one in each direction. In the following, we require that the considered graph property Ψ is either preservation of connectivity, or some spanner-property, such as hop-spanner, distance-spanner, or power-spanner. The proposed methods cannot be used with arbitrary properties. In this case, preservation of connectivity means that each pair of nodes that is connected by a path in G also has to be connected by a path in G'.

5.2.1 MaxSINR Topology and MinPower Topology

In our experiments, the scheduling heuristics from Section 4.3 produced very good schedules. Therefore, it is tempting to extend them to topology control algorithms, hoping that the links of the resulting topologies can also be scheduled efficiently. The basic idea of both scheduling algorithms was to fill the slots one by one, as good as possible. Every slot is filled in a greedy fashion, such that the minimum SINR is maximized, or that the necessary maximum power is minimized. These approaches can be extended to topology control algorithms as follows: We still add the transmissions one by one to the schedule. Furthermore, we maintain a data structure, which allows to check efficiently if a transmission t is still necessary to fulfill the required graph property Ψ. As soon as we encounter a transmission that is no longer necessary, we remove it from the list of unprocessed transmissions. In the end, all transmission that have been assigned to slots belong to the topology.

If we want the topology to be undirected, we have to take care that for every transmission that has been scheduled, the second transmission belonging to the same edge (in opposite direction) also will be scheduled. For this purpose, we add a marker to each transmission. As soon as one of the transmissions belonging to an edge is added to the topology, we mark the second transmission of the edge as necessary. Thus, the transmission will be scheduled for sure at some point. Besides, as soon as we add the first transmission $t_1 = (u, v)$ of a pair (t_1, t_2) of transmissions to some slot, we update our data structure as if $t_2 = (v, u)$ already belongs to the topology.

Let us see how we can maintain the aforementioned data structure and how we can decide if a transmission is unnecessary:

Connectivity If m is the number of nodes in the network, we just need a $m \times m$-matrix $\mathcal{C}$, which stores for every pair of nodes whether it is already connected. Every time a transmission (u, v) is added, we check for all nodes if they are already connected by some path to u and not yet connected to v. For all nodes that fulfill this requirement, we update the connectivity information in $\mathcal{C}$ such that they now are also connected to v, as well as to all nodes that are reachable from v.
If we have to decide if a transmission $t = (u, v)$ should be added to the topology, we check using $\mathcal{C}[u, v]$ if they are already connected by some path. If this is true and t is not marked to be necessary, we know that we can discard t.

It is obvious, that in the end all pairs of nodes that are in the same connected component of the input graph are also connected by some path in the resulting topology. This holds, because no edge which could be used to connect two yet unconnected components in the new topology is discarded.

Spanner-Property Given some distance measure $\mathcal{M}$, we compute for all pairs (u, v) of nodes the length $p(u, v)$ of a shortest path in the weighted input graph $G = (V, E)$. If we want to compute a s-spanner $G' = (V, E')$ with regard to $\mathcal{M}$, we have to make sure that the length $p'(u, v)$ of a shortest path between every pair (u, v) of nodes in G' is at most s times the length $p(u, v)$. In the beginning, all distances in G' are set to ∞. Every time we add a transmission $t = (u, v)$ with edge-weight $w(u, v)$, we check for all nodes $x \in V$ if $p'(x, u) + w(u, v) < p'(x, v)$. For all nodes which fulfill this condition, we update all shortest path lengths $p'(x, z)$ to $p'(x, z) = \min\{p'(x, z), p'(x, u) + w(u, v) + p'(v, z)\}$.
If we have to decide if a transmission $t = (u, v)$ should be added to the topology, we check if $p'(u, v) \leq s \cdot p(u, v)$. If this is true, we know that there already exists a path between u and v that fulfills the s-spanner property, and, if t is not marked as necessary, we can discard it.

In the end, every node u is connected in G' to all of its neighbors v in G by a path $p'(u, v)$ with $p'(u, v) \leq s \cdot w(u, v)$.

Then, for all $u, v \in V$, it holds that $p'(u, v) \leq s \cdot p(u, v)$. *Proof:* Let $(u, x_1, \ldots, x_n, v)$ be a shortest path between u and v in G. Every edge (v_1, v_2) of this path can be replaced in G' by a path with length less than or equal to $s \cdot w(v_1, v_2)$. Thus, there exists a path in G' that is not longer than $s \cdot p(u, v)$ and G' is an s-spanner with respect to the distance measure.

Note, that these algorithms are not proposed to be used in real networks. They are mainly intended to verify if topologies that are generated by such an approach are easier to schedule than topologies which are produced by simpler heuristics. We will see later, that the produced topologies can be scheduled very well, but that the much easier approach to greedily take the shortest links into the topology generates topologies that are equally good.

5.2.2 MinInterference Topology

Many of the existing topology algorithms try to minimize some interference measure. The proposed interference measures range from node degrees to the coverage of an edge that was used for the LIFE and LISE algorithms (cf. Section 5.1.2).

To our knowledge, there exists no interference measure for topology edges that is motivated by the SINR model. Such an interference measure has to take into account that transmissions over shorter distances can tolerate more interference and that the interference received from some sender decreases with the distance with path loss exponent α. Furthermore, if the measure should be meaningful, there has to be some upper bound on the interference received from a single sender. This is because the worst thing that can happen is that a pair of transmissions cannot be scheduled concurrently. If this is the case, it does not matter how close together interfering sender an disturbed receiver actually are.

Therefore, we define that the interference $\mathcal{I}(t_1, t_2)$ from transmission t_1 to another transmission t_2 equals 1 if t_1 and t_2 cannot be active simultaneously. For a pair of transmissions (t_1, t_2) that can be scheduled together, we define $\mathcal{I}(t_1, t_2)$ as follows: Let $B(t_2)$ be the maximum interference that t_2 can tolerate if the sender of t_2 sends with maximum power, and let $I(t_1, t_2)$ be the interference that transmission t_1 causes at t_2 in the case that the sender of t_1 sends with maximum power. Now, for our interference measure, we define the interference $\mathcal{I}(t_1, t_2)$ from t_1 to t_2 as $\mathcal{I}(t_1, t_2) = I(t_1, t_2)/B(t_2)$. This equals the fraction that t_1 uses of the interference buffer of t_2 if both transmit with maximum power.

Then, the interference $\mathcal{I}(t)$ of a transmission t is the sum of the interferences from all transmissions that interfere with t. The interference $\mathcal{I}(e)$ of a topology edge $e = \{u, v\}$ is defined as the sum of the interferences of transmissions $t_1 = (u, v)$ and $t_2 = (v, u)$ that represent e in our optimization problem.

The proposed topology control algorithm is now similar to the LIFE algorithm, with the difference that we use our interference measure $\mathcal{I}(e)$ instead of coverage $\mathrm{Cov}(e)$. In order to be able to compute spanner topologies, we further use the techniques from Section 5.2.1. It is shown in [3], that it is sufficient to consider the t-neighborhood of every node to compute a topology that is a t-spanner based on an appropriate distance measure. Thus, if we compute a spanner instead of a spanning tree, the algorithm can be realized based on local information.

Our interference measure $\mathcal{I}(e)$ has several advantages compared to $\mathrm{Cov}(e)$: Both, the interference buffer $B(t)$ of a transmission and the interferences $I(t_1, t_2)$ can be determined without having any knowledge about positions, distances, or directions. In an initial phase, the nodes just have to send *ping* messages with maximum power to allow the other nodes to measure the received signal strength. Then, the nodes tell the other nodes which signal strengths they measured. There is nothing more necessary to compute the required information. This approach even takes into account obstacles that negatively influence the power gain between pairs of nodes. Furthermore, it is considered that transmissions over short ranges can tolerate more interference.

Note, however, that we build our topology based on the interferences weights that every node gets assigned in the input graph. In general, the final topology has much less edges. Thus, the interference measure of an edge differs between input graph and output topology. The resulting topology therefore is not optimal regarding our interference measure. Nevertheless, it seems to be a good heuristic and this topology control algorithm performed very well in our experiments. It should be especially useful in situations where no kind of position information is available, or when the signal strength is not solely determined by distance.

5.2.3 Constraint Programming

The constraint programs from Section 4.2.1 can be extended to a CP formulation that finds, given an input graph $G = (V, E)$, a time-optimal topology fulfilling the desired graph property Ψ. For this, we introduce for every edge $e = \{u, v\}$of the input graph two transmissions $t_1 = (u, v)$ and $t_2 = (v, u)$, which have to be scheduled. In the following, let $m = |V|$ be the number of nodes, and $n = 2|E|$ the number of transmissions.

To be able to model that some edges do not belong to the resulting topology, we introduce a new *dummy* time slot with ID 0. The transmissions corresponding to edges that do not belong to the final topology are assigned to this dummy time slot.

Let us first assume that we want to preserve connectivity in our resulting topology. We introduce m^2 new Boolean variables $\mathit{connected}[u][v]$, which store for every pair (u, v) of nodes, whether they are already connected in the current partial topology. If a transmission $t = (u, v)$ is added to the partial topology, then its sender and receiver are connected. This can be realized by a constraint

$$\mathit{slot}[t] > 0 \Rightarrow \mathit{connected}[u][v] = 1 \tag{5.1}$$

for all transmissions $(u, v) \in \mathcal{T}$. For every triple (u, v, w) of nodes, we add another constraint

$$\mathit{connected}[u][v] \wedge \mathit{connected}[v][w] \Rightarrow \mathit{connected}[u][w] \tag{5.2}$$

Now, every time a transmission t is added, Gecode updates the connectivity variables in $O(n^2)$.

To achieve that transmissions are not taken into the topology, if they are not necessary because sender and receiver are already connected otherwise, we add, for every transmission $t = (u, v)$, a new custom propagator to $\mathit{connected}[u][v]$, which enforces:

$$\mathit{connected}[u][v] = 1 \wedge \mathit{slot}[t] \text{ is not yet assigned} \Rightarrow \mathit{slot}[t] = 0 \tag{5.3}$$

Next, as we want to get an undirected topology, we add for each edge $\{u, v\} \in E$ with corresponding transmissions $t_1 = (u, v)$ and $t_2 = (v, u)$ a constraint:

$$\mathit{slot}[t_1] > 0 \Leftrightarrow \mathit{slot}[t_2] > 0 \tag{5.4}$$

Now, let us get to the branching: As we do not have to schedule all transmissions, it does no longer seem to be a wise choice to always select the transmission variable with minimum domain size for the next branching. If we would do so, we would first check out the variables that suffer from the most interference. There is a good chance, that these variables do not belong to an optimum topology. Instead, we order the transmissions according to their sender-receiver-distance, and begin with the short-distance transmissions. As we will see later in the experimental section, good topologies usually mainly consist of the shortest edges of the input graph. Now, the order in which the values from the domain of the chosen variable are checked out has to be determined. Again, it seems to be a good choice to start with the slots in which the considered transmission receives least interference. If we unsuccessfully tested out all possibilities to assign the transmission to one of the slots, we assign it to the dummy slot 0, meaning that the corresponding edge will not be in the final topology. Thanks to constraint 5.4, the second transmission belonging to the same edge will automatically be assigned to slot 0.

Every solution of the CP that is found defines a spanning forest of the input graph. As soon as we find a new solution, we add a new constraint that enforces that the next solution has to be scheduled using one slot less.

This CP can be easily adjusted to compute topologies which fulfill arbitrary graph properties. Let us, for example, assume that we want to compute a topology that is a distance s-spanner. To do this, we maintain a matrix of shortest paths in the partial solution and implement a new propagator, which keeps this matrix up to date. As we are only interested in a distance s-spanner, it is sufficient to consider the s-neighborhood (meaning the neighborhood within a radius of s times the maximum transmission range) of every node. If for every node the distance stretch factor to every other node in the s-neighborhood is below s, then the stretch factor to all nodes in the topology is below s [3]. Now, the decision whether a transmission has to be scheduled or not is made directly within the branching. If the nodes involved in a transmission t are already connected with stretch factor less than s in the current topology, then t is unnecessary and we can immediately set $slot[t] = 0$. If the nodes are not yet connected, or if they are connected only with stretch factor greater than s, we first try to assign the transmission to any of the possible slots before we decide to discard it from our topology.

Note that the maximum clique optimization cannot be used in connection with this CP formulation, as we cannot be sure that all transmissions of a maximum clique do belong to edges of a time-optimal topology. We first expected that the CP that only preserves connectivity should be usable in networks with many edges, because we never have to schedule more transmissions at the same time than twice the number of nodes. Unfortunately, if we want to prove optimality, this assumption is wrong. The reason for this is, that, given $2|E|$ transmissions, the number of valid partial schedules is immense, even if not all transmissions have to be scheduled at the same time.

6 Experimental Results

Most of the algorithms and techniques described in this thesis have been implemented in Java and C++. In this chapter, we use these implementations to analyze some fundamental properties of sensor networks and to compare the performance and running times of the different methods. At this, the comparison between the constraint programs and the integer programs is one of the main aspects.

This chapter is organized as follows: In Section 6.1, the implementation and testing environment are specified. Afterwards, in Section 6.2, we show important aspects of the choice of SINR model parameters and describe which parameters were used for the experiments in this thesis. The experiments that are related to scheduling can be found in Section 6.3, those related to topology control in Section 6.4.

6.1 Implementation and Testing Environment

The algorithms and methods which are described in this thesis have been implemented partly in *Java* and partly in *C++*. For the solution of the ILPs and CPs, we used the external solvers *ILOG CPLEX 11* [5] and *Gecode 2.1.1* [7]. CPLEX is a commercial product and is known to be one of the fastest and most developed solvers for (integer) linear programs. Gecode is an open source environment for modeling and solution of constraint programs. A detailed introduction into Gecode is given in Section 2.4.

Originally, we intended to do all the implementations in Java, in order to avoid implementing the core functionality (e.g., the SINR model) twice. We decided on Java because it is freely available, highly portable, almost as fast as C++, and there are several good libraries for the visualization of graphs and networks available.

Unfortunately, the Gecode libraries for Java are currently, according to the authors, mainly intended for testing purposes and not optimized for high performance. Normally, this is no problem as the performance of a constraint program is usually measured in the number of failures and backtracks, and not in running time. The number of failures allows for comparisons which are independent of the testing environment and the CP solver in use. But as one of the goals of this thesis was to evaluate the usability of constraint programming compared to integer linear programming, it would have been unfair to compare the actual running times of the Java Gecode implementation with the running times of the highly optimized ILOG CPLEX Solver. For this reason, the constraint programming related parts have been implemented in C++.

For the visualization of the sensor networks, we used *JUNG 2.0 alpha 1* [17], the Java Universal Network/Graph Framework.

The experiments and running time comparisons have been done on a computer with two Dual-Core AMD Opteron™ 2218 processors (2.6 GHz) and 32 GB RAM, running Linux 2.6.16.13 (SUSE).

6.2 Parameters used for the SINR Model

In order to stay realistic, it is assumed that every sensor node has some maximum transmission power $P_{\max}$. For the sake of simplicity, we use the same maximum transmission power for all sensor nodes. In the case of fixed transmission powers, we set $P = P_{\max}$. The minimum SINR κ is set to 10, a value which should be realistic for modern hardware. For the path loss exponent we choose $\alpha = 4$.

If we consider the case that there is no interference due to concurrent transmissions and solve the SINR inequality for the distance d between sender and receiver, this gives us:

$$d \leq \sqrt[\alpha]{\frac{P_{\max}}{\eta\kappa}} \tag{6.1}$$

As α, η, and κ are constants, it is obvious that the maximum transmission power implies a maximum transmission range $d_{\max}$. In order to make the results descriptive for the reader it seems to be natural to normalize the parameters by setting η such that $P_{\max} = 1$ and $d_{\max} = 1$. This means $\eta = \kappa^{-1}$.

However, this can lead to undesirable results. If we allowed transmissions over a distance of almost one, then this would mean that the transmission cannot tolerate concurrent transmissions even from senders relatively far away. This is unrealistic and complicates theoretical considerations about the model. Therefore, it is better to restrict transmissions to a maximum sender-receiver-distance $d^*_{\max} < d_{\max}$. In the following, we will refer to $d^*_{\max}$ as *transmission radius*. We can normalize $d^*_{\max}$ to the value 1 using the following definition for η:

$$\eta = \frac{1}{(1+\beta)\kappa} \tag{6.2}$$

In this equation, β is some arbitrary constant that defines how much interference (in multiples of η) every transmission with transmission power $P_{\max}$ and sender-receiver-distance less than or equal $d^*_{\max} = 1$ should be able to tolerate from concurrent transmissions.

The advantage of using $d^*_{\max}$ instead of $d_{\max}$ as maximum sender-receiver-distance is shown in Figure 6.1. The left picture shows 145 nodes which are randomly distributed in an area of 12×12 units of length. Every pair of nodes with distance not greater than 1 is connected. Now, we compare the conflict graphs in dependence of our choice

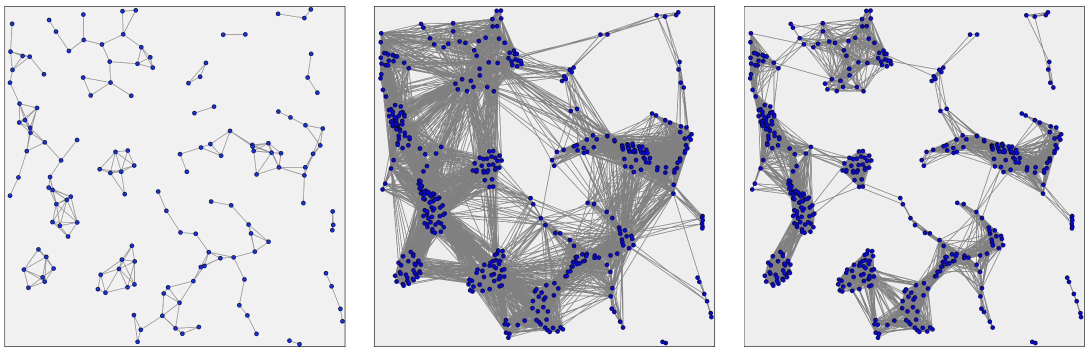

(a) Unit Disk Graph (b) Bad model parameters (c) Good model parameters

Figure 6.1: Good vs. bad choice of SINR model parameters

of β. The result for $\beta = 0$ can be seen in Figure 6.1(b): Even transmissions which are really far away from each other conflict with each other. Theoretically, a transmission over distance $d_{\max}$ could fail because of one single active sender in arbitrary distance. Figure 6.1(c) shows the situation for $\beta = 1$. Here, the conflicts only occur between transmissions whose nodes are near to each other as one would expect in reality.

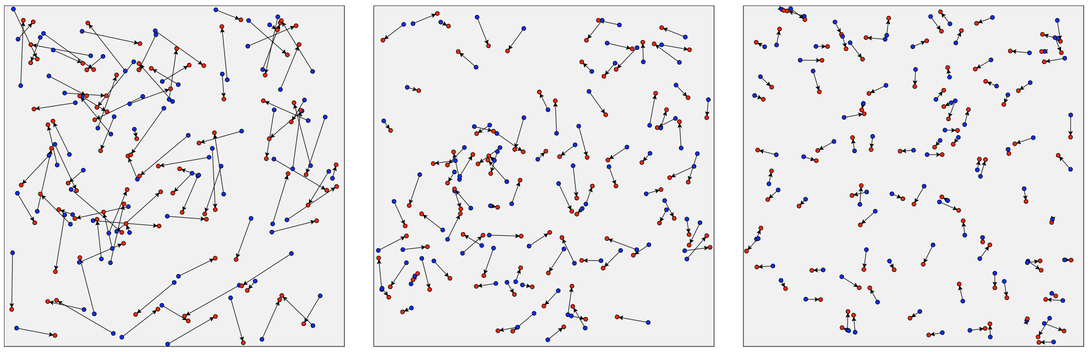
6.3 Scheduling

In the following sections, we study several properties of wireless networks that are related to scheduling. After a brief description of our test data, we compare graph-based interference models with the SINR model. Subsequently, we examine optimum schedules and the advantages of using variable transmission powers instead of fixed transmission powers. This is followed by an evaluation of the quality of our scheduling heuristics, as well as of the lower bounds that are achieved by computing a maximum clique of the conflict graph. The ILPs and CPs are compared based on the percentage of problems which they were able to solve within 180 seconds, and based on the average number of slots of the schedules that they found within the given time. This section concludes with an evaluation of random restarts as a tool to speed up constraint programs.

6.3.1 Test data

In order to make the different experiments as comparable as possible, most of the experiments were run on the same sets of input data. The defining properties of each data set are the number of transmissions and the dimensions of the area on which the transmissions are placed. Obviously, both play an important role for the running time of the algorithms as well as for the number of slots in an optimum solution. The dimensions of the considered area are given in transmission radii. At this, a transmission radius is the maximum distance between a sender and a receiver, such that they still are able to communicate if the only interference is caused by the background noise η. The data sets are divided into three big classes, depending on the dimensions of the test area: 5×5 transmission radii, 10×10 transmission radii, and 15×15 transmission radii. Of course, this only covers rather small sensor networks. But, due to the NP-hardness of the scheduling problem, the computation of exact solutions for bigger instances would be too time-consuming. Nevertheless, the results show clear tendencies and thus can be easily transfered to bigger dimensions.

For the classes with 5×5, 10×10, and 15×15 transmission radii, we created data sets with $n = 10, 20, ..., 100$, $n = 10, 20, ..., 120$, and $n = 10, 20, ..., 170$ transmissions, respectively. Each data set consists of 50 random transmission sets. They were generated by first placing each sender randomly within the test area and, afterwards, placing the corresponding receiver randomly within transmission range such that the receiver also lies in the test area.



(a) 5×5 radii, 100 transmissions (b) 10×10 radii, 100 transmissions (c) 15×15 radii, 100 transmissions

Figure 6.2: Examples of the transmission sets used for the scheduling experiments

Of course, the transmission density, meaning the number of transmissions per unit area, is the main determination factor for the number of slots in an optimum schedule. Therefore, in order to give a first impression on the transmission densities, Figure 6.2 depicts three of the overall 1950 transmission sets.

6.3.2 Graph-based Models vs. SINR Model

There are several scheduling algorithms that are based on conflict graphs. Depending on the quality of the conflict graph used, these algorithms either put transmissions which cannot be scheduled together into the same slot, or they do not allow the concurrent execution of transmissions which actually could be scheduled together. It is widely believed that the SINR model represents reality pretty well. Therefore, it seems to be interesting to compare some frequently used conflict graph models to the SINR conflict graph. Edges additional to the edges of the corresponding SINR conflict graph indicate that the considered model is over-restrictive, edges missing compared to the SINR conflict graph indicate that the model is not sufficiently restrictive.

The first experiment to compare graph based conflict graphs with the SINR conflict graph is accomplished as follows: We generated 100 node sets of 300 nodes at a time, which have been randomly distributed within an area of 15×15 transmission radii. For each of the node sets, the unit disk graph was constructed. Then, we computed the conflict graphs of the following models:

Distance-2 Fixed Model In the distance-2-fixed model, every sender has a fixed transmission range r. If two transmissions $t_1 = (u, v)$ and $t_2 = (u', v')$ are active concurrently, then transmission t_1 fails if

$$\mathrm{dist}(u', v) \leq r. \tag{6.3}$$

Distance-2 Variable Model In the distance-2-variable model, every node adjusts its transmission range on a per-message basis. Here, transmission t_1 fails if

$$\mathrm{dist}(u', v) \leq \mathrm{dist}(u', v'). \tag{6.4}$$

Sigma Model In the σ-model with parameter $\sigma \geq 0$, the distance between sender u and corresponding receiver v is taken into account. Furthermore, the parameter σ allows to specify, that the interference is not restricted to the maximum transmission range r. Transmission $t_1 = (u, v)$ fails, if for some concurrently active sender u'

$$\mathrm{dist}(u', v) \leq (1 + \sigma)\mathrm{dist}(u, v). \tag{6.5}$$

This model comes closest to the SINR model.

Two Hop Model In the two-hop model, two transmissions are interfering if the corresponding edges in the unit disk graph are adjacent or if they are connected by an edge. This model was used by Tamura et al. in [42].

The average edge numbers of the resulting conflict graphs now give some insight on how the methods behave compared to the SINR conflict graphs. The results can be seen in Table 6.1. For the sigma model, $\sigma = 2.16$ was used.

	Distance-2 Fix	Distance-2 Var	Sigma	Two Hop
Conflict graph edges	12161.9	9127.1	20464.4	13854.7
Additional to SINR Fixed	476.1	44.5	2664.2	808.4
Missing from SINR Fixed	6114.4	8717.6	0.0	4753.9
Additional to SINR Var	1717.7	401.5	7849.0	2508.9
Missing from SINR Var	2171.2	3889.8	0.0	1269.6

Table 6.1: Number of edges in different conflict graph models

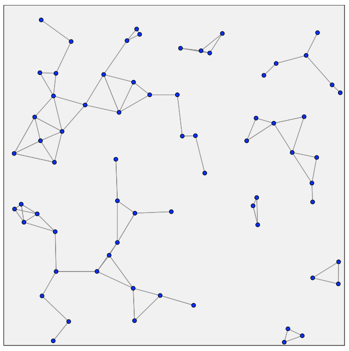
The SINR conflict graph (fixed power) and SINR conflict graph (variable power) had an average of 17800 and 12615 edges, respectively. Again, it can be seen that the use of variable transmission powers pays off. The first thing that strikes if one looks at Table 6.1 is that all conflict graphs contain a big number of edges that do not exist in the corresponding SINR conflict graph. The reason for this is, that the models do not take advantage of short distances between senders and corresponding receivers. As the signal quality decreases extremely fast with the distance, a short sender-receiver distance means that the transmission can tolerate much interference from outside. However, being too cautious with concurrent transmissions is by far not as bad as allowing transmissions to be scheduled together which interfere so much that one or both transmissions fail. Unfortunately, all models but the Sigma model are missing a tremendous number of edges from the SINR conflict graph. This certainly shows that scheduling based on this conflict graphs does not result in reliable schedules.

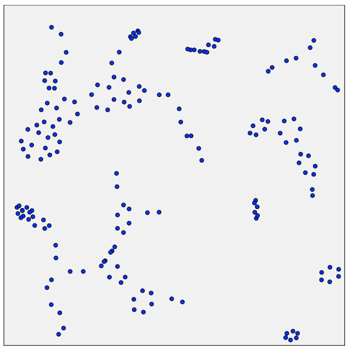
So let us have a closer look on how the conflict graph models differ. Figure 6.3(a) shows a set of randomly distributed sensor nodes with corresponding unit disk graph. The vertices of the corresponding conflict graph are depicted in Figure 6.3(b). In the remaining pictures of Figure 6.3, the different conflict graphs are compared. Probably the most obvious distinction is, that the SINR conflict graphs connect edges that do not belong to the same connected component.

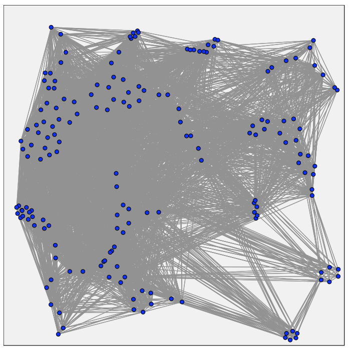
If one wants to use a graph based model and tolerates rare cases of transmission failures, then the SINR based conflict graphs should be the first choice. On the other hand, if failures are not tolerable, then the methods proposed in my student research paper [46] and in [20] should be considered.

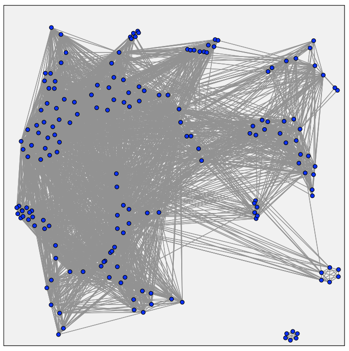
6.3.3 Analysis of Optimum Schedules

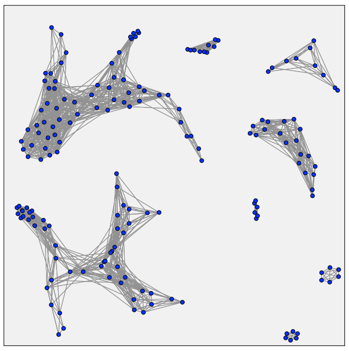
In this section, we will have a closer look on the properties of optimum schedules. Figure 6.4 shows the average slot numbers of the schedules that were generated with the ILPs and CPs described in this thesis. The scheduling problems have been solved to optimality for most of the transmission sets. If neither the ILP nor the CP was able to compute the optimum solution within the given time, then the best of both solutions

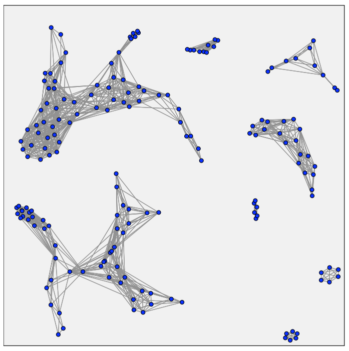

(a) Unit Disk Graph (88)

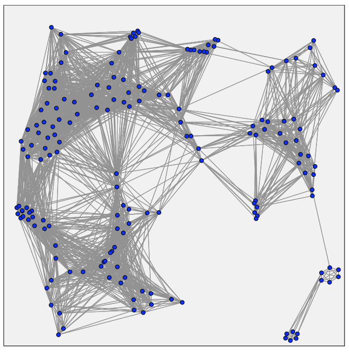

(b) Conflict Graph Vertices


(c) SINR Fixed (5142)


(d) SINR Variable (3612)


(e) Distance-2 Fixed (1302)


(f) Distance-2 Variable (1050)


(g) Sigma 2.16 (2554)

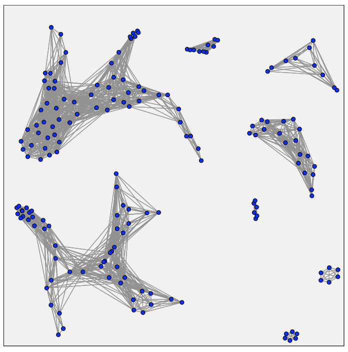

(h) Two-Hop (1516)

Figure 6.3: Graph-based conflict graphs vs. SINR-based conflict graphs

was used. More information on the transmission sets which could not be solved to optimality and the quality of the resulting schedules can be found in Section 6.3.6. By all means, we can assume that the schedules which we analyze in this chapter are almost optimal if not optimal. At this point, we only want to analyze the asymptotical behavior of optimum schedules for fixed and variable transmission powers, independently of each other. They will be compared with each other later in Section 6.3.4.

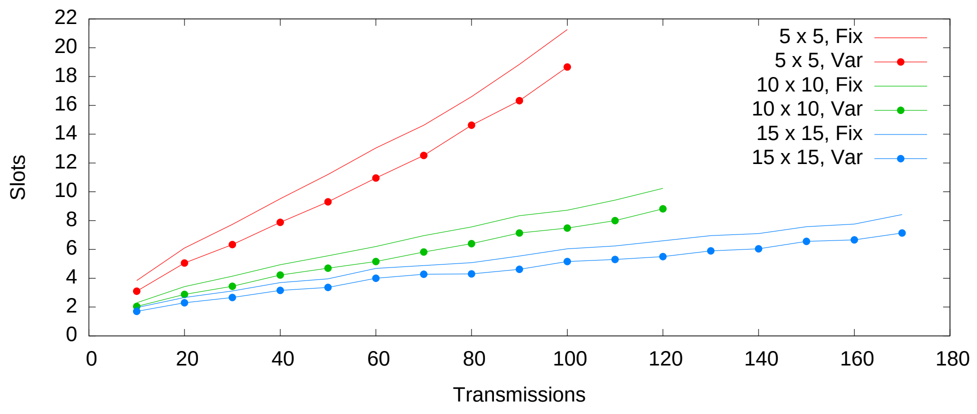


Figure 6.4: Optimum solutions fixed power vs. optimum solutions variable power

Probably the most outstanding property of the graphs is, for fixed dimensions, the almost linear dependence of transmission number and slot number in the corresponding (almost) optimal schedule. Especially for the transmission sets with dimension 5×5 it seems that the slot number increases more than linear for transmission numbers greater than 70. However, this is obviously only an artifact of the non-optimality of some of the schedules and can be ignored at this point. It is not surprising that the slot number of an optimal schedule decreases on average, when we consider the same number of transmissions distributed over a bigger area. Actually, we would have expected that the number of necessary slots is linear in the density of the transmissions, meaning transmissions per unit area. However, in our experiments the slot number of an optimal solution decreased almost exactly with $A^{-3/4}$ instead of A^{-1}, A being the available area. This can bee seen in Figure 6.5 where the x-value is given by *transmissions* / $A^{-3/4}$. We assume that this behavior is causes by some kind of border effect, which is proportional to $A^{1/4}$. For example, the bigger the area, the more receivers lie near the border of the test area and thus have less interfering senders around them.

One property of the optimal solution, which maybe does not sound too interesting at the first moment, is the spread of optimal solutions. However, this explains why the median running time of scheduling algorithms is relatively meaningless. This becomes important in connection with so-called hardness peaks, which provide further insights in the complexity of a problem. Hardness peaks are not covered in this thesis, but further information can be found in [9]. We will only consider the transmission sets

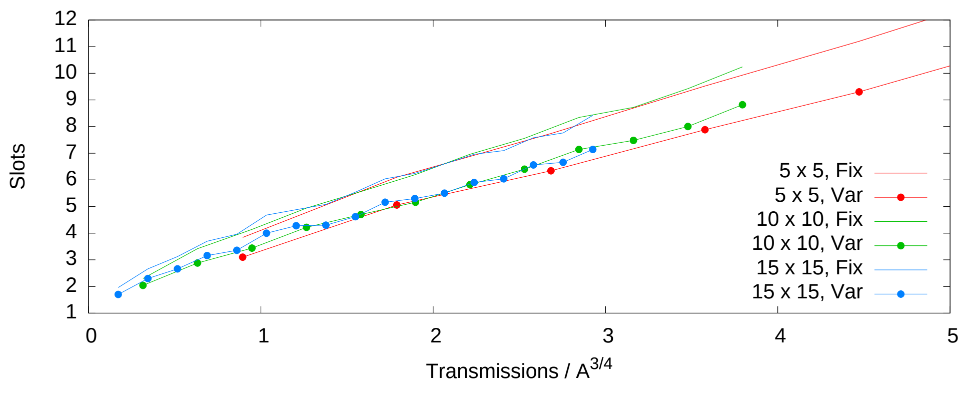


Figure 6.5: Dependence of the optimal solution on test area

with dimensions 5×5 and fixed transmission powers. Choosing a different class of transmission sets would yield similar results.

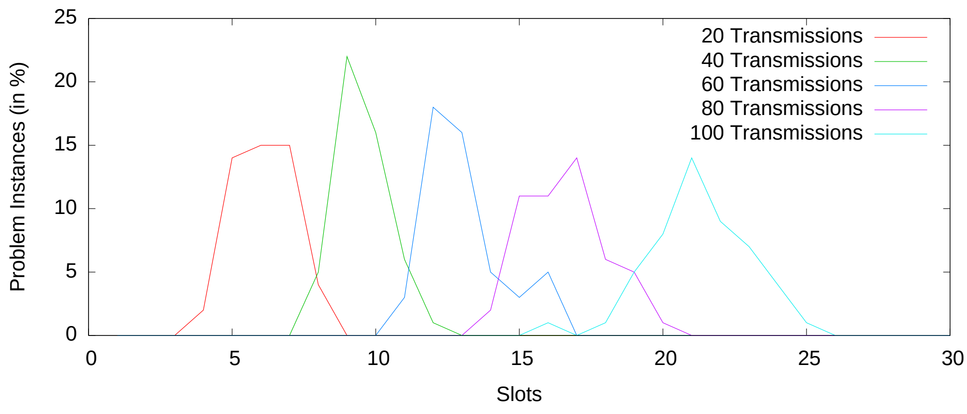


Figure 6.6: Spread of optimal solutions in dependence of transmission number

Figure 6.6 shows the distribution of the optimal solutions for 20, 40, 60, 80, and 100 transmissions. For every number of transmissions we get a definite peak. Probably, we would see some kind of Gauss distribution if the resolution were high enough. There are two main aspects that one should note. First, the optimal solutions are spread so widely that no slot number is optimal for more than 25% of the transmission sets. Second, the offset between the different peaks seems to be about proportional to the number of transmissions of the corresponding peak. This second observation can be refined by looking at Figure 6.7.

Here, the slot number is divided by the number of transmissions. As expected, most of the peaks overlap. The two outliers, 20 transmissions and 40 transmissions, can be

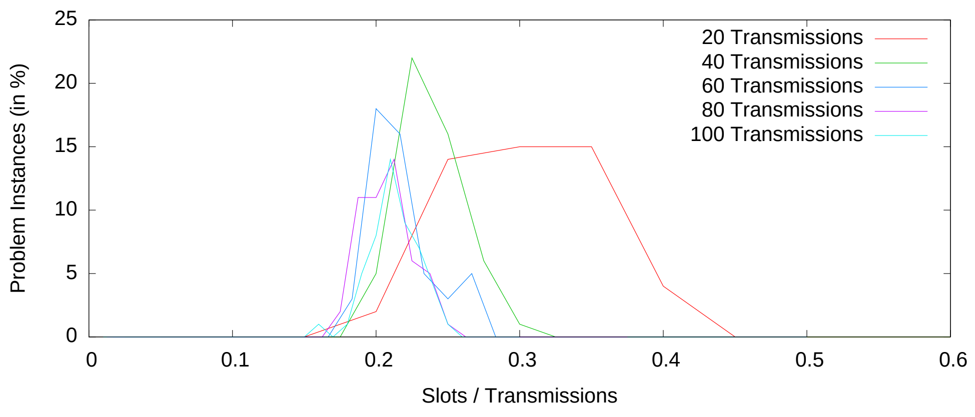


Figure 6.7: Normalized illustration of optimal solution spread

easily explained: The test area is probably not covered well enough. Even if we have, for instance, only two transmissions on a huge area, it could happen that those two transmissions are next to each other and occupy two slots. Something similar seems to happen with the transmission sets with only 20 and 40 transmissions The remaining transmission sets with 60, 80, and 100 transmissions overlap almost exactly and we can read from the figure that we need about one slot per 5 transmissions.

6.3.4 Variable Power vs. Fixed Power

Throughout this thesis, we distinguished between networks with variable transmission powers and networks with fixed transmission powers. It turned out that fixed transmission powers are much easier to handle than variable powers. In this section, we analyze the advantages that one can achieve by allowing variable transmission powers. Figure 6.4 in Section 6.3.3 already gives a good impression on the absolute difference between optimum slot numbers for schedules with fixed and variable powers. The relative differences are shown in Figure 6.8.

On average, it seems that optimal scheduling with fixed powers requires about 15% to 20% more slots than with variable powers. This ratio seems to be almost constant independently from the number of transmissions. Only for the data sets with dimensions 5×5 it looks as if the ratio would decrease. However, in this case the data are not completely accurate for more than 60 transmissions because many schedules could not be solved to optimality. Therefore, we assume that this ratio should also approach some fixed value if one uses a sufficient number of samples and solves every problem to optimality.

It is questionable whether it is worth to expend the additional effort of scheduling with variable powers if one can only be about 15% faster. But the time savings are probably

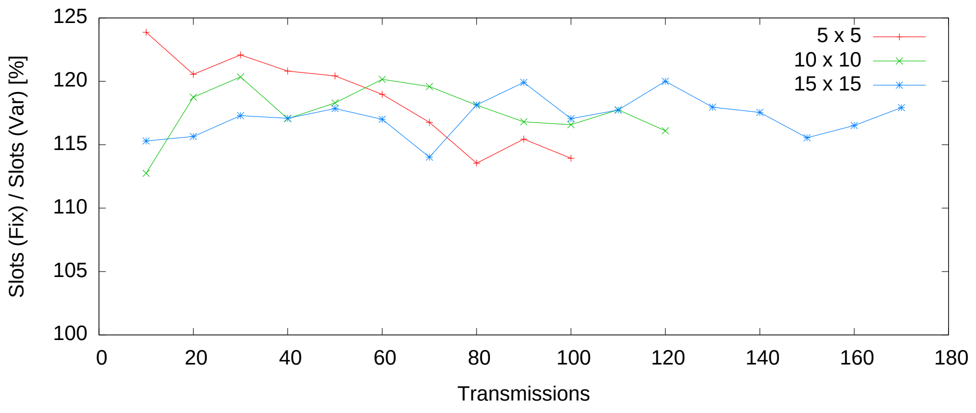


Figure 6.8: Percental disadvantage of fixed power scheduling

by far not the most important advantage of using variable powers. As already stated, saving power is one of the most substantial issues in algorithm design for wireless sensor networks. So let us have a closer look on the power consumption. For fixed transmission powers the case is clear: we always use maximum power. In the case of variable transmission powers, it seems in the first moment to be plausible that we also need high transmission powers for optimal schedules because we pack as many transmissions as possible in every time slot. Amazingly, it looks as if we can save a lot of energy by using variable transmission powers. Figure 6.9 shows the average transmission powers used for the time-optimal schedules which we generated with the CPs and ILPs.

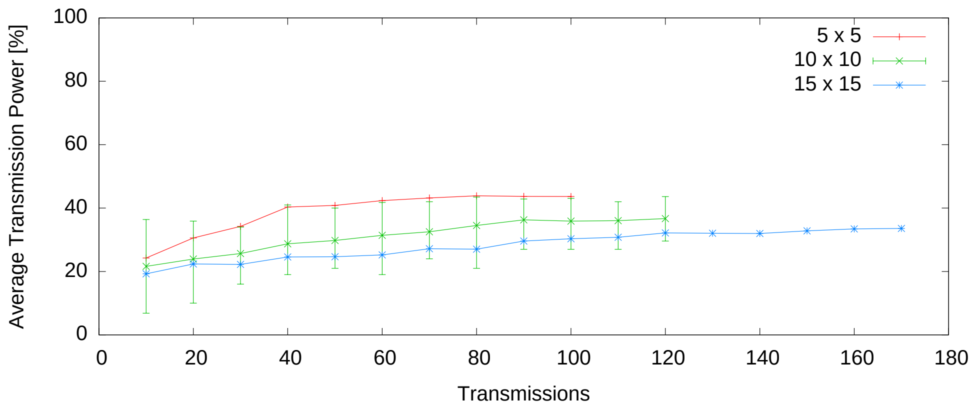


Figure 6.9: Average transmission powers (time-optimal schedule, variable power)

The average transmission power is always way below 50%. This means that we are not only faster by using variable transmission powers, but we also can save more than 50% transmission energy and thus we possibly are able to double the lifetime of the wireless network. Note that neither the CP nor the ILP was optimized for producing power-efficient schedules. So there is a good chance that we could do even better. The figure also shows the maximum and minimum of the average transmission powers as errorlines for the data with dimensions 10 × 10. Of course, this spread is the bigger the less transmissions are involved. Interestingly, even the maximal average powers are only about 40% of the maximum transmission power.

6.3.5 Quality of Heuristic and Lower Bound

As said before, the scheduling problem is NP-complete for fixed transmission powers and believed to be NP-complete for variable transmission powers. To the best of our knowledge, there is also no exact algorithm for scheduling with variable powers available, which has a good average running time. For this reason, algorithms that give good upper and lower bounds on optimal solutions are of great importance. In this chapter, we want to analyze the quality of the upper bound computed by the scheduling heuristic and the lower bound given by the maximum clique of the conflict graph in comparison to the solutions given by the ILPs and CPs.

Figure 6.10 and Figure 6.11 show the average slot numbers for fixed and variable transmission powers, respectively. Obviously, both bounds are pretty tight. Especially the maximum clique almost exactly predicts the number of necessary slots. The discrepancy for high transmission numbers, in particular for more than 60 transmissions on an area of 5 × 5 transmission radii, can again be explained by the non-optimality of some solutions because the instances were to hard to solve within the given time. For an unbiased interpretation, one should only consider the data points for which all instances could be solved to optimality (cf. 6.3.6).

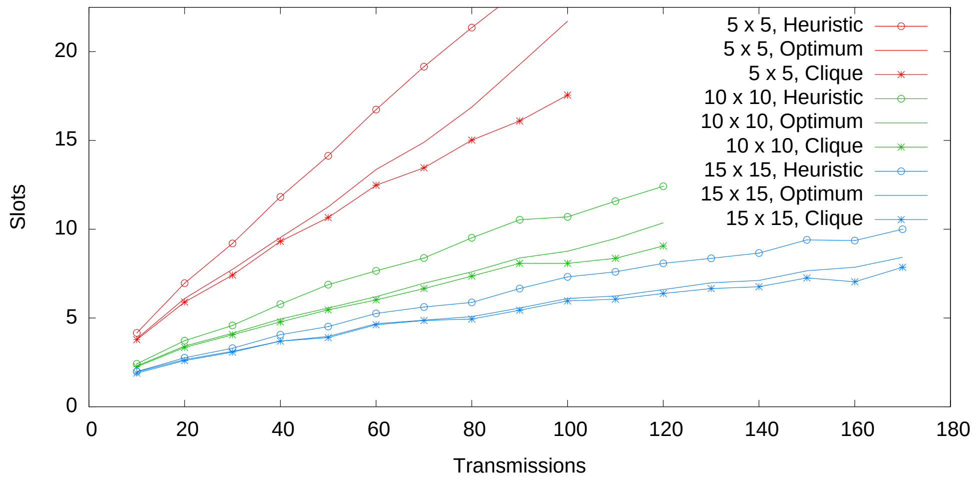


Figure 6.10: Quality of heuristic and lower bound (fixed power)

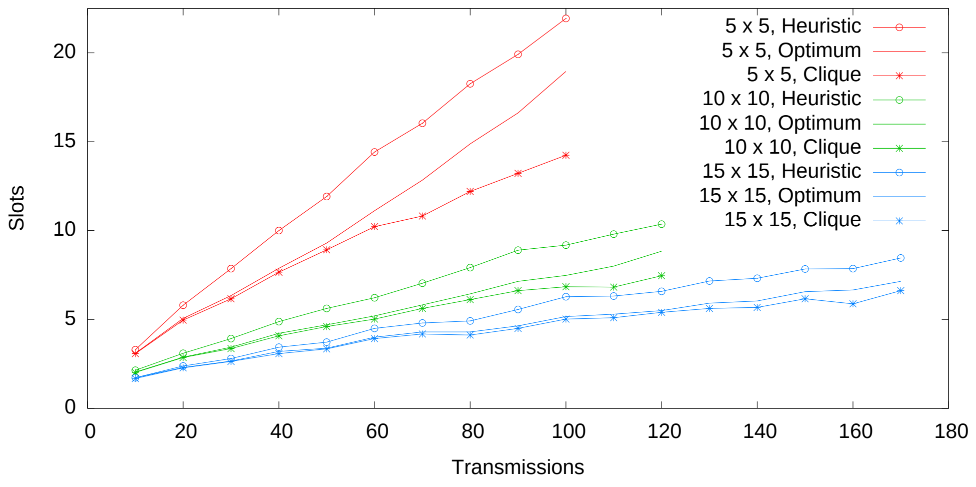


Figure 6.11: Quality of heuristic and lower bound (variable power)

The ratio of the upper and lower bounds to the ILP and CP solution is shown in Figure 6.12. In order to improve the comparability, we chose to use the transmission density (with rectified border effects) on the x-coordinate. The heuristics for fixed and variable power seem to behave equally well, with an average performance ratio of 1.25 or better for the considered data sets. The lower bound is even tighter and does not underestimate the optimal value by more than 6 percent on the average within the depicted range.

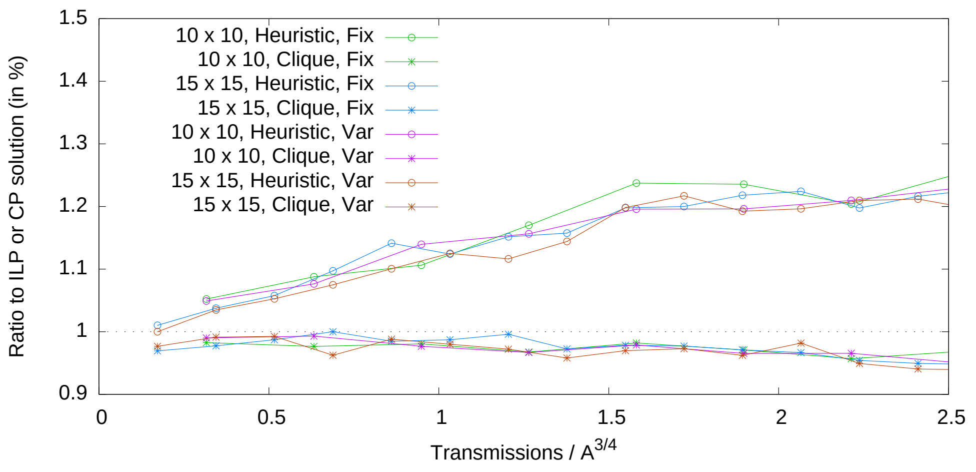
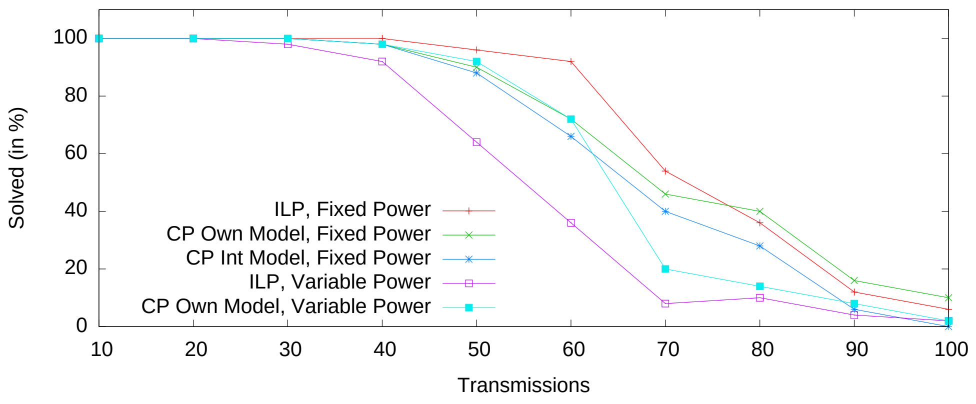


Figure 6.12: Performance ratios of heuristic and lower bound

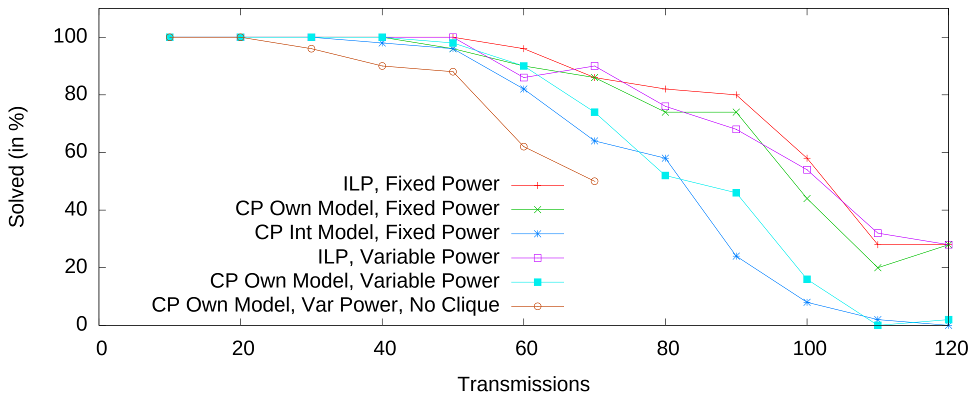
6.3.6 Performance

We already mentioned, that some of the problem instances have been too big to be solved to optimality. In the next two sections, we will examine which problems could actually be solved completely and how good the CP and ILP behaved in the other cases. Figure 6.13 depicts the percentages of problems that could be solved in 180 seconds, using the different methods. Considering the CP, we distinguish two different methods. *CP Int Model* is solely based on Gecode standard propagators and branchings, whereas in *CP Own Model*, we implemented our own SINR propagator and our own branching (cf. 4.2.1). As we can take from the figure, the own model performed substantially better than the standard implementation. Probably, this is mainly thanks to the improved branching. For big problem sizes, in most cases, the ILP was able to solve more problems than the CP. In our opinion, the reason for this is that the ILP is not so dependent on the input order. In every step of the ILP, the solver can choose one arbitrary variable to be adjusted, whereas in the CP, a variable assignment is only revised if it has been shown that there exists no feasible solution with this assignment. Two possibilities to avoid this problem are the use of random restarts and the use of limited discrepancy search, as described in Section 4.2.1.3. On an area of 5×5, the ILP with variable power performed surprisingly bad. One explanation for this could be the relative high slot number of an optimum solution for those instances. Possibly, the ILP is more dependent on the number of slots than the CP. On the other hand, the performance of the CP seems to depend strongly on the number of transmissions. Even on an area of 15×15, where the transmission density is rather small, the performance of the CPs went down for more than 100 transmissions. Again, this probably is due to the fixed order in which the Gecode branch-and-bound-search considers the variables.

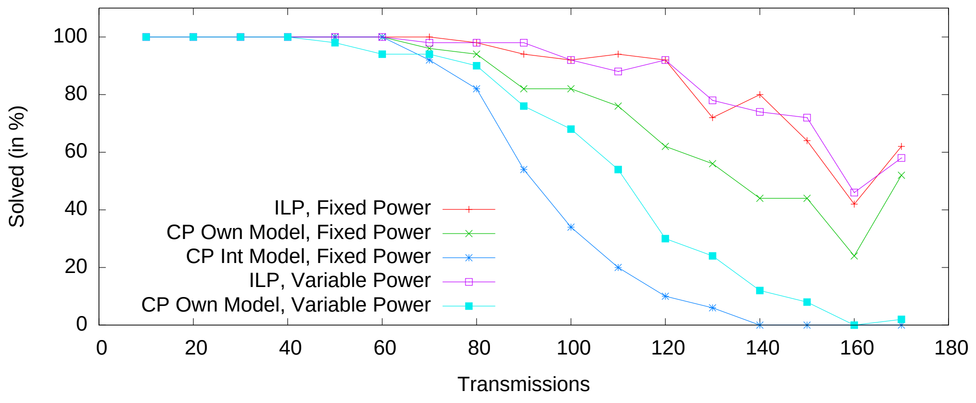
Usually Gecode was faster to find an optimum solution, while CPLEX was faster in proving the optimality. In general, it takes much longer to prove the optimality of a solution than to find the solution. Thus, there is a good chance that many of the computed schedules have been optimal, even though the ILP and CP were not able to prove it.



(a) Dimension: 5x5, Timeout: 180 sec



(b) Dimension: 10x10, Timeout: 180 sec



(c) Dimension: 15x15, Timeout: 180 sec

Figure 6.13: Percentage of solved problem instances (timeout: 180 sec)

6.3.7 Solution Quality

In the last section, we saw that for big problem sizes many problems could not be solved to optimality. Now, we want to compare the best solutions found by the ILP and the CP within 180 seconds with each other, as well as with the solution of the heuristic. The average schedule length is shown for fixed transmission powers in Figure 6.14, and for variable transmission powers in Figure 6.15.

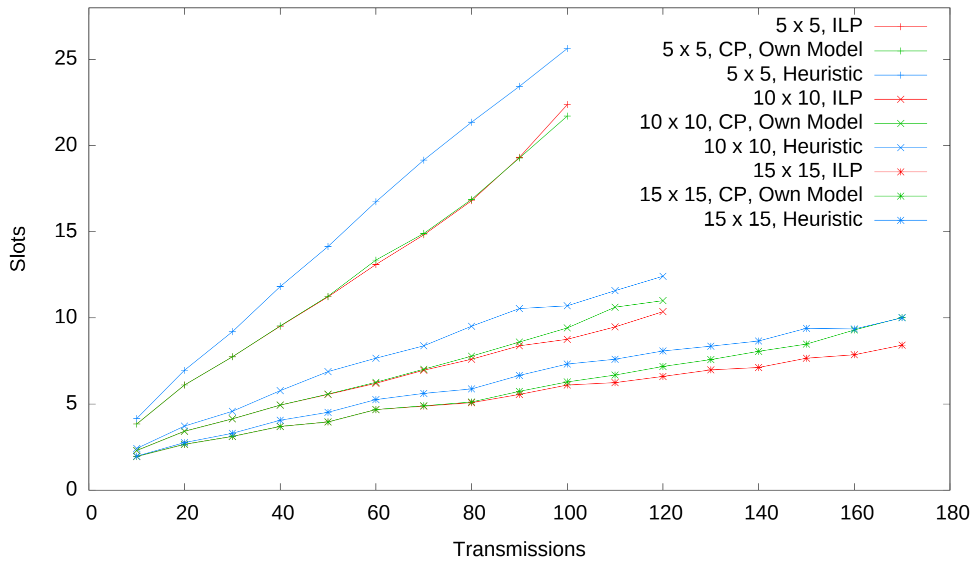


Figure 6.14: Average length of schedules found by ILP, CP, and heuristic (fixed power)

Up to about 80 transmissions, the solution quality of ILP and CP is almost identical. This is a strong indication that those instances have been solved to optimality for the most part. For 80 transmissions and above, the solution quality of the CP decreases. The number of slots of the final schedule is now mainly determined by the quality of the initial solution, as most of the variables cannot be revised within the given 180 seconds. Interestingly, for variable transmission powers, the MinPower heuristic outperforms the CP. The reason for this is simple: In our branching, we chose the next variable by its domain size and not by interference-based considerations. As a result, the schedule of the MinPower heuristic is much better than the initial solution of the CP. At this point, there is surely room for optimizations.

However, as we will see in the next section, randomization can help substantially to improve the solutions of the CP.

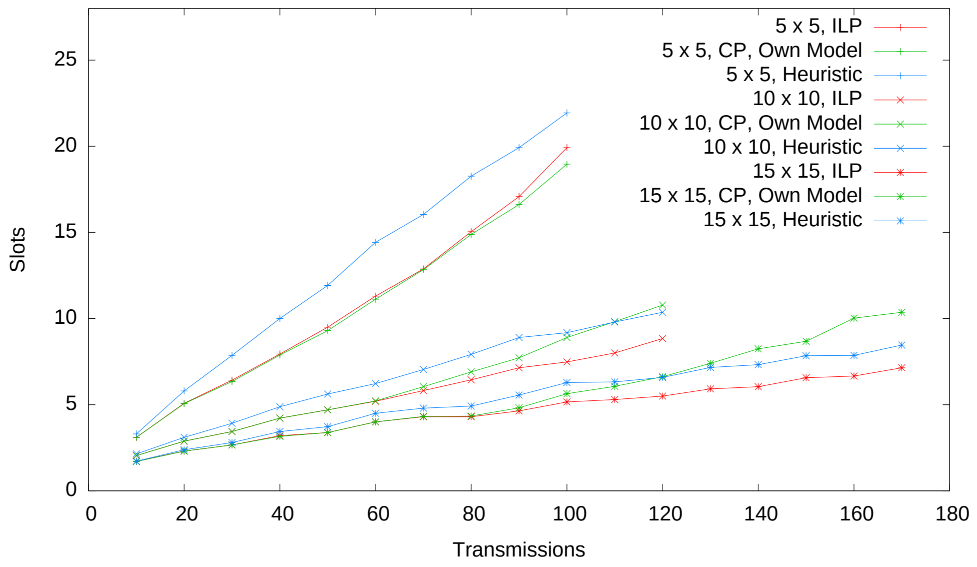


Figure 6.15: Average length of schedules found by ILP, CP, and heuristic (var. power)

6.3.8 Random Restarts

As we saw in the previous sections, constraint programming is very sensitive to the order in which the decision variables are assigned. This is in stark contrast to integer programming, where the variables can be updated in arbitrary order depending on the objective function. In this section, we want to analyze this dependence of constraint programming on the order of the input. For this purpose, we compare the results of a single CP execution with timeout 180 seconds against the best result achieved by 30 independent executions with timeout 6 seconds on the same input data in randomized order. For the comparisons, we use the 50 data sets with 100 transmissions on an area of 10×10 transmission radii.

In Figure 6.3.8, we compare for each data set the best solution (minimum number of slots) against the solution of the 3 minute run. In all but two of the 50 data sets, the random restart method behaves at least as good as the 3 minute computation. On the contrary, in most cases it is clearly superior to the single run, although every run had a timeout of only 6 seconds instead of 3 minutes. At first glance, this might look surprising. One reason for the good performance of the restarts is that there exist a relatively big number of optimum solutions. With every restart, there is a good chance that the assignment of the first transmissions allows for a good schedule. Even if some of the input orderings are bad, this does not make a difference thanks to the many restarts. In contrast, if in the single run some of the first transmissions are placed unfavorably, then all possible assignments of the remaining transmission have to be enumerated before those bad assignments can be undone.

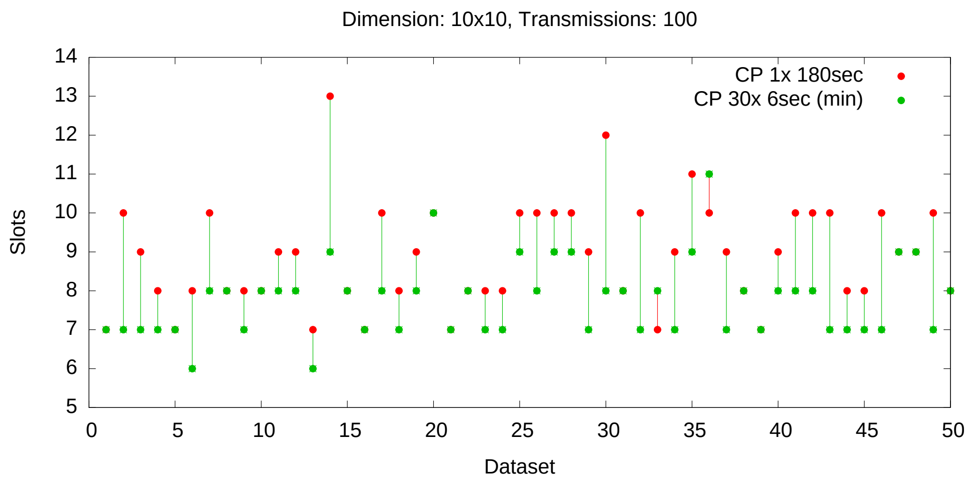


Figure 6.16: Improvement by using 30 random restarts (CP vs. CP)

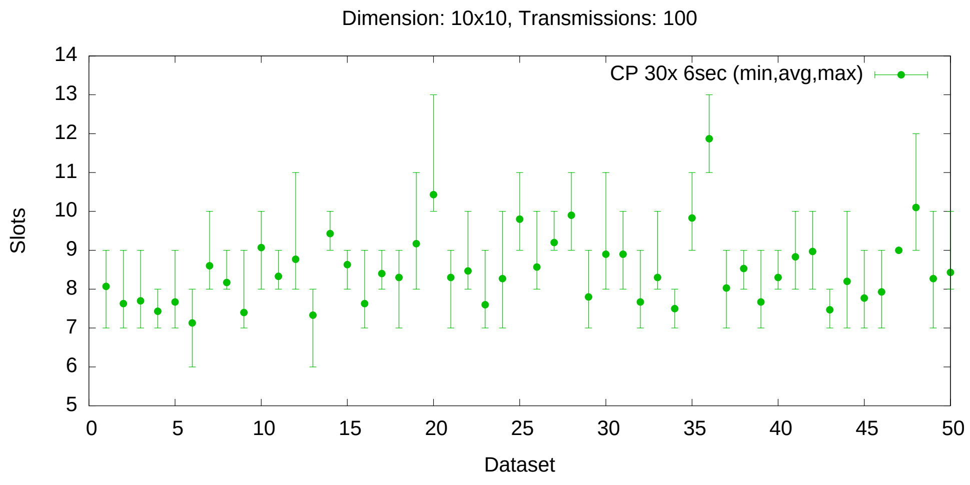


Figure 6.17: Range of solutions produced by 30 CP restarts

In order to give an impression on how the solutions spread over the different independent runs, Figure 6.3.8 gives an overview on best, worst, and average solutions. The span between best and worst solution is usually between two and three slots, which is actually over 20% of the slot number. Obviously, the performance is heavily influenced on the input order.

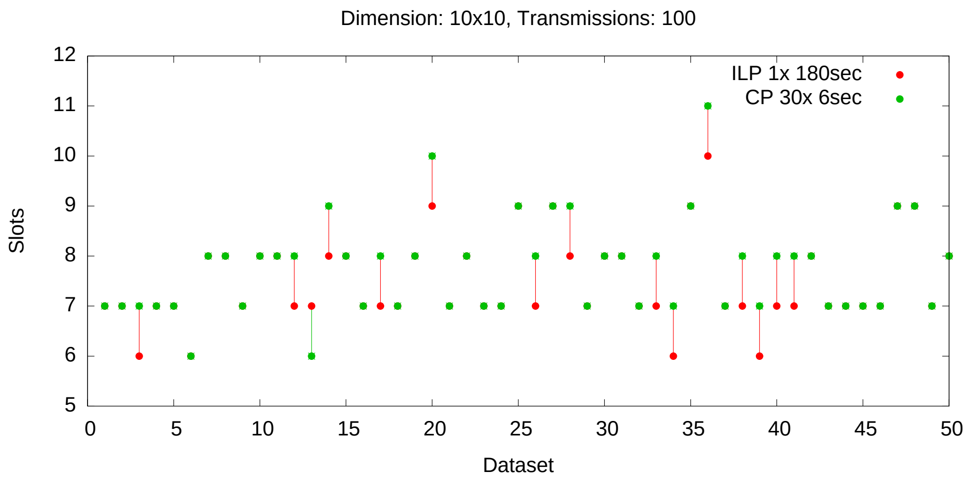


Figure 6.18: Improvement by using 30 random restarts (CP vs. ILP)

Last but not least, Figure 6.3.8 compares the best solution of the CP restarts against the best solution of a single ILP run. The ILP has still a slight advantage over the CP, but in most cases the CP now lies level with the ILP. We also did some tests with restarts of the ILP. But as expected, this did hardly bring an advantage for the ILP. This is not astonishing because the ILP is free to reassign an arbitrary variable in every step and thus, it is not so dependent on the input order.

A different approach to avoid the dependence of CPs on the input order is the use of limited discrepancy search [14].

6.4 Topology Control

The major part of the previous chapters has dealt with scheduling algorithms and with topologies that are fast to schedule. In the following, we will use the proposed heuristics and some other quality measures to compare several topology control algorithms (including the ones proposed in this thesis) to each other. We start with a visual comparison of the different topologies in Section 6.4.3. Subsequently, a series of 500 independent topology computations is analyzed, based on the quality measures described in the next section. The chapter is concluded with a study of the similarities between the different topologies.

6.4.1 Quality Measures

A common approach in most papers about topology control is to optimize the new algorithm for a certain structural property, and then to show that the new method outperforms existing algorithms. In this thesis, we want to make the comparison between different topologies as broad as possible. For this purpose, we used the common quality measures, as well as some new ones, to compare the topologies. What follows is a list of the used quality measures:

Edge number One of the goals of topology control is to reduce the number of links in the topology, in order to avoid the use of costly links and to simplify routing. Thus, the number of topology edges is of special importance.

Node degree If a node has many neighbors in the topology, then it is likely that a lot of traffic will be routed through the node. Therefore, it can be worthwhile to avoid nodes with high degree, to avoid congestions and to get a good load sharing. The average node degree, of course, has a similar meaning as the edge number.

Transmission radius Usually, the energy required for a transmission grows at least quadratically with sender-receiver-distance. Thus, the average transmission radius gives an estimate on the expected power consumption. Besides, a long-distance transmission causes strong interference at other receivers nearby. Hence, avoiding long links is one of the best tactics for topology control.

Transmission power Of course, it is also possible to measure the average transmission power that a sender needs to communicate with one of his neighbors (in the absence of interference). Minimizing this measure can help to improve the lifetime of the network. The maximum transmission power is not so meaningful, because it is often necessary to allow single long-distance links in order to preserve connectivity.

Hop stretch factor The average number of hops that is necessary for communication between pairs of senders influences the network performance in several ways: The more hops are necessary, the more nodes have to relay the messages. Thus, a higher average number of hops makes congestions more likely and increases the network latency. The average hop stretch factor is defined by the quotient of average number of hops necessary in the new topology and average number of hops necessary in the original topology. The maximum hop stretch factor is defined analogously.

Power stretch factor The ultimate goal of topology control is power conservation. Let us assume that communication is equally likely between every pair of nodes. The average power stretch factor tells us, how much more power, on average, is needed for a power-optimal transmission in the new topology compared to a power-optimal transmission in the original topology. Of course, this does neither consider interferences, nor the possibility of transmission failures. As we will see later, we usually can discard most of the network links without getting a *maximum* power stretch factor of more than 1.2.

Distance stretch factor Several algorithms for the computation of distance-spanners have been proposed in the literature. Like hop spanners, distance spanners can be easily computed using local topology control algorithms. This makes the resulting algorithms well applicable for use in wireless sensor networks. The distance stretch factor is defined analogously to hop stretch factor and power stretch factor.

Conflict graph edges The number of edges in the conflict graph of the topology is an excellent measure for the interference that we have to expect. It tells us, how many pairs of links cannot be used concurrently without transmission failures.

Length of an optimum schedule One very interesting property for classification of a topology is the length of an optimum schedule. It tells, how many slots would be required to schedule all links of the topology in a time-division-multiple-access (TDMA) fashion. The shorter the schedule is, the higher is the expected throughput. Using such a TDMA approach, transmission failures could be almost ruled out. Unfortunately, the computation of optimum schedules is too time-consuming to be performed on hundreds of topologies. Therefore, we used the aforementioned scheduling heuristics to get upper bounds on this measure.

Conflict graph clique A maximum clique of the conflict graph gives a good approximation on the length of a time-optimal schedule and provides a lower bound on such a schedule. However, the computation of a maximum clique is NP-complete, therefore it might be better in most cases to use some fast heuristic to determine a big clique in the conflict graph.

Node coverage The node coverage is an interference measure proposed in [3]. The coverage $\mathrm{Cov}(u, v)$ of an edge (u, v) is defined as the number of network nodes that are affected by a communication between nodes u and v. Further information is given in Section 5.1.2.

Interference This is our interference measure discussed in Section 5.2.2. In contrast to most interference measures proposed before, it takes the sender-receiver distances into account. This means for a transmission $t = (u, v)$, that the interference caused by other senders is weighted depending on the distance between u and v, and depending on the distance between v and the interfering senders.

6.4.2 Test Data and considered Topologies

We chose to perform our experiments on networks with 200 nodes, which are randomly distributed over an area of 7×7 transmission radii. Most of the topology control algorithms could be easily used on much bigger networks, but the computation of some of the quality measures is rather time consuming. For example, the optimum distances which are needed for the stretch factors are computed by a variation of the Floyd-Warshall algorithm which has time complexity $O(n^3)$. However, as we will see in the next sections, the topologies are big enough to allow for a good comparison of the topologies.

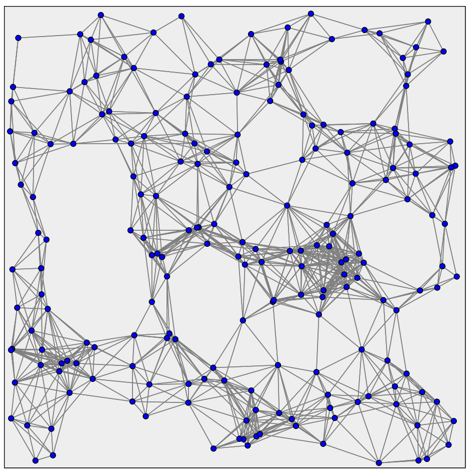
In our experiments, we compared the following topologies: Unit Disk Graph (Section 3.1), Gabriel Graph and Relative Neighborhood Graph (Section 5.1.1), LIFE and LISE (Section 5.1.2), MaxSINR and MinPower (Section 5.2.1), Interference, Random, and Distance. Interference is an abbreviation for the MinInterference algorithm from Section 5.2.2. Random and Distance work on the same principle as Interference, with the difference that Random chooses in every step a random edge from all edges that are not marked as unnecessary and Distance chooses the shortest edge from all edges that are not marked as unnecessary. The names of these algorithms will be highlighted with a capital first letter, to make them distinguishable from the remaining text. All topologies but Unit Disk Graph, Gabriel Graph, and Relative Neighborhood Graph, are used in three variations: to produce a connected topology, to produce a hop-3-spanner, and top produce a power-1.2-spanner. Note, that LIFE originally was only intended to compute connected topologies, and LISE was intended for the computation of spanners. Nevertheless, the underlying principles can be easily extended to allow the computation of topologies with almost arbitrary graph properties. Of course, if LISE is adjusted to compute only a connected topology, it is no longer locally computable. But in the following, we are only interested in properties of the resulting topologies, and not in how efficiently they can be computed.

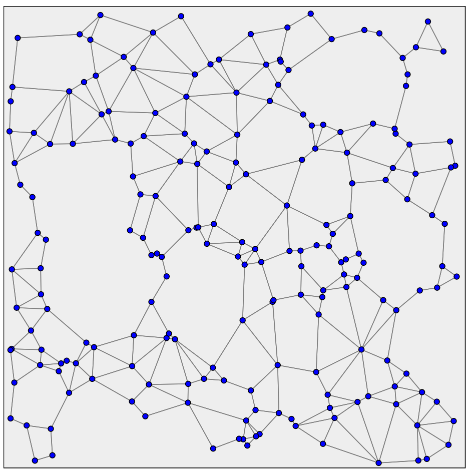
6.4.3 Visual Comparison based on a single Sample

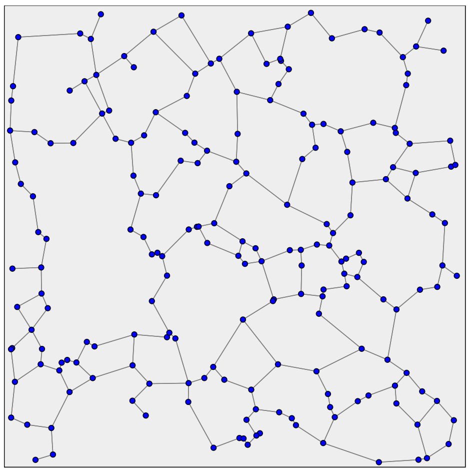
The goal of this section is to provide the reader with a visual impression of the input graphs and the different topologies. The number of edges of every topology is shown in the corresponding caption. This section contains three figures: The first figure shows the Unit Disk Graph of the given point set, Gabriel Graph and Relative Neighborhood Graph, and three Random topologies of which the first one preserves connectivity, the second one is a hop-3-spanner, and the third one a power-1.2 spanner. The remaining topologies are depicted in the following three figures. In Figure 6.20, all algorithms have been configured to generate connected topologies, and in Figure 6.21 and Figure 6.22 to produce hop-3-spanners and power-1.2-spanners, respectively.

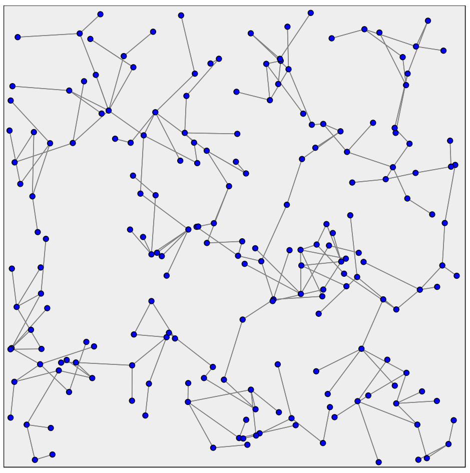
Let us first have a closer look at the Random topologies. Compared to all other topologies, they look more chaotic and, obviously, they consist of much longer edges. Due to this, they need much more edges to guarantee spanner properties. The MaxSINR, MinPower, Distance, Interference, and LIFE topology all look very similar, although they are motivated differently. Also, concerning the number of edges, they only vary slightly. We will see in the following sections that this is no coincidence. The topologies produced by the LISE algorithm always have the highest number of edges. This is not surprising, as LISE only tries to minimize the maximum transmission power and not the number of edges.

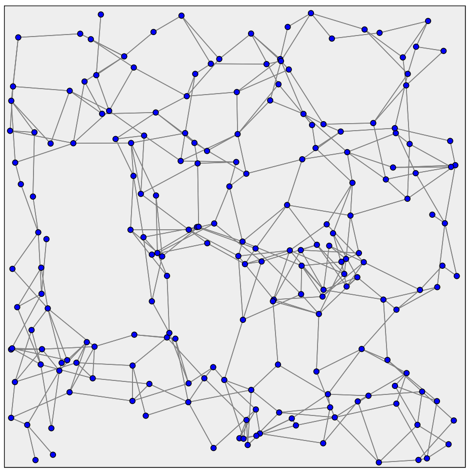
Now that we have a feeling about how the topologies look like, let us proceed to a detailed analysis of the algorithms based on several measures.


(a) Unit Disk Graph (2288)


(b) Gabriel Graph (704)


(c) Relative Neighborhood Graph (476)


(d) Random (Connected, 398)


(e) Random (Hop-3-Spanner, 720)

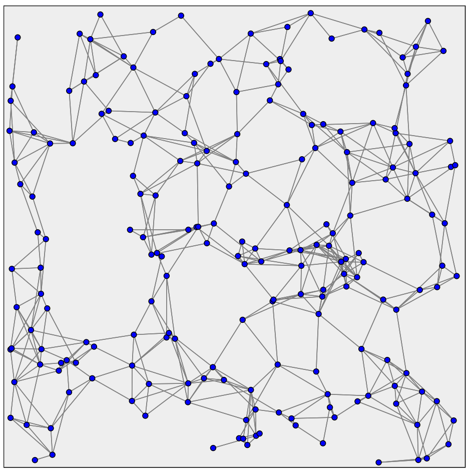
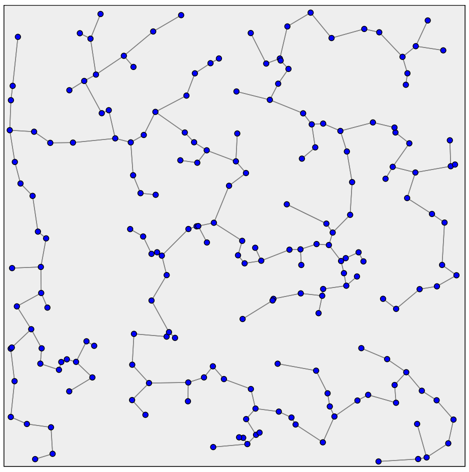
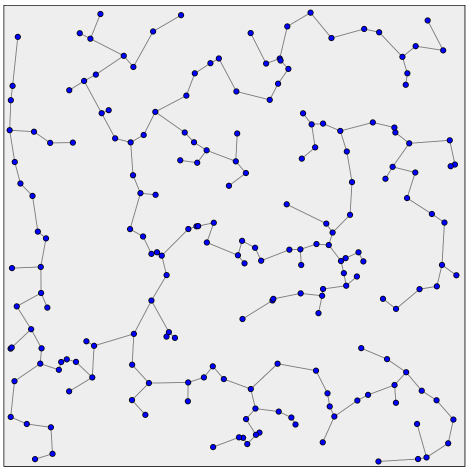

(f) Random (Power-1.2-Spanner, 1066)

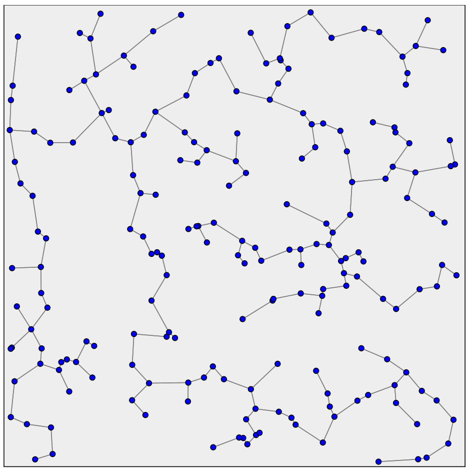
Figure 6.19: Comparison of different topologies



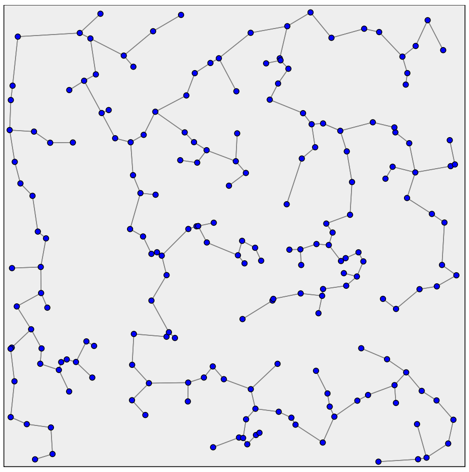
(a) MaxSINR (398)



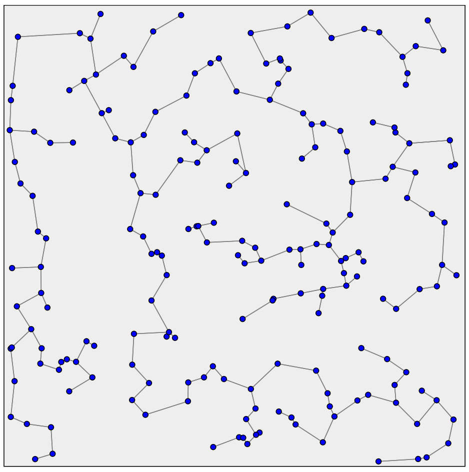
(b) MinPower (398)



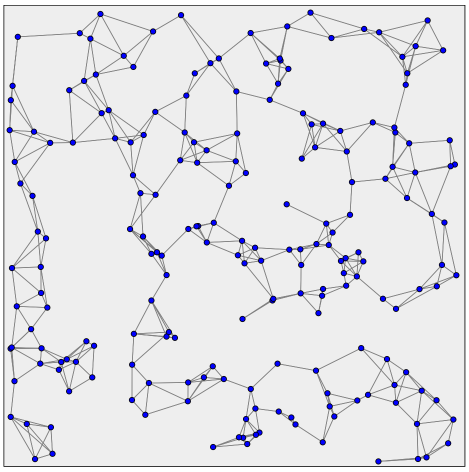
(c) Distance (398)



(d) Interference (398)

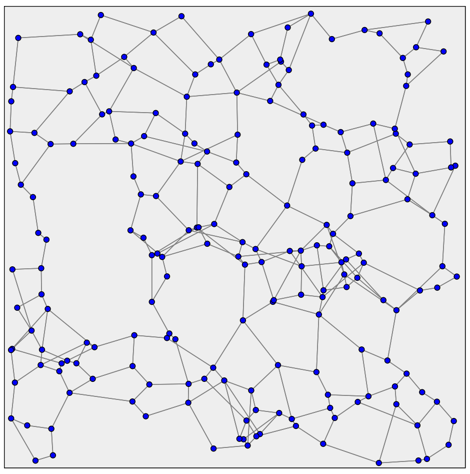


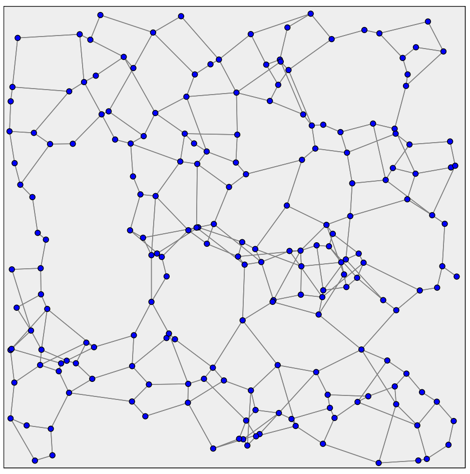
(e) LIFE (398)

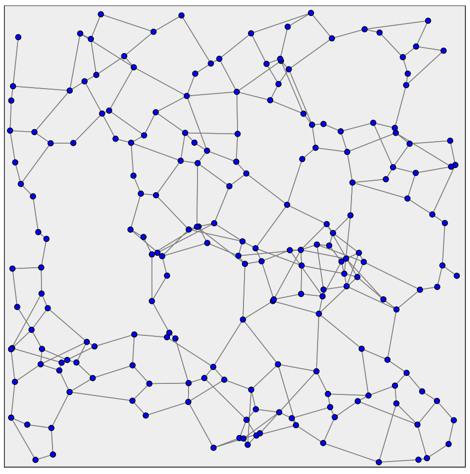


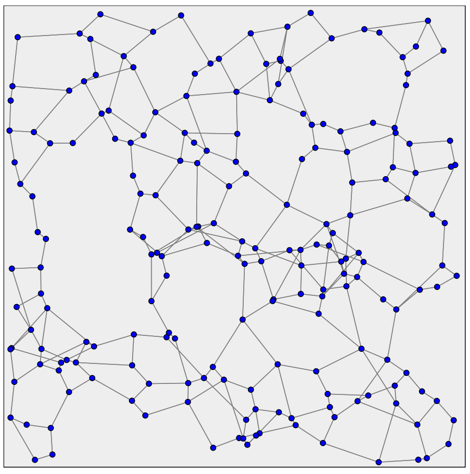
(f) LISE (922)

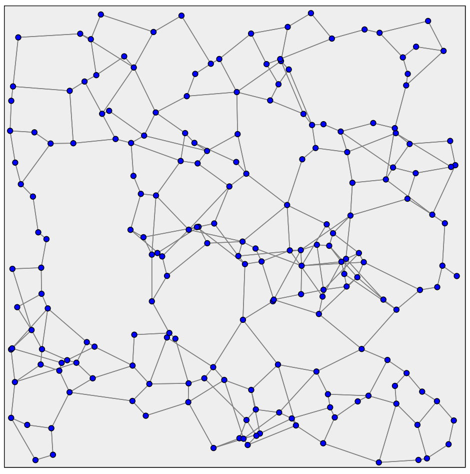
Figure 6.20: Comparison of different topologies (connected)


(a) MaxSINR (638)


(b) MinPower (642)


(c) Distance (634)


(d) Interference (632)


(e) LIFE (646)

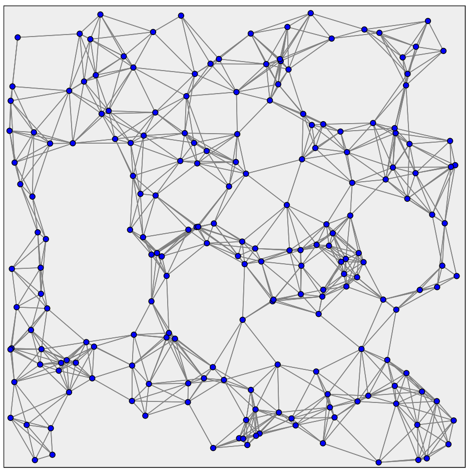
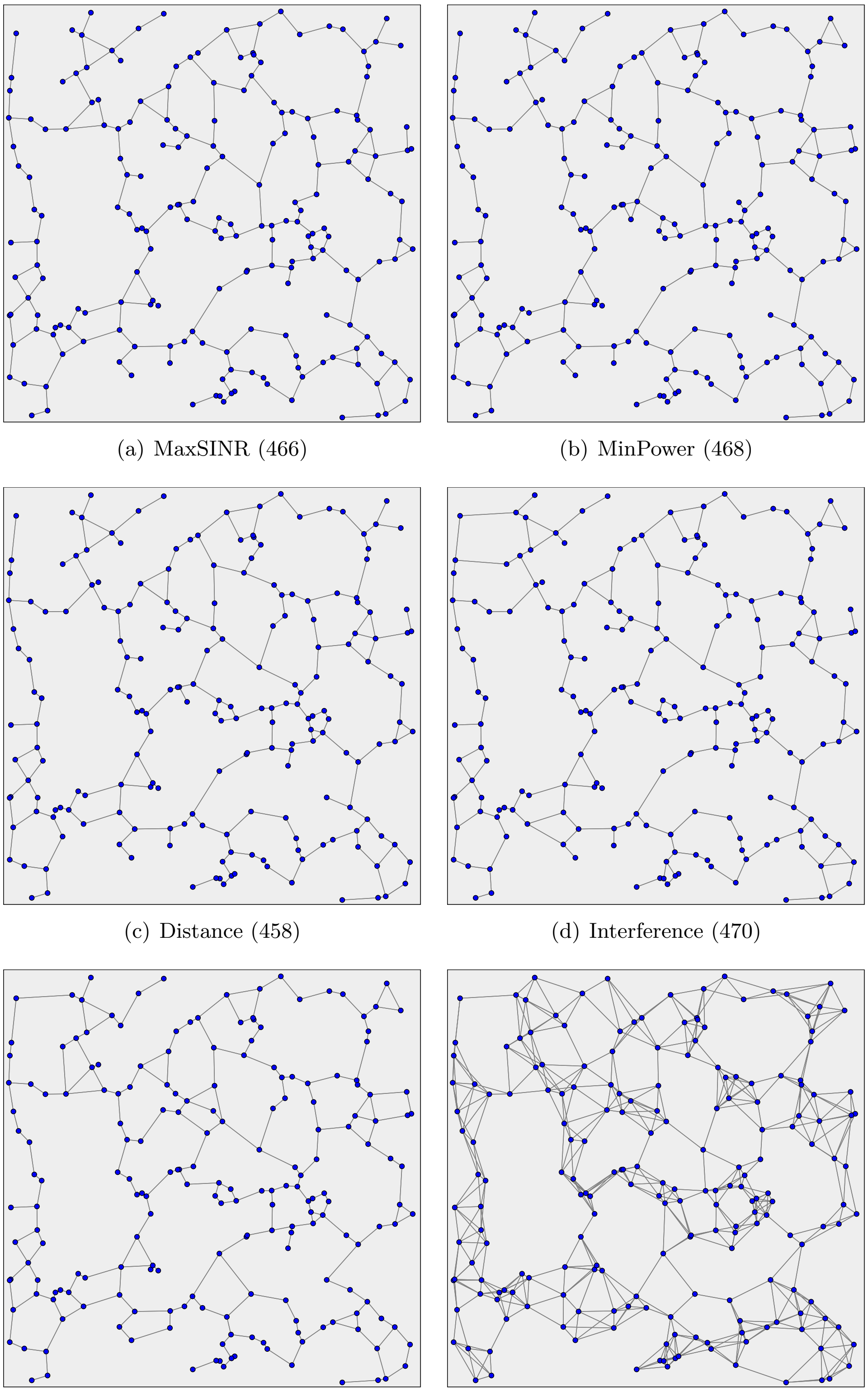

(f) LISE (1720)

Figure 6.21: Comparison of different topologies (hop-3-spanner)



(a) MaxSINR (466)

(b) MinPower (468)

(c) Distance (458)

(d) Interference (470)

(e) LIFE (482)

(f) LISE (1228)

Figure 6.22: Comparison of different topologies (power-1.2-spanner)

6.4.4 Comparison based on a Series of Samples

For the results in this section, we computed topologies on 500 random graphs. Every graph was generated by placing 200 nodes randomly in an area of 7×7 transmission radii, and subsequent computation of the corresponding unit disk graphs. Those unit disk graphs have been used as input for the topology control algorithms. Finally, several measures have been determined on the resulting topologies. In the following tables, the averages of all those measures are depicted. We will briefly analyze some of them one by one.

Concerning the average edge number, MaxSINR, MinPower, Distance, and Interference perform about equally well, with Distance slightly in front. The Random topologies, although they are also designed to reduce the number of edges, are significantly worse on average. This can be seen as a justification for sophisticated topology control algorithms. Interestingly, most algorithms need less than 20% nodes additional to a minimum spanning forest in order to guarantee that the power stretch factor of all possible node pairs is bounded by 1.2. Compared to the better algorithms, a Random spanner needs more than twice the number of edges. Even concerning average node degree, average radius, and average power, most algorithms behave similar. The average power seems to be surprisingly low. This is a result of the strong dependence of signal strength and distance (the path-loss-exponent α was assumed to be 4).

Table 6.3 depicts upper and lower bounds of the length of a time-optimal schedule. In every column, the left values represent the results based on the SINR model with fixed powers, and the right values those based on the SINR model with variable powers. Again, the results can be seen as a motivation for using variable transmission powers. The results also show that, using topology control, the length of an optimum schedule of the topology can be drastically reduced. The lower bound on the schedule of the Unit Disk Graph is about five times the upper bound of a schedule Distance (Power-1.2-Spanner). If the network would be denser, this difference would be even bigger.

The average and maximum stretch factors are shown in Table 6.4. Note, that in this case *maximum* does not denote the maximum of all topologies, but the average of the maximums of every single topology. This makes the enormous stretch factors of the Random topologies even more surprising. Regarding stretch factors, LISE outperforms the other methods. This is logical, as LISE keeps much more transmissions. Surprisingly, the average power stretch factor is almost one for most spanner topologies, even for those that are only hop-spanners. Again, the sole exemption is the Random topology. The huge hop stretch factors and non-negligible power stretch factors discourage the use of spanning trees as network topologies.

At first glance, the almost constant value of the covered nodes (fix) measure might puzzle. However, this measure is defined as number of nodes covered by the two disk with radius 1 around both endpoints of an edge. This is independent of the edge length and thus, most edges are almost equally bad. Concerning the other interference measures, there is again no big difference between MaxSINR, MinPower, Distance, Interference and LIFE. .

	Edges (Avg)	Node Degree (Avg / Max)	Radius (Avg)	Power (Avg)
Unit Disk Graph	2258.41	11.30 / 20.88	0.66	16 %
Gabriel Graph	709.75	3.55 / 6.82	0.46	6 %
Relative Neighborhood Graph	481.48	2.41 / 4.01	0.39	4 %
Random (Connected)	398	1.99 / 5.69	0.66	16 %
MaxSINR (Connected)	398	1.99 / 3.78	0.34	2 %
MinPower (Connected)	398	1.99 / 3.72	0.34	2 %
Distance (Connected)	398	1.99 / 3.78	0.33	2 %
Interference (Connected)	398	1.99 / 3.72	0.34	2 %
LIFE (Connected)	398	1.99 / 3.71	0.34	2 %
LISE (Connected)	897.99	4.49 / 7.01	0.43	4 %
Random (Hop-3-Spanner)	717.98	3.59 / 7.12	0.68	18 %
MaxSINR (Hop-3-Spanner)	644.18	3.22 / 5.51	0.49	9 %
MinPower (Hop-3-Spanner)	644.35	3.22 / 5.49	0.49	9 %
Distance (Hop-3-Spanner)	642.70	3.21 / 5.52	0.49	8 %
Interference (Hop-3-Spanner)	646.19	3.23 / 5.52	0.49	9 %
LIFE (Hop-3-Spanner)	645.59	3.23 / 5.50	0.49	9 %
LISE (Hop-3-Spanner)	1725.14	8.63 / 12.90	0.58	11 %
Random (Power-1.2-Spanner)	1097.18	5.49 / 11.66	0.58	12 %
MaxSINR (Power-1.2-Spanner)	467.59	2.34 / 4.16	0.37	3 %
MinPower (Power-1.2-Spanner)	469.05	2.35 / 4.14	0.37	3 %
Distance (Power-1.2-Spanner)	458.06	2.29 / 4.08	0.37	3 %
Interference (Power-1.2-Spanner)	475.54	2.38 / 4.20	0.37	3 %
LIFE (Power-1.2-Spanner)	474.15	2.37 / 4.19	0.37	3 %
LISE (Power-1.2-Spanner)	1307.91	6.54 / 9.68	0.51	7 %

Table 6.2: Topology comparison based on graph properties and transmission power

	Conflict Graph Clique (Fixed / Variable)	Schedule Heuristic (Fixed / Variable)
Unit Disk Graph	149.05 / 142.81	343.99 / 303.32
Gabriel Graph	25.46 / 18.82	61.28 / 48.33
Relative Neighborhood Graph	14.14 / 9.80	32.04 / 24.07
Random (Connected)	25.96 / 22.87	57.22 / 49.09
MaxSINR (Connected)	10.28 / 6.72	19.39 / 14.60
MinPower (Connected)	9.90 / 6.77	20.32 / 13.74
Distance (Connected)	10.32 / 6.69	20.39 / 14.98
Interference (Connected)	10.08 / 7.13	20.42 / 14.70
LIFE (Connected)	9.84 / 7.13	21.12 / 15.07
LISE (Connected)	20.86 / 17.87	60.15 / 45.47
Random (Hop-3-Spanner)	47.90 / 43.71	110.21 / 96.14
MaxSINR (Hop-3-Spanner)	30.70 / 24.42	63.57 / 52.93
MinPower (Hop-3-Spanner)	30.38 / 24.49	63.71 / 52.48
Distance (Hop-3-Spanner)	30.46 / 24.48	63.56 / 52.64
Interference (Hop-3-Spanner)	30.12 / 24.38	63.91 / 52.96
LIFE (Hop-3-Spanner)	30.93 / 24.66	64.30 / 53.22
LISE (Hop-3-Spanner)	60.29 / 48.98	186.02 / 153.72
Random (Power-1.2-Spanner)	61.18 / 53.54	136.69 / 117.18
MaxSINR (Power-1.2-Spanner)	13.01 / 9.61	29.02 / 21.72
MinPower (Power-1.2-Spanner)	13.00 / 9.64	29.40 / 21.63
Distance (Power-1.2-Spanner)	12.50 / 9.29	28.25 / 20.93
Interference (Power-1.2-Spanner)	13.08 / 10.08	29.75 / 22.14
LIFE (Power-1.2-Spanner)	13.39 / 10.05	30.08 / 22.39
LISE (Power-1.2-Spanner)	34.69 / 29.25	111.06 / 87.84

Table 6.3: Topology comparison based on schedule length (lower and upper bound)

	Distance (Avg / Max)	Hop (Avg / Max)	Power (Avg / Max)
Unit Disk Graph	1.00 / 1.00	1.00 / 1.00	1.00 / 1.0
Gabriel Graph	1.09 / 1.74	1.80 / 5.52	1.00 / 1.00
Relative Neighborhood Graph	1.26 / 3.55	2.44 / 9.26	1.01 / 1.67
Random (Connected)	3.44 / 248.94	3.67 / 37.28	2921 / 56585061
MaxSINR (Connected)	2.56 / 30.61	5.63 / 60.39	1.78 / 19.12
MinPower (Connected)	2.87 / 37.46	6.23 / 68.68	2.17 / 31.74
Distance (Connected)	2.32 / 26.12	5.26 / 57.99	1.31 / 7.38
Interference (Connected)	3.24 / 51.79	6.56 / 75.05	3.40 / 90.41
LIFE (Connected)	2.73 / 35.16	5.73 / 61.72	2.14 / 39.20
LISE (Connected)	1.45 / 12.58	2.27 / 19.14	1.22 / 7.70
Random (Hop-3-Spanner)	1.27 / 68.60	1.34 / 3.00	1586 / 30914380
MaxSINR (Hop-3-Spanner)	1.14 / 2.46	1.64 / 3.00	1.01 / 2.22
MinPower (Hop-3-Spanner)	1.14 / 2.46	1.64 / 3.00	1.01 / 2.05
Distance (Hop-3-Spanner)	1.14 / 2.40	1.64 / 3.00	1.00 / 1.69
Interference (Hop-3-Spanner)	1.14 / 2.51	1.64 / 3.00	1.01 / 2.44
LIFE (Hop-3-Spanner)	1.14 / 2.62	1.63 / 3.00	1.01 / 7.43
LISE (Hop-3-Spanner)	1.03 / 1.85	1.18 / 2.74	1.00 / 1.01
Random (Power-1.2-Spanner)	1.11 / 3.50	1.30 / 5.19	1.00 / 1.19
MaxSINR (Power-1.2-Spanner)	1.35 / 5.84	2.78 / 15.34	1.00 / 1.18
MinPower (Power-1.2-Spanner)	1.35 / 5.75	2.76 / 14.91	1.00 / 1.18
Distance (Power-1.2-Spanner)	1.37 / 6.08	2.84 / 15.99	1.00 / 1.18
Interference (Power-1.2-Spanner)	1.34 / 5.67	2.75 / 14.80	1.00 / 1.18
LIFE (Power-1.2-Spanner)	1.35 / 5.78	2.75 / 14.99	1.01 / 1.18
LISE (Power-1.2-Spanner)	1.08 / 3.21	1.43 / 5.30	1.00 / 1.05

Table 6.4: Topology comparison based on stretch factors

	Covered Nodes (Fix / Var)	CG Edges (Fix / Var)	Interference (Fix / Var)
Unit Disk Graph	12.63 / 6.75	396.68 / 263.94	915.72 / 291.97
Gabriel Graph	11.75 / 3.29	64.36 / 32.67	312.27 / 35.52
Relative Neighborhood Graph	11.57 / 2.33	30.91 / 13.49	216.46 / 15.05
Random (Connected)	11.76 / 6.39	61.23 / 39.57	179.72 / 44.53
MaxSINR (Connected)	11.45 / 1.89	17.89 / 7.00	184.94 / 8.32
MinPower (Connected)	11.43 / 1.87	17.90 / 6.79	184.14 / 8.12
Distance (Connected)	11.56 / 1.85	17.77 / 6.93	116.62 / 8.26
Interference (Connected)	11.26 / 1.88	17.74 / 6.79	115.65 / 8.12
LIFE (Connected)	11.42 / 1.79	18.14 / 6.83	121.31 / 8.14
LISE (Connected)	11.56 / 2.86	62.46 / 33.60	302.99 / 36.26
Random (Hop-3-Spanner)	12.02 / 6.83	124.34 / 82.83	261.45 / 93.22
MaxSINR (Hop-3-Spanner)	11.97 / 4.04	70.41 / 36.44	279.43 / 39.88
MinPower (Hop-3-Spanner)	11.96 / 4.03	70.28 / 36.35	278.64 / 39.79
Distance (Hop-3-Spanner)	11.97 / 4.01	70.08 / 36.20	130.99 / 39.63
Interference (Hop-3-Spanner)	11.95 / 4.06	70.61 / 36.53	101.46 / 40.00
LIFE (Hop-3-Spanner)	11.95 / 3.99	71.13 / 36.77	116.55 / 40.28
LISE (Hop-3-Spanner)	12.01 / 5.05	221.72 / 138.41	449.06 / 151.12
Random (Power-1.2-Spanner)	12.19 / 5.40	155.15 / 91.91	583.74 / 100.95
MaxSINR (Power-1.2-Spanner)	11.55 / 2.17	26.34 / 11.37	212.20 / 12.82
MinPower (Power-1.2-Spanner)	11.53 / 2.17	26.66 / 11.47	212.01 / 12.92
Distance (Power-1.2-Spanner)	11.55 / 2.11	25.22 / 10.76	105.62 / 12.19
Interference (Power-1.2-Spanner)	11.50 / 2.20	27.11 / 11.77	97.10 / 13.24
LIFE (Power-1.2-Spanner)	11.55 / 2.14	27.11 / 11.72	105.03 / 13.18
LISE (Power-1.2-Spanner)	11.73 / 3.92	126.65 / 74.95	374.75 / 80.63

Table 6.5: Topology comparison based on interference measures

6.4.5 Similarities between the Topologies

In the previous sections, we realized that some of the algorithms, although they are defined differently, have very similar properties. For this reason, we want to analyze the similarities between the topologies in the following. Tables 6.6, 6.7, and 6.8 deal with the algorithms for connected topologies, hop-3-spanners, and power-1.2-spanners, respectively. Unit Disk Graph (UDG), Gabriel Graph (GG), and Relative Neighborhood Graph (RNG) are presented in all three tables to simplify comparison.

In the columns, the topology names are abbreviated by numbers. The corresponding name can be found out by looking at the topology names in the rows. The value v in row y and column x represents the average ratio of edges, which are both in topology x and topology y, to the edges, which are only in topology x. For example, we can take from Table 6.6, column 7 and row 6, that the connected MinPower topology contains, on average, 94% of the edges from an Euclidean minimum spanning tree. In contrast, row 4 tells us, that the connected Random topology only shares about 19% of its edges with the other topologies. It is no coincidence that this almost equals the ratio of edge number in a spanning topology to the average number of nodes in the Unit Disk Graph.

Topology	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(1) UDG	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(2) GG	0.31	1.00	1.00	0.34	1.00	1.00	1.00	1.00	0.98	0.64
(3) RNG	0.21	0.68	1.00	0.23	0.97	0.97	1.00	0.95	0.96	0.50
(4) Random	0.18	0.19	0.19	1.00	0.19	0.19	0.19	0.20	0.19	0.19
(5) MaxSINR	0.18	0.56	0.80	0.19	1.00	0.92	0.93	0.91	0.89	0.44
(6) MinPower	0.18	0.56	0.80	0.19	0.92	1.00	0.94	0.91	0.90	0.44
(7) Distance	0.18	0.56	0.83	0.19	0.93	0.94	1.00	0.90	0.91	0.44
(8) Interference	0.18	0.56	0.79	0.20	0.91	0.91	0.90	1.00	0.88	0.44
(9) LIFE	0.18	0.55	0.79	0.19	0.89	0.90	0.91	0.88	1.00	0.44
(10) LISE	0.40	0.80	0.93	0.43	0.99	0.99	0.99	0.99	1.00	1.00

Table 6.6: Percentage of common edges (Connected Topologies)

Tables 6.6, 6.7, and 6.8 also confirm that the MaxSINR, MinPower, Distance, and Interference topologies are extremely similar. In the case of connected topologies, they coincide in about 90% of the edges. The corresponding power-1.2-spanner topologies even coincide in about 95% of the edges. In our experiments, the MaxSINR and Min-Power algorithms did not bring any advantage in comparison to the simple Distance algorithm. Especially, against our hope, they did not produce topologies that can be scheduled faster than topologies produced by simpler algorithms. Therefore, the use of these methods does not seem to be worthwhile, even in cases where an initial time-consuming computation of a schedule can be tolerated.

Topology	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(1) UDG	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(2) GG	0.31	1.00	1.00	0.31	0.78	0.79	0.79	0.78	0.78	0.40
(3) RNG	0.21	0.68	1.00	0.21	0.70	0.71	0.73	0.69	0.70	0.28
(4) Random	0.32	0.31	0.31	1.00	0.31	0.31	0.31	0.31	0.31	0.32
(5) MaxSINR	0.29	0.71	0.94	0.28	1.00	0.88	0.90	0.86	0.82	0.34
(6) MinPower	0.29	0.71	0.95	0.28	0.88	1.00	0.92	0.87	0.84	0.34
(7) Distance	0.28	0.71	0.98	0.28	0.90	0.92	1.00	0.86	0.86	0.34
(8) Interference	0.29	0.71	0.93	0.28	0.86	0.87	0.86	1.00	0.81	0.34
(9) LIFE	0.29	0.71	0.93	0.28	0.83	0.84	0.86	0.81	1.00	0.34
(10) LISE	0.76	0.98	1.00	0.76	0.90	0.90	0.91	0.90	0.91	1.00

Table 6.7: Percentage of common edges (Hop-3-Spanners)

Topology	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(1) UDG	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(2) GG	0.31	1.00	1.00	0.43	0.99	1.00	1.00	1.00	0.98	0.50
(3) RNG	0.21	0.68	1.00	0.34	0.94	0.95	0.97	0.93	0.93	0.36
(4) Random	0.49	0.67	0.78	1.00	0.81	0.81	0.82	0.80	0.81	0.59
(5) MaxSINR	0.21	0.66	0.91	0.35	1.00	0.96	0.98	0.95	0.94	0.36
(6) MinPower	0.21	0.66	0.92	0.35	0.97	1.00	0.99	0.95	0.95	0.36
(7) Distance	0.20	0.65	0.92	0.34	0.96	0.97	1.00	0.94	0.94	0.35
(8) Interference	0.21	0.67	0.91	0.35	0.96	0.97	0.98	1.00	0.94	0.36
(9) LIFE	0.21	0.66	0.91	0.35	0.95	0.96	0.98	0.94	1.00	0.36
(10) LISE	0.58	0.92	0.99	0.71	1.00	1.00	1.00	1.00	1.00	1.00

Table 6.8: Percentage of common edges (Power-1.2-Spanners)

7 Final Remarks

In this thesis, we studied the problems of scheduling and topology control in wireless networks. Our considerations were based on the physically motivated SINR interference model. In this model, scheduling with fixed transmission powers is known to be NP-complete, and scheduling with variable transmission powers is also believed to be NP-hard. We gave constraint programming (CP) formulations and integer linear programming (ILP) formulations for the exact solution of the scheduling problem. Although we implemented several optimizations, most problems with more than 100 transmission still cannot be solved within reasonable time. Therefore, we analyzed heuristics for the scheduling problem, as well as methods to compute lower bounds on the length of optimum schedules. In our experiments, the schedules produced by the heuristics have only been slightly worse than the optimum schedules that were computed using the ILP and CP. For computing good lower bounds on the length of an optimum schedule, the maximum clique of the conflict graph proved to be very useful. The maximum clique problem is known to be NP-complete, but thanks to the (effective) locality of interference, there is some chance that it still can be solved in real-size networks. In cases where computation of a maximum clique is too time-consuming, one still can use heuristics for finding big cliques in the conflict graph.

The aforementioned methods have been used to study fundamental properties of scheduling in the SINR model. For example, it turned out that scheduling our randomly generated transmission sets with fixed transmission powers needed about 20% more time slots than scheduling the same transmissions with variable powers. Although we computed time-optimal schedules, which entails high interference, we mostly required less than 40% of the maximum transmission power to realize a successful transmission. We also compared the performance of the ILPs and CPs, but this is topic of the next section.

Concerning topology control, we studied the problem of computing a topology that can be scheduled efficiently. For this purpose, we extended our scheduling heuristics to topology control algorithms. Furthermore, we proposed a topology control algorithm that tries to minimize a new interference measure, which takes into account the signal quality of transmissions. We compared these algorithms and several established topology control algorithms to each other. It turned out that the best of those algorithms, although they are defined differently, compute very similar topologies. These topologies mainly consist of short edges. None of the algorithms was able to outperform the simple greedy algorithms that adds the shortest edge until the desired graph property is fulfilled. The additional effort of our MaxSINR and MinPower topology control algorithms did not result in an any advantages compared to this much simpler algorithm. The topology control algorithm that tries to minimize interference performed slightly worse than the one that greedily chooses the shortest edge. However, it can be used without any knowledge of distances or directions. Additionally, it probably would outperform the other algorithms in situations with obstacles, where the distance is not the dominating factor of transmission quality.

7.1 Mathematical Programming vs. Constraint Programming

In our opinion, ILPs and CPs both are appropriate tools for solving small scheduling problems to optimality. In situations with few transmissions and many slots, the CPs outperformed the ILPs. On the other hand, for problems with many transmissions, the ILPs performed much better. One of the reasons for this behavior was, that our branching heuristics was not optimal. But using Gecode's branch-and-bound search, even with a good branching heuristic, the initial variable assignment almost determines the quality of the final solution. This is because the branch-and-bound search only revises a decision when it was able to prove that the decision does not lead to a feasible solution. Thus, if the problem consists of hundreds of transmissions, most of the variable assignments will not be revised within reasonable time. Two ways out of this dilemma are the use of random restarts or limited discrepancy search. We checked out the first approach, and it led to significantly better solutions. Unfortunately, even though the best solution found this way is much better, none of both methods is able to sufficiently speed up the process of proving optimality. In most cases, Gecode was faster to find the optimum solution, and CPLEX was faster in proving optimality.

Constraint programming has the big strength that pretty much every combinatorial problem can be modeled. If the solution has to fulfill some property which is not directly modelable, one simply implements a new propagator that takes care of the condition. By combining the own custom constraints with the optimized constraints already offered by the CP solver, one can save developing time and gets an efficient tool to solve his problem. Moreover, the high level of abstraction of problem definitions makes it easy for people without much programming experience to solve their problems efficiently.

Both problems, scheduling with fixed power and scheduling with variable power, could be elegantly formulated as ILPs and MIPs with only $O(n^2)$ variables and constraints. In situations where such an efficient modeling is not possible, constraint programming is surely the tool of choice.

Lastly, we have to note that constraint programming can be combined with arbitrary methods to solve a problem. For example, some solvers allow to specify parts of the CP as LPs or ILPs. The issues that we encountered while solving our CPs were mainly caused by the use of Gecode's branch-and-bound-method, which is the standard approach of Gecode for solving constraint optimization problems. Other CP solvers might be based on different principles and thus be more efficient for this specific problems.

7.2 Outlook

The most interesting problem concerning scheduling in wireless networks surely is the complexity of scheduling with variable transmission powers. Although it is assumed that this problem is NP-hard, so far no one was able to proof it. There are also several possibilities how the results of this thesis could be extended or improved. For example, if we know the length of a time-optimal schedule, another interesting problem is to distribute the transmissions such to the time slots that the maximum or average of the transmission powers is minimized. Concerning the scheduling CP, there surely is room for optimizations by using better branching heuristics or techniques such as limited discrepancy search. Furthermore, it would be very interesting to evaluate the performance of the different topology control algorithms in situations where the signal strength does not only depend on the distance between sender and receiver. This would reflect reality better and should thus be analyzed. Our intention is, that in such situations our interference minimization algorithm outperforms other approaches. Actually, we believe that it would be even better to greedily choose simply the link that maximizes the signal strength instead of choosing the link which minimizes our interference measure. In the geometric SINR model used in this thesis, this equals the algorithm that chooses the links in increasing order of length. But in situations with obstacles, it is a completely different method, which is very simple and which has the potential to outperform most other methods.

8 Zusammenfassung *(German abstract)*

Der erste Teil dieser Arbeit beschäftigt sich mit dem Problem des Schedulings in Sensornetzen. Es werden mehrere Heuristiken vorgestellt, mit denen sich verhältnismäßig effizient gute obere und untere Schranken für die Anzahl der in einem optimalen Schedule benötigten Zeitslots bestimmen lassen. Hierbei wird sowohl der Fall fester Sendeleistungen, als auch der Fall, dass alle Sender ihre Leistung bis zu einer gewissen Obergrenze selbst regeln können, betrachtet. Im Falle fester Sendeleistungen ist bekannt, dass das Scheduling Problem NP-vollständig ist, und man geht davon aus, dass auch das Scheduling mit variablen Sendeleistungen NP-schwer ist. Daher ist die Suche nach einem exakten Algorithmus mit polynomieller Laufzeit für das Schedulingproblem relativ aussichtslos. Stattdessen werden in dieser Arbeit Methoden der mathematischen Optimierung (ganzzahlige lineare Programme, ILPs) und der kombinatorischen Optimierung (Constraint Programs, CPs) eingesetzt, um mittels hochoptimierter Solver für kleine Probleminstanzen Optimallösungen bestimmen zu können. Ferner werden zahlreiche Optimierungen untersucht, mit denen sich die CPs und ILPs erheblich beschleunigen lassen. Um die Methoden miteinander vergleichen zu können, wurden die ILPs auf Basis des kommerziellen ILOG CPLEX Solvers, und die CPs auf Basis des frei verfügbaren GECODE Solvers implementiert. Die vorgeschlagenen Heuristiken wurden mittels Java implementiert. Anschließend wurden umfangreiche Tests und Laufzeitmessungen durchgeführt, um einen Eindruck über die Qualitäts- und Leistungsunterschiede der einzelnen Verfahren zu erhalten, und um verschiedene Versionen der ILPs und CPs miteinander zu vergleichen.

Aufbauend auf den zuvor genannten Ergebnissen zum Scheduling werden diverse Fragestellungen im Zusammenhang mit Sensornetzen angegangen. So wird beispielsweise untersucht, wie stark das Scheduling durch den Einsatz variabler Sendeleistung im Vergleich zum Scheduling mit festen Sendeleistungen beschleunigt werden kann, und wieviel Energie sich hierbei einsparen lässt. Des Weiteren wird der Frage nachgegangen, wie realistisch graphenbasierte Interferenzmodelle im Vergleich zum physikalisch motivierten und als realitätsnah angesehenen SINR-Interferenzmodell sind.

Der zweite Teil der Arbeit beschäftigt sich mit Topology Control in Sensornetzen. Hierbei geht es um die Bestimmung einer Netztopologie, also einer Teilmenge aller möglichen Verbindungen zwischen benachbarten Sensorknoten, so dass bestimmte Netzwerkeigenschaften wie beispielsweise Zusammenhang oder Spanner-Eigenschaften erhalten bleiben, das Netzwerk aber gleichzeitig auf geringen Energieverbrauch oder ähnliches optimiert wird. In dieser Arbeit beschäftigen wir uns mit dem Optimierungsprob-

lem eine Topologie zu bestimmen, bei der die Kanten der Topologie möglichst effizient gescheduled werden können. Die Idee hierbei besteht darin, dass es aus Gründen der Energieeinsparung durchaus sinnvoll sein kann Sensorknoten nur in vordefinierten festen Zeitslots senden zu lassen. Auf diese Weise kann garantiert werden, dass es zu keinen Kollisionen bei den Übertragungen kommt. Hierdurch können letztendlich Mehrfachübertragungen vermieden werden. Da man lange Latenzzeiten im Netzwerk verhindern will, empfiehlt es sich die Topologie so zu bestimmen, dass die Anzahl der zum Schedulen aller Kanten benötigten Zeitslots möglichst gering ist. Es werden Topology Control Heuristiken zur Bestimmung derartiger Topologien vorgeschlagen, die wahlweise nur den Zusammenhang des Netzes sichern, oder aber zusätzlich Spannereigenschaften, z.B. für Energieverbrauch oder Distanzen im Netz, garantieren können. Diese Heuristiken sind aufgrund ihrer Komplexität für den Einsatz in realen Netzen weniger geeignet. Stattdessen sollen sie dazu genutzt werden, die Qualität einfacher Topology Control Algorithmen in Bezug auf Scheduling-Eigenschaften einzuschätzen.

Darüber hinaus wird ein einfacher Topology Control Algorithmus angegeben, der darauf abzielt die Interferenz in der erzeugten Topologie zu minimieren. Im Gegensatz zu den meisten existierenden Topology Control Algorithmen mit ähnlichem Ziel verwendet dieser Algorithmus einen am SINR Modell orientierten Interferenzbegriff. Dieser Algorithmus besitzt mehrere für den praktischen Einsatz wertvolle Eigenschaften: Er ist verteilt berechenbar und kommt ohne jegliche Entfernungs- oder Richtungsinformationen aus. Jeder Netzknoten muss lediglich die Leistungen der einkommenden Signale messen und hat dadurch bereits alle benötigten Informationen. Hierdurch werden automatisch Effekte berücksichtigt, die die Sendeleistung beeinflussen. Dies ist bei den meisten bekannten Verfahren zum Topology Control nicht der Fall.

Selbstverständlich lässt sich das Problem, eine effizient planbare Topologie zu bestimmen, auch optimal lösen. Hierzu wird ein Constraint Program beschrieben. Aufgrund der Komplexität des Problems ist dieses leider nur für kleine Eingabegrößen nutzbar.

Neben den beschriebenen Methoden zur Topologiebestimmung wurden noch einige Standardalgorithmen des Topology Control implementiert. Die einzelnen Verfahren wurden experimentell anhand zahlreicher Qualitätsmaße für Netztopologien, und unter Verwendung der in dieser Arbeit vorgestellten Methoden, miteinander verglichen. Hierbei stellte sich heraus, dass viele der Methoden trotz teils relativ unterschiedlicher Ansätze im Ergebnis sehr ähnlich zueinander sind.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] S. Agmon. The relaxation method for linear inequalities. *Canad. J. Math.*, 6:382–392, 1954.

[2] Patrik Björklund, Peter Värbr, and Di Yuan. A column generation method for spatial TDMA scheduling in ad hoc networks. *Ad Hoc Networks*, 2:2004, 2004.

[3] Martin Burkhart, Pascal von Rickenbach, Roger Wattenhofer, and Aaron Zollinger. Does topology control reduce interference? In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 9–19, New York, NY, USA, 2004. ACM.

[4] A. Chandrakasan, R. Amirtharajah, Seonghwan Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang. Design considerations for distributed microsensor systems. *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, pages 279–286, 1999.

[5] *ILOG CPLEX: High-performance software for mathematical programming and optimization.* http://www.ilog.com/products/cplex/.

[6] Reinhard Diestel. *Graph Theory (Third Edition)*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 2005.

[7] *GECODE: Generic Constraint Development Environment.* http://www.gecode.org.

[8] *GECODE Benchmarks.* http://www.gecode.org/benchmarks.html.

[9] Carla P. Gomes and Bart Selman. Problem structure in the presence of perturbations. In *In Proceedings of the 14th National Conference on AI*, pages 221–226. AAAI Press, 1997.

[10] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 431–437, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[11] Olga Goussevskaia, Yvonne Anne Oswald, and Roger Wattenhofer. Complexity in geometric SINR. In *MobiHoc '07: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 100–109, New York, NY, USA, 2007. ACM.

[12] S.A. Grandhi, R. Vijayan, and D.J. Goodman. Distributed power control in cellular radio systems. *IEEE Transactions on Communications*, 42(234):226–228, Feb/Mar/Apr 1994.

[13] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[14] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, pages 607–615. Morgan Kaufmann, 1995.

[15] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks. In *Proceedings of the International Conference on Networked Sensing Systems (INSS)*, Tokyo, Japan, June 2004.

[16] J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, Sep 1992.

[17] *Java universal network/graph framework (JUNG)*. `http://jung.sourceforge.net`.

[18] Joseph M. Kahn, Randy H. Katz, and Kristofer S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *MOBICOM*, pages 271–278, 1999.

[19] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM.

[20] Bastian Katz, Markus Völker, and Dorothea Wagner. Link Scheduling in Local Interference Models. In *Proceedings of the 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2008. to appear.

[21] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[22] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Computer Science, Hanover, NH, July 2003.

[23] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 24–33, New York, NY, USA, 2002. ACM.

[24] Li Li, Joseph Y. Halpern, Paramvir Bahl, Yi-Min Wang, and Roger Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 264–273, New York, NY, USA, 2001. ACM.

[25] Li Li, Joseph Y. Halpern, Paramvir Bahl, Yi-Min Wang, and Roger Wattenhofer. A cone-based distributed topology-control algorithm for wireless multi-hop networks. *IEEE/ACM Trans. Netw.*, 13(1):147–159, 2005.

[26] Ning Li and J.C. Hou. Localized topology control algorithms for heterogeneous wireless networks. *Networking, IEEE/ACM Transactions on*, 13(6):1313–1324, Dec. 2005.

[27] Ning Li, Jennifer C. Hou, and Lui Sha. Design and analysis of an mst-based topology control algorithm. In *INFOCOM*, 2003.

[28] Thomas Locher, Pascal von Rickenbach, and Roger Wattenhofer. Sensor networks continue to puzzle: Selected open problems. In Shrisha Rao, Mainak Chatterjee, Prasad Jayanti, C. Siva Ram Murthy, and Sanjoy Kumar Saha, editors, *ICDCN*, volume 4904 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 2008.

[29] Clyde Monma and Subhash Suri. Transitions in geometric minimum spanning trees (extended abstract). In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*, pages 239–249, New York, NY, USA, 1991. ACM.

[30] T. Moscibroda and R. Wattenhofer. The complexity of connectivity in wireless networks. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–13, April 2006.

[31] Thomas Moscibroda, Roger Wattenhofer, and Aaron Zollinger. Topology control meets sinr:: the scheduling complexity of arbitrary topologies. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 310–321, New York, NY, USA, 2006. ACM Press.

[32] Tim Nieberg, Stefan Dulman, Paul Havinga, Lodewijk van Hoesel, and Jian Wu. Collaborative algorithms for communication in wireless sensor networks. pages 271–294, 2003.

[33] LAN MAN Standards Committee of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specification. *IEEE Std 802.11-1997*, November 1997.

[34] Ravi Prakash. Unidirectional links prove costly in wireless ad hoc networks. In *DIALM '99: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 15–22, New York, NY, USA, 1999. ACM.

[35] Lawrence G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, 1975.

[36] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[37] Stefan Schmid and Roger Wattenhofer. Algorithmic models for sensor networks. In *IPDPS*. IEEE, 2006.

[38] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, June 1998.

[39] Yao Shen, Yunze Cai, and Xiaoming Xu. Localized interference-aware and energy-conserving topology control algorithms. *Wirel. Pers. Commun.*, 45(1):103–120, 2008.

[40] Steven Skiena. *Implementing discrete mathematics: combinatorics and graph theory with Mathematica*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

[41] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, 1984.

[42] H. Tamura, K. Watanabe, M. Sengoku, and S. Shinoda. On a new edge coloring related to multihop wireless networks. *Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on*, 2:357–360 vol.2, 2002.

[43] Jian Tang, Guoliang Xue, Christopher Chandler, and Weiyi Zhang. Link scheduling with power control for throughput enhancement in multihop wireless networks. In *IEEE Transactions on Vehicular Technology*, volume 55, pages 733–742, Washington, DC, USA, 2006. IEEE Computer Society.

[44] F. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part II–The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975.

[45] P. von Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8 pp.–, April 2005.

[46] Markus Völker. (Lokales) Scheduling von Übertragungen in Sensornetzen im SINR-Modell. Student research paper, Universität Karlsruhe, January 2008.

[47] Dorothea Wagner and Roger Wattenhofer, editors. *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[48] Roger Wattenhofer and Aaron Zollinger. XTC: A Practical Topology Control Algorithm for Ad-Hoc Networks. In *4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN), Santa Fe, New Mexico*, April 2004.

[49] *Xpress-MP: suite of mathematical modeling and optimization tools*. http://www.dashoptimization.com.